TECHNISCHE HOCHSCHULE LÜBECK

Programmieren II

Informatik (B. Sc.)
2. Semester

Unit 09

Einführung in Java Swing

Prof. Dr. Nane Kratzke

Units

Rekursion

01 Einführung in die rekursive Programmierung
02 Sequenzbasierte Rekursionen
03 Rekursive Datenstrukturen

Funktionale Programmierung

04 Einführung in die funktionale Programmierung
05 Funktionale Programmierung mit Streams
06 Thinking in Filter → Map → Reduce

Generische Datentypen <T>

07 Einführung in Gen. Datentypen (Type Erasure)
08 Bounded Types

Graphical User Interfaces

09 Einführung in Swing (Typvertreter)
10 Model - View - Controller (MVC)
11 MVC an einer Beispielanwendung

TECHNISCHE HOCHSCHULE LÜBECK

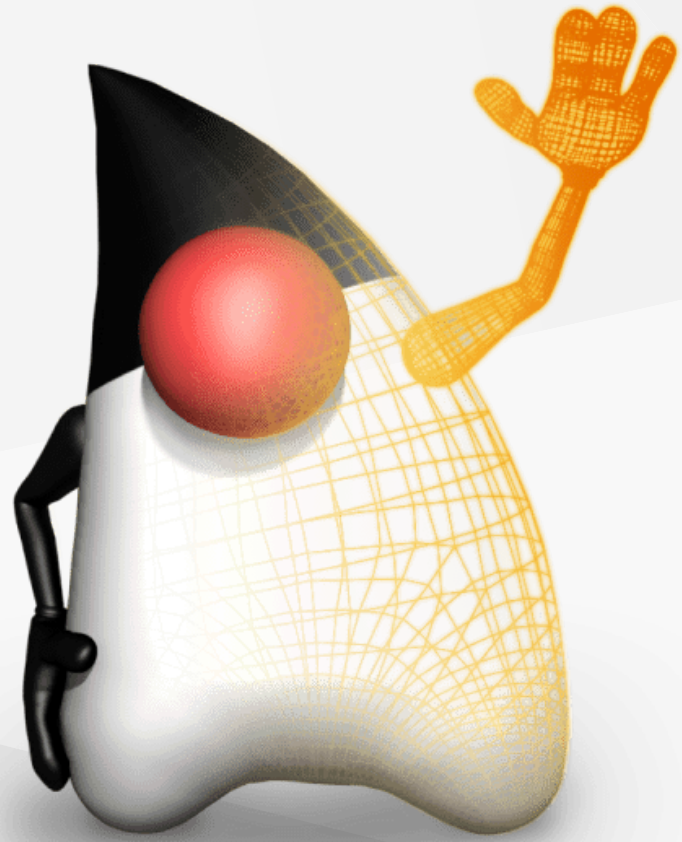
Unit 09

Einführung in Swing

Inhalte:

- Das GUI Framework Swing
- Bedienelemente
- Anordnung von Bedienelementen (Layout Manager)
- Reaktion auf durch Bedienelemente ausgelöste Ereignisse (Event Handling)

Übungsaufgabe (Ascii-Art Editor)



Einleitung

Grafische Benutzeroberflächen

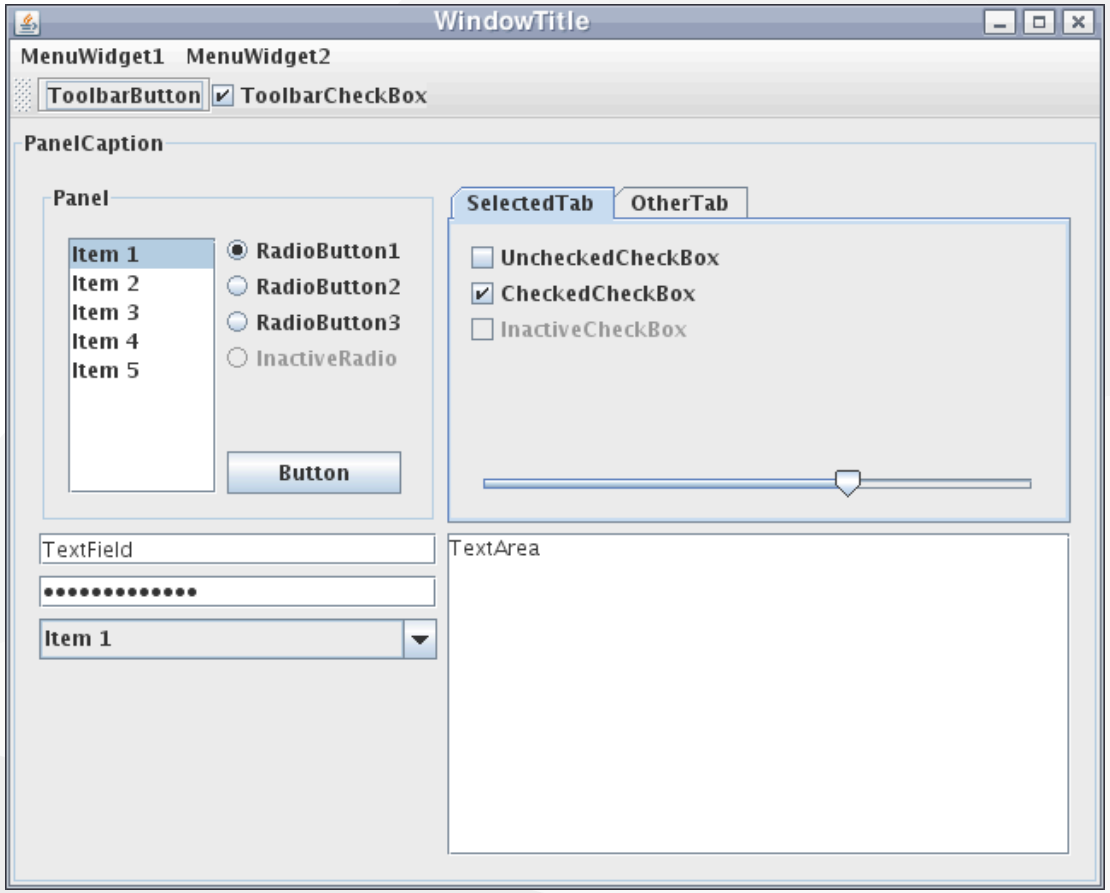
Bedeutung von grafischen Benutzeroberflächen (GUIs)

GUIs:

- Grafische Benutzeroberflächen bieten eine interaktive Möglichkeit, mit Softwareanwendungen zu interagieren.
- Erleichtern die Bedienung von Anwendungen durch visuelle Komponenten anstelle von reinen Textbefehlen.
- SWING als GUI-Bibliothek: Entwickelt zur Erstellung **plattformunabhängiger** grafischer Benutzeroberflächen in Java.

Übliche Funktionalitäten:

- Unterstützung für komplexe **Komponenten** und **Layouts**.
- **Event-gesteuert**: Benutzerinteraktionen (z.B. Mausklicks) lösen Ereignisse aus, die durch Event-Handler verarbeitet werden.
- Objektorientierte Modellierung: Grafische Objekte in SWING lassen sich ideal in der objektorientierten Programmierung darstellen und verwalten.
- Anordnung durch **Layoutmanager**: Flexibles Layout-Management ermöglicht die Gestaltung ansprechender Benutzeroberflächen.

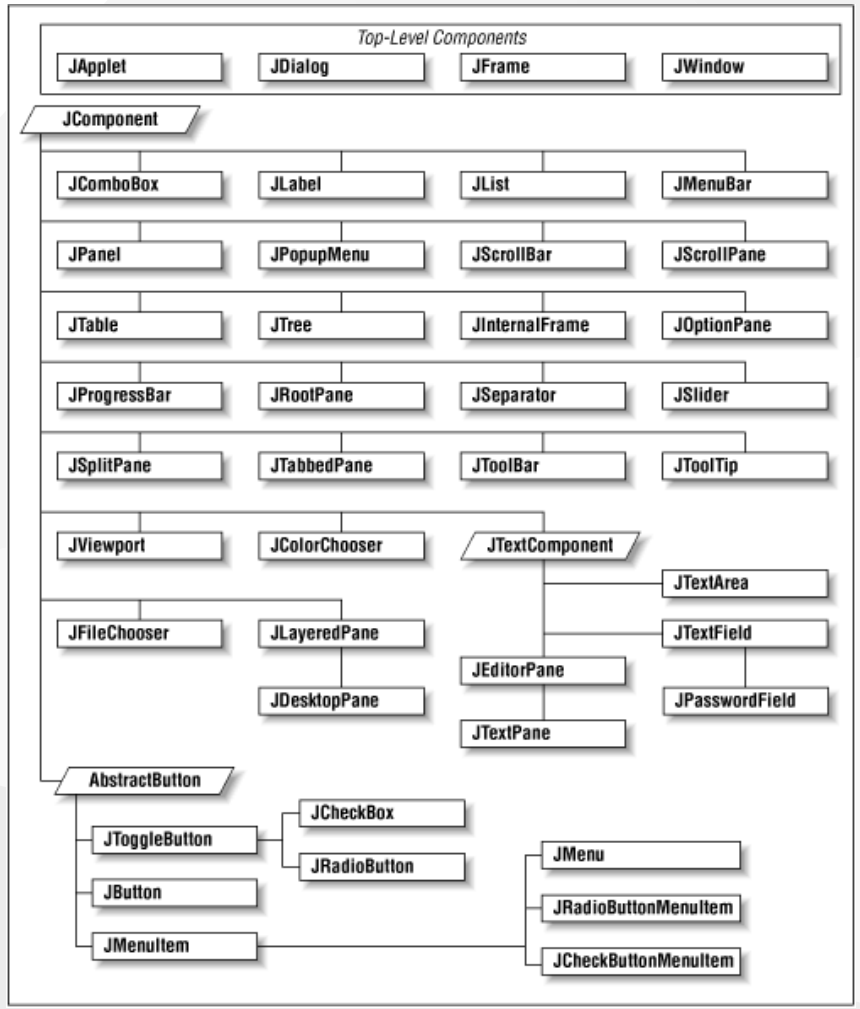


Bildquelle: Wikipedia

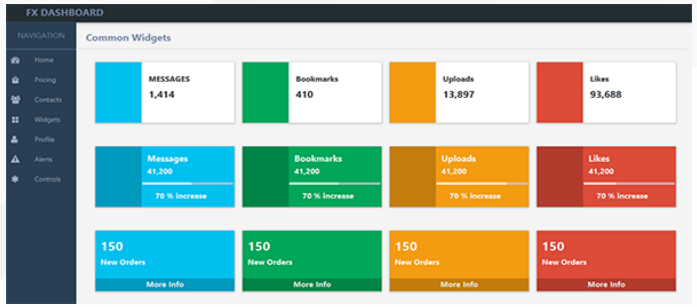
Ziel der Einführung in Swing

Verständnis der grundlegenden Konzepte zur Erstellung grafischer Benutzeroberflächen (GUIs) mit der Swing-Bibliothek.

- Vertrautmachen mit Swing: Einführung in die grundsätzliche Struktur und Funktionalität der Swing-Bibliothek.
- Erstellung von GUI-Komponenten:
  - Kennenlernen der verschiedenen GUI-Komponenten (z.B. JButton, JTextField, JLabel).
  - Einsatz von Layout-Managern zur Anordnung von Komponenten in Fenstern.
- Ereignisgesteuerte Programmierung:
  - Implementierung von Event-Handling zur Reaktion auf Benutzerinteraktionen (z.B. Mausklicks).
  - Verwendung von Lambda-Ausdrücken für eine effiziente Ereignisbehandlung.
- Überblick über Alternativen:
  - Verständnis der Evolution von GUI-Bibliotheken in Java, mit Ausblick auf JavaFX als zukünftigen Standard.
  - Ziel ist es, die Studierenden in die Lage zu versetzen, eigenständig GUIs mit Swing zu entwickeln und ein solides Fundament für die Programmierung mit anderen GUI-Frameworks zu schaffen.

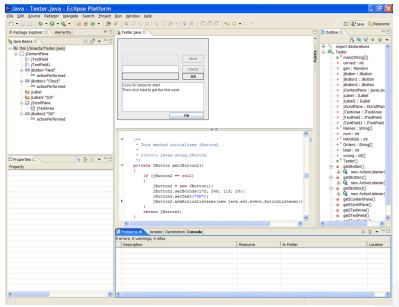


Überblick über weitere GUIs in Java



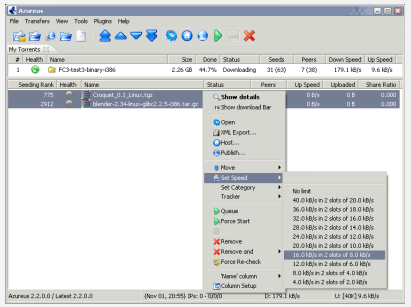
JavaFX

- Modernes Framework zur Erstellung von GUIs
- Unterstützt CSS zur Gestaltung und FXML für Layouts
- Entkoppelt von Swing und AWT
- Erfordert allerdings HTML/CSS Kenntnisse



JFace

- Teil der Eclipse Rich Client Platform (RCP)
- Entwickelt für die Erstellung von Desktop-Anwendungen auf Basis von SWT
- Bietet Erweiterungen und Unterstützung für die SWT-Bibliothek



SWT (Standard Widget Toolkit)

- Alternativ-Toolkit zur AWT und Swing, entwickelt von Eclipse
- Ggf. bekannt von der IDE Eclipse
- Bietet native GUI-Komponenten, die auf Betriebssysteme abgestimmt sind
- Besser geeignet für Anwendungen, die ein natives Look-and-Feel benötigen

Anmerkung:
Wir nutzen Swing ein, da es eine einfachere Grundlage bietet und dennoch ermöglicht, die essenziellen Konzepte der Benutzerinteraktion zu verstehen, ohne sich sofort in die komplexeren und umfangreicheren Frameworks wie JavaFX vertiefen zu müssen.

Die Swing-Bibliothek

Plattformunabhängiges GUI-Framework mit
schwer- und leichtgewichten Komponenten

JFrame • JDialog • JButton • JTextField • JTextArea • ...

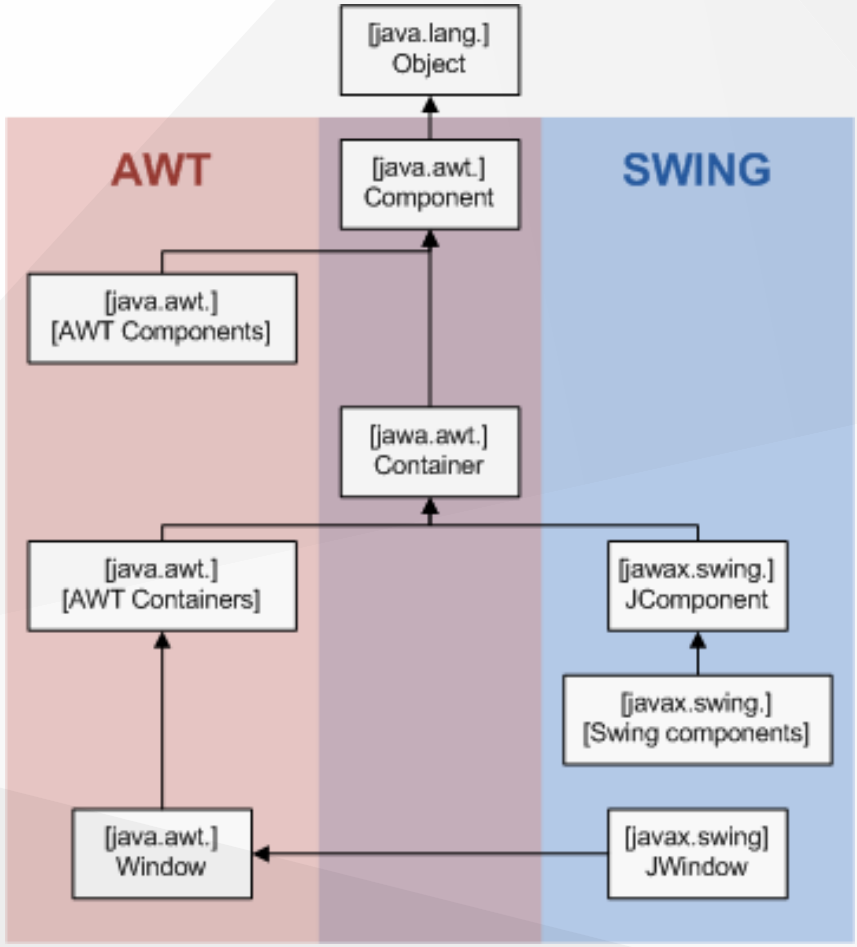
Struktur der Swing-Bibliothek

Schwergewichtige Komponenten: Nutzen die native grafische Schnittstelle des Betriebssystems zur Darstellung.

- Betriebssystem-spezifisch
- Inkonstantes Aussehen und Verhalten auf verschiedenen Systemen
- Beispiele: `java.awt.Button`, `java.awt.Label`

Leichtgewichtige Komponenten: Vollständig in Java implementiert, unabhängig von der nativen Grafikschnittstelle des Betriebssystems.

- Konsistentes Aussehen und Verhalten plattformübergreifend
- Effektive Ereignisbehandlung
- Beispiele: `javax.swing.JButton`, `javax.swing.JTextField`, `javax.swing.JPanel`



Grundlegende Komponenten

JFrame

```java
import javax.swing.*; // <== Erforderlicher Swing-Import

public class HelloWorldFrame extends JFrame {

    public HelloWorldFrame() {
        super("Hello World Frame"); // Setzt den Fenstertitel
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Schließt die Anwendung beim Schließen des Fensters

        setSize(400, 200); // Setzt die Fenstergröße (Breite, Höhe)

        JLabel label = new JLabel("Hello, World!", SwingConstants.CENTER);
        // Erzeugt ein Label mit dem Text
        add(label); // Fügt das Label zum Frame hinzu

        setVisible(true);
        // Macht das Fenster sichtbar. Muss immer die letzte Operation sein.
    }

    public static void main(String[] args) {
        new HelloWorldFrame(); // Erzeugt eine Instanz des Fensters
    }
}
```

Ein `JFrame` ist ein grafisches Fenster in Java Swing, das eine Titelleiste, einen Rahmen und die Möglichkeit bietet, andere GUI-Komponenten zu enthalten und darzustellen.

JTabbedPane

```java
import javax.swing.*; // <== Erforderlicher Swing-Import

public class TabbedPaneExample extends JFrame {

    public TabbedPaneExample() {
        super("TabbedPane Example");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP); // Erstellt JTabbedPane

        // Erstellen von Labels für die Tabs
        JLabel helloLabel = new JLabel("Hello", SwingConstants.CENTER);
        JLabel worldLabel = new JLabel("World", SwingConstants.CENTER);

        // Fügt die Labels in JScrollPanes ein
        JScrollPane helloScrollPane = new JScrollPane(helloLabel);
        JScrollPane worldScrollPane = new JScrollPane(worldLabel);

        // Fügt Tabs hinzu
        tabbedPane.addTab("Tab 1", helloScrollPane); // Erster Tab
        tabbedPane.addTab("Tab 2", worldScrollPane); // Zweiter Tab

        // Fügt das TabbedPane dem Fenster hinzu
        add(tabbedPane);

        setSize(400, 200);
        setVisible(true);
    }

    public static void main(String[] args) {
        new TabbedPaneExample();
    }
}
```

Ein JTabbedPane ist ein Swing-Container, der mehrere "Karten" (Tabs) ermöglicht, die übereinander gestapelt sind und abwechselnd angezeigt werden können, um verschiedene Inhalte in einem einzigen Fenster zu organisieren.

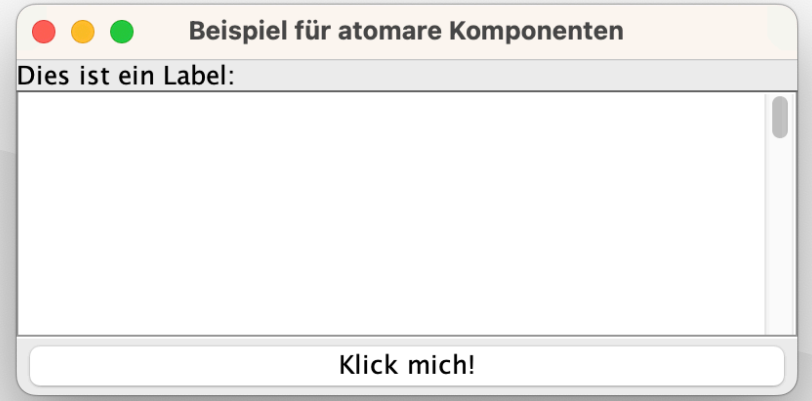
Atomare Komponenten

JLabel, JButton, JTextField, JTextArea

```java
public class ComponentsExample {
    public static void main(String[] args) {
        // Erstelle ein neues JFrame
        JFrame frame = new JFrame("Beispiel für atomare Komponenten");

        // Erstelle atomare Komponenten
        JButton button = new JButton("Klick mich!");
        JLabel label = new JLabel("Dies ist ein Label:");
        JTextArea textArea = new JTextArea(40, 20);
        textArea.setLineWrap(true);
        textArea.setWrapStyleWord(true);

        // Füge die Komponenten zum Frame hinzu
        frame.add(label, BorderLayout.NORTH);
        frame.add(new JScrollPane(textArea), BorderLayout.CENTER);
        frame.add(button, BorderLayout.SOUTH);

        // Konfiguriere das JFrame
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400 , 200);
        frame.setVisible(true);
    }
}
```

Atomare Komponenten sind visuelle Bedienelemente, die selbst keine weiteren Unterkomponenten enthalten. Beispielsweise:

- JLabel: Zeigt nicht editierbaren Text. Wird häufig zur Beschreibung anderer Komponenten verwendet.
- JTextArea: Ein mehrzeiliges Textfeld, das es ermöglicht, längere Texte einzugeben.
- JScrollPane: Ermöglicht das Scrollen von anderen Komponenten.
- JButton: Eine interaktive Schaltfläche, die Aktionen bei einem Klick auslöst.



Layout Manager

Automatische Positionierung von Bedienelementen

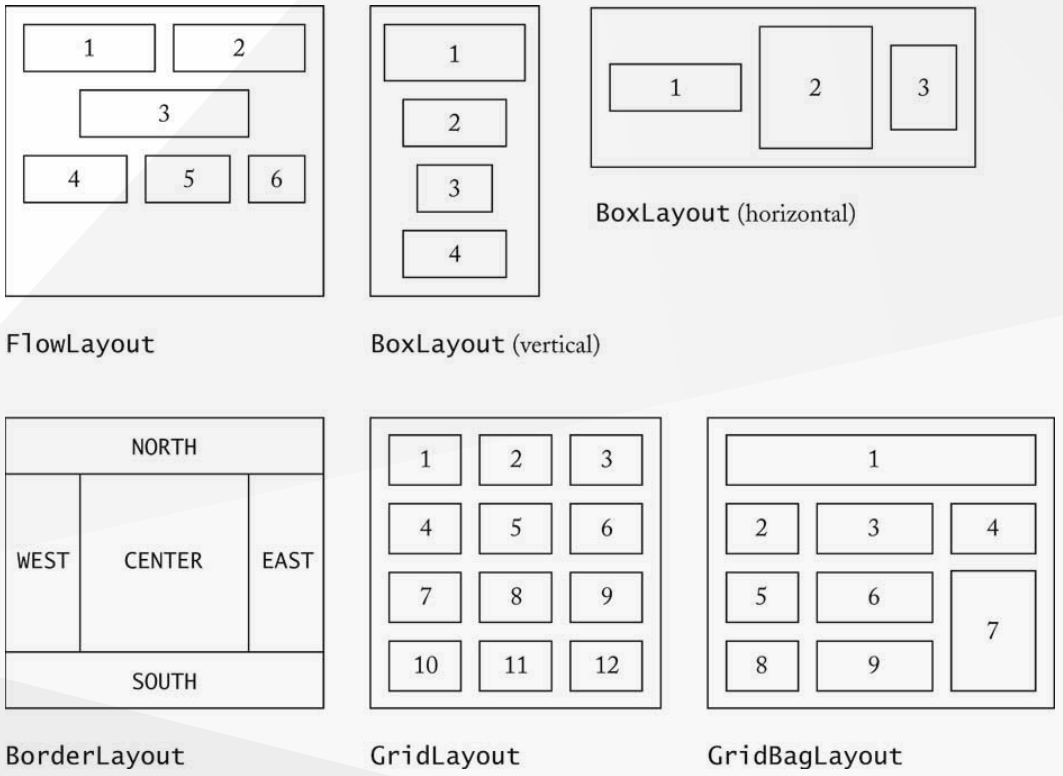
GridLayout • FlowLayout • BorderLayout • GridBagLayout • BoxLayout &bulett; ...

Layout-Management in Swing

Layout Manager ermöglichen es Entwicklern Bedienelemente, wie z. B. Buttons, Textfelder und andere GUI-Komponenten in einem Container (z. B. einem Fenster oder einem Panel) zu gruppieren und anzuordnen.

1. Automatische Anordnung: Layout Manager kümmern sich automatisch um die Positionierung und Anordnung von Komponenten, sodass sie responsiv und dynamisch auf verschiedene Fenstergrößen reagieren können.
2. Größenverwaltung: Layout Manager berücksichtigen die bevorzugten, maximalen und minimalen Größen von Komponenten, was die Anpassungsfähigkeit und Benutzerfreundlichkeit verbessert.
3. Benutzerdefinierte Layouts: Es können eigene Layout Manager implementiert werden, indem die `LayoutManager` -Schnittstelle implementiert wird *(obwohl dies aufgrund der Komplexität nicht empfohlen wird, da die eingehenden Layout-Manager meist mächtig genug sind)*.

Layout-Manager in Swing

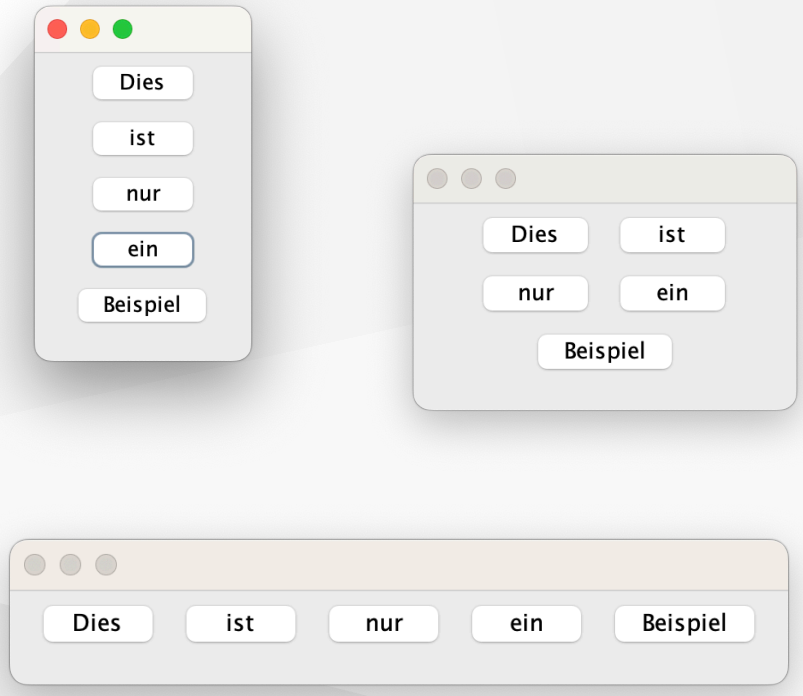


Bildquelle: Object-Oriented Design and Patterns (2nd Edition), Cay S. Horstmann, Wiley, 2005.

Beispiel: FlowLayout Manager

```java
public class FlowLayoutExample extends JFrame {
    public FlowLayoutExample() {
        // Setze das Layout des Panels auf FlowLayout
        JPanel pane = new JPanel();
        pane.setLayout(new FlowLayout());

        // Füge die Buttons hinzu
        pane.add(new JButton("Dies"));
        pane.add(new JButton("ist"));
        pane.add(new JButton("nur"));
        pane.add(new JButton("ein"));
        pane.add(new JButton("Beispiel"));

        // Füge das Panel zum Frame hinzu
        add(pane);

        // Setze die Größe des Fensters
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args) {
        new FlowLayoutExample();
    }
}
```

Komponenten werden der Reihe nach von links nach rechts in einer Zeile anordnet und ggf. in eine neue Zeile umgebrochen.



Abhängig von der Breite des Fensters werden die Element früher oder später umgebrochen.

Beispiel: BorderLayout Manager

```java
public class BorderLayoutExample extends JFrame {
    public BorderLayoutExample() {
        // Setze das Layout des Panels auf BorderLayout
        JPanel pane = new JPanel();
        pane.setLayout(new BorderLayout());

        // Füge die Buttons hinzu
        pane.add(new JButton("Dies"), BorderLayout.NORTH);
        pane.add(new JButton("ist"), BorderLayout.WEST);
        pane.add(new JButton("nur"), BorderLayout.CENTER);
        pane.add(new JButton("ein"), BorderLayout.EAST);
        pane.add(new JButton("Beispiel"), BorderLayout.SOUTH);

        // Füge das Panel zum Frame hinzu
        add(pane);

        // Setze die Größe des Fensters
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }

    public static void main(String[] args) {
        new BorderLayoutExample();
    }
}
```

Anordnung von bis zu fünf Komponenten an festgelegten Positionen:

- **Norden** (oben, volle Breite, minimale Höhe)
- **Süden** (unten, volle Breite, minimale Höhe)
- **Osten** (rechts, volle Höhe, minimale Breite)
- **Westen** (links, volle Höhe, minimale Breite)
- **Zentrum** (zentral, maximale Höhe, maximale Breite)

Dieses Layout ist von vielen Websites und Programmen als Grundlayout bekannt (Kopfzeile, linke und rechte Seitenleiste, Statuszeile und zentraler Contentbereich).

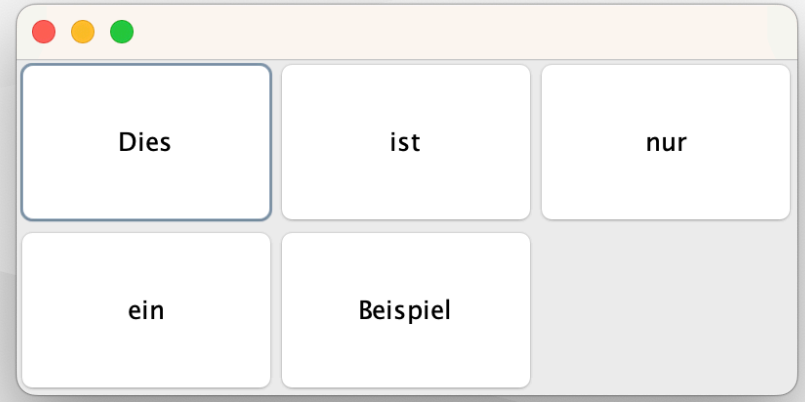
Beispiel: GridLayout Manager

```
public class GridLayoutExample extends JFrame {
  public GridLayoutExample() {
    // Setze das Layout des Panels auf GridLayout
    JPanel pane = new JPanel();
    pane.setLayout(new GridLayout(2, 3)); // 2 Zeilen, 3 Sp.

    // Füge die Buttons hinzu
    pane.add(new JButton("Dies"));
    pane.add(new JButton("ist"));
    pane.add(new JButton("nur"));
    pane.add(new JButton("ein"));
    pane.add(new JButton("Beispiel"));

    // Füge das Panel zum Frame hinzu
    add(pane);

    // Setze die Größe des Fensters
    setSize(400, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }

  public static void main(String[] args) {
    new GridLayoutExample();
  }
}
```

Mit GridLayout können Komponenten in einem flexiblen Raster anordnet werden, wobei jede Komponente gleich groß ist und die Größe sich nach dem größten Element orientiert.

Die Komponenten werden von oben links beginnend mit jedem add() von links nach rechts, Zeile für Zeile im Grid platziert.

Nicht genutzte Plätze im Grid bleiben leer.



Panels und Nested Layouts

Unter Nested Layouts versteht man, mehrere Layout-Manager zu kombinieren, indem man mehrere Panel-Objekte ineinander verschachtelt. So können unterschiedliche Layout Manager für unterschiedliche Bereiche einer Oberfläche angewendet werden.

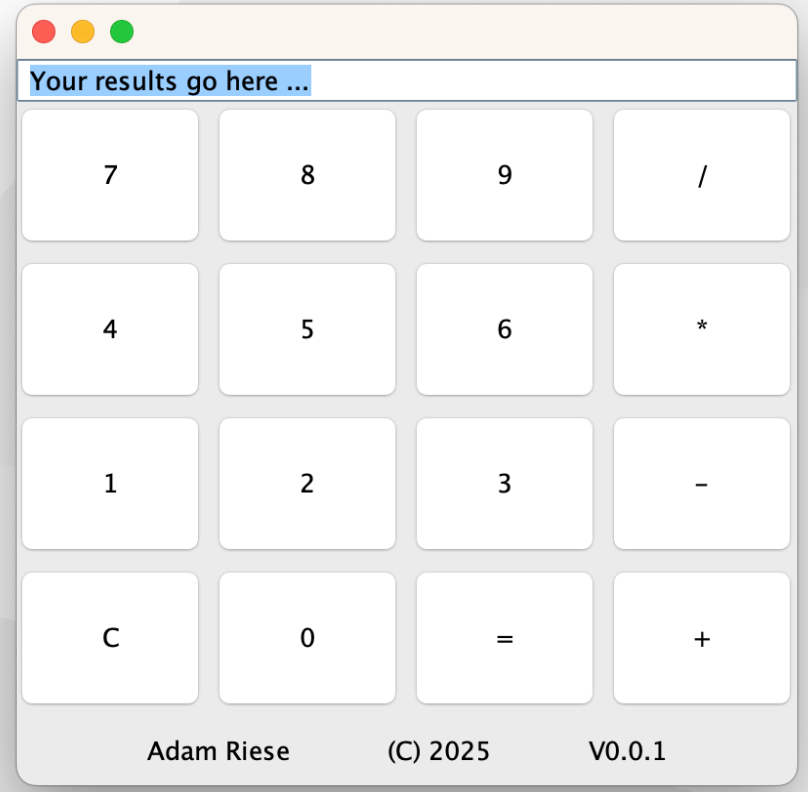
- Ein `JPanel` dient als Container für andere grafische Komponenten.
- Panels können in andere Container eingefügt und miteinander verschachtelt werden, was es ermöglicht, komplexe Benutzeroberflächen zu erstellen.
- Die Verwendung von Borders um das Panel kann man in Nested Layouts visuell gliedern. Zum Beispiel kann ein `TitledBorder` verwendet werden, um dem Panel einen Titel zu geben.

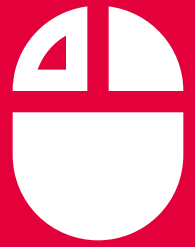
```
JPanel outer = new JPanel();
outer.setLayout(new GridLayout(2, 1)); // 2 Zeilen
outer.setBorder(new TitledBorder("Outer GridLayout"));
add(outer);

JPanel inner_top = new JPanel();
inner_top.setBorder(new TitledBorder("FlowLayout"));
inner_top.setLayout(new FlowLayout());
inner_top.add(new JLabel("| Dies ist mal wieder nur"));
inner_top.add(new JLabel("| so ein"));
inner_top.add(new JLabel("| sinnfreier"));
inner_top.add(new JLabel("| Text, der hier"));
inner_top.add(new JLabel("| alles"));
inner_top.add(new JLabel("| erklären soll"));
outer.add(inner_top);

JPanel inner_bottom = new JPanel();
inner_bottom.setBorder(new TitledBorder("BorderLayout"));
inner_bottom.setLayout(new BorderLayout());
inner_bottom.add(new JButton("N"), BorderLayout.NORTH);
inner_bottom.add(new JButton("S"), BorderLayout.SOUTH);
inner_bottom.add(new JLabel("East"), BorderLayout.EAST);
inner_bottom.add(new JLabel("West"), BorderLayout.WEST);
outer.add(inner_bottom);
```

Beispiel: Kombination von Layout Managern

```
public class CombinedLayouts extends JFrame {
  public CombinedLayouts() {
    setLayout(new BorderLayout()); // BorderLayout für Hauptfenster

    // Ergebnisfeld direkt als atomare Komponente
    this.add(new TextField("Your results ..."), BorderLayout.NORTH);

    // Erstelle ein Tastenfeld mit GridLayout
    JPanel tastenfeld = new JPanel();
    tastenfeld.setLayout(new GridLayout(4, 4, 5, 5));

    for (String t : "789/456*123-C0=+".split("")) {
        tastenfeld.add(new JButton(t));
    }

    // Erstelle eine Statusleiste mit GridLayout
    JPanel fuss = new JPanel();
    fuss.setLayout(new FlowLayout(FlowLayout.CENTER, 50, 10));
    fuss.add(new JLabel("Adam Riese"));
    fuss.add(new JLabel("(C) 2025"));
    fuss.add(new JLabel("V0.0.1"));

    this.add(tastenfeld, BorderLayout.CENTER);
    this.add(fuss, BorderLayout.SOUTH);

    // [...]
  }
}
```

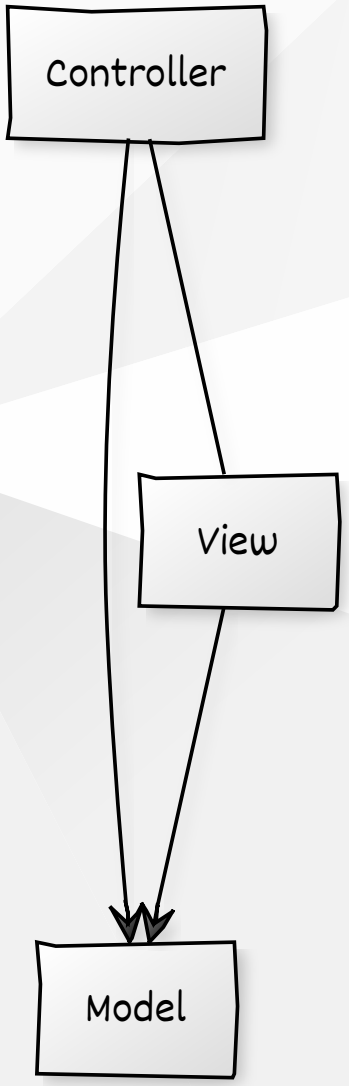
Auf diese Art kann man beispielsweise eine Oberfläche für einen Taschenrechner gestalten.





Event Handling

Reagieren auf durch Bedienelemente ausgelöste Ereignisse
Model - View - Controller



CREATED WITH YUML

Model - View - Controller

Das Model-View-Controller (MVC) Pattern ist ein weitverbreitetes architektonisches Muster in der Softwareentwicklung, das eine saubere Trennung von Logik, Daten und Benutzeroberfläche ermöglicht. Es organisiert den Code in drei Hauptkomponenten:

1. Model: Repräsentiert die Daten und die Geschäftslogik der Anwendung. Es verwaltet die Daten und die Regeln, die auf diese Daten angewendet werden.
2. View: Stellt die grafische Benutzeroberfläche (UI) dar und zeigt die Daten aus dem Model an. Die UI empfängt auch Benutzereingaben.
3. Controller: Vermittelt zwischen Model und View, bearbeitet Benutzereingaben und aktualisiert das Model oder die View entsprechend.

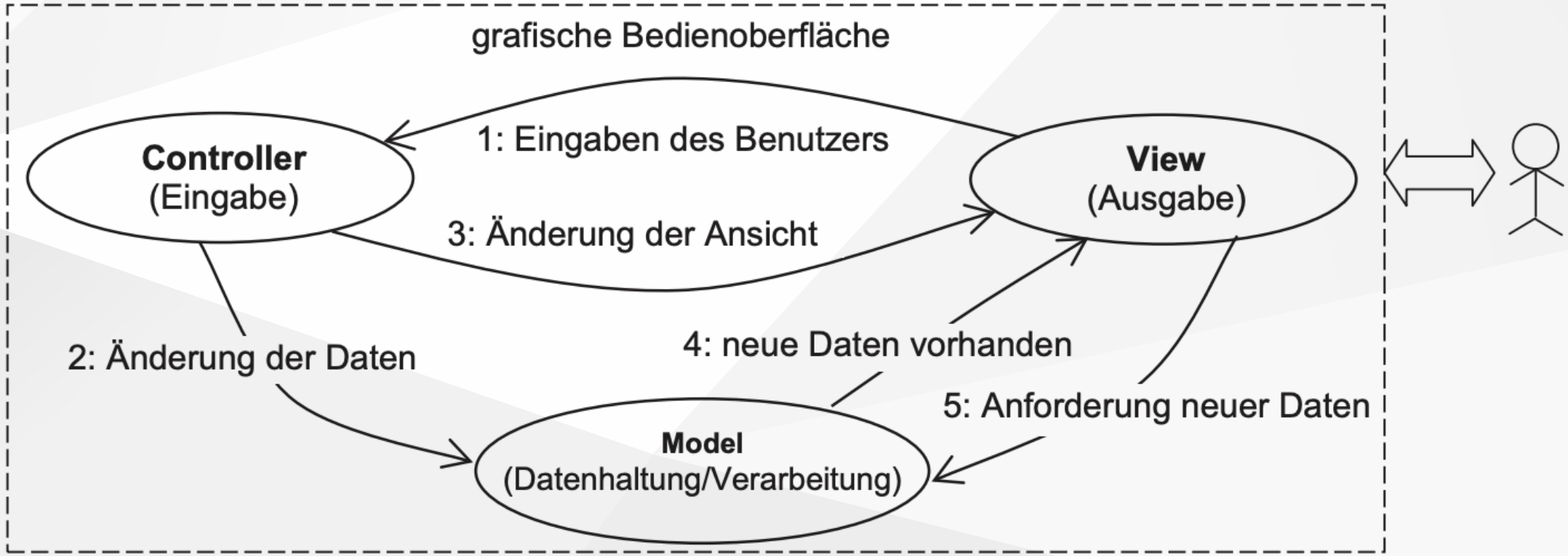
Vorteile von MVC:

- Trennung der Anliegen: Verbesserte Wartbarkeit und Testbarkeit der Anwendung.
- Wiederverwendbarkeit: Die Komponenten (insb. Model) können leichter unabhängig voneinander wiederverwendet und angepasst werden.

Wichtig:

> Die Vorteile von MVC treten nur dann ein, wenn das Model isoliert vom View und Controller ist. D.h. das Model keine Abhängigkeiten zum Code von Controller oder View hat. Man achte auf die Pfeilrichtungen der Assoziationen!

Zusammenarbeit von Model + View + Controller



Bildquelle: Java als erste Programmiersprache, Springer 2016

Ereignisbehandlung in Swing

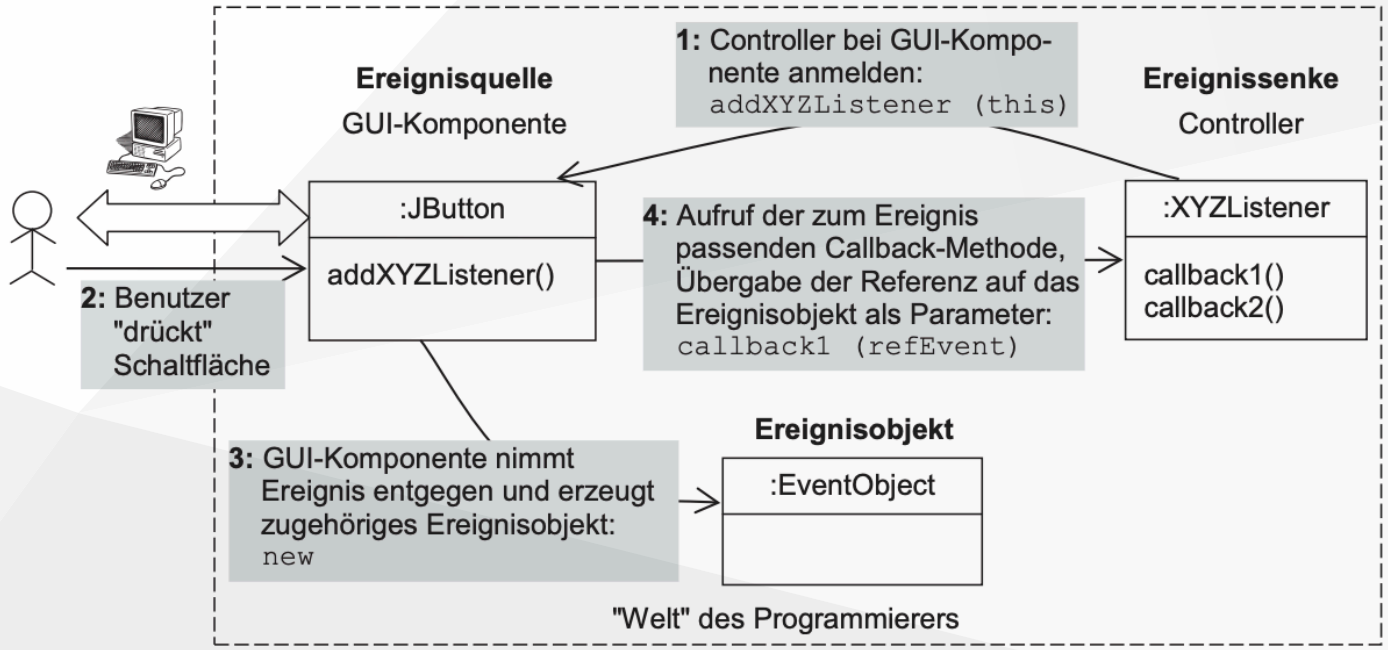
Bei der Interaktion des Benutzers mit der grafischen Bedienoberfläche werden Ereignisse ausgelöst.

Einer GUI-Komponente sind sogenannte Callback-Schnittstellen (`XYZListener`) zugeordnet, die in einem Controller implementiert werden müssen, falls Ereignisse für diese GUI-Komponente verarbeitet werden sollen.

Eine **Ereignissenke** (Listener) meldet sich bei einer **Ereignisquelle** (GUI-Komponente) für spezielle Ereignisse an.

Tritt ein Ereignis auf (z.B. Button-Klick), so wird dies von der **Ereignisquelle** an alle für dieses Ereignis angemeldeten Ereignissenken (Listener) weitergeleitet und dort verarbeitet.

- **Ereignissenke**: ein vordefinierter oder ein selbst geschriebener **Controller** (Listener)
- **Ereignisquelle**: eine **GUI-Komponente** (Views)



Bildquelle: Java als erste Programmiersprache, Springer 2016

Beispiel: Einfaches Event-Handling mit Lambda-Ausdrücken

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

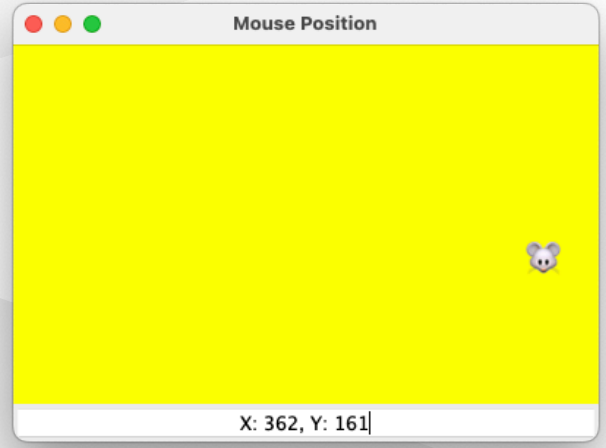
```java
JFrame frame = new JFrame("Cat Clicker");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300, 200);

JTextArea text = new JTextArea();
text.setEditable(false);
text.setLineWrap(true);
frame.add(text, BorderLayout.CENTER);

JButton button = new JButton("Click meow 😈");
frame.add(button, BorderLayout.SOUTH);

// Event-Handler (als Lambda-Funktion)
// ActionListener ist eine funktionale Schnittstelle!
button.addActionListener((ActionEvent e) ->
    text.setText(text.getText() + "😈")
);

frame.setVisible(true);
```

Listener-Schnittstelle mit nur einer Methode:

Lösen Komponenten (z.B. `JButton`) Events aus deren Listener-Schnittstelle (z.B. `ActionListener`) nur eine Methode definieren (z.B. `actionPerformed`), so können diese Events in Lambda-Ausdrücken kompakt behandelt werden, weil solche Listener-Schnittstellen funktionale Schnittstellen sind.

Beispiel: Komplexeres Event-Handling mit anonymen Klassen

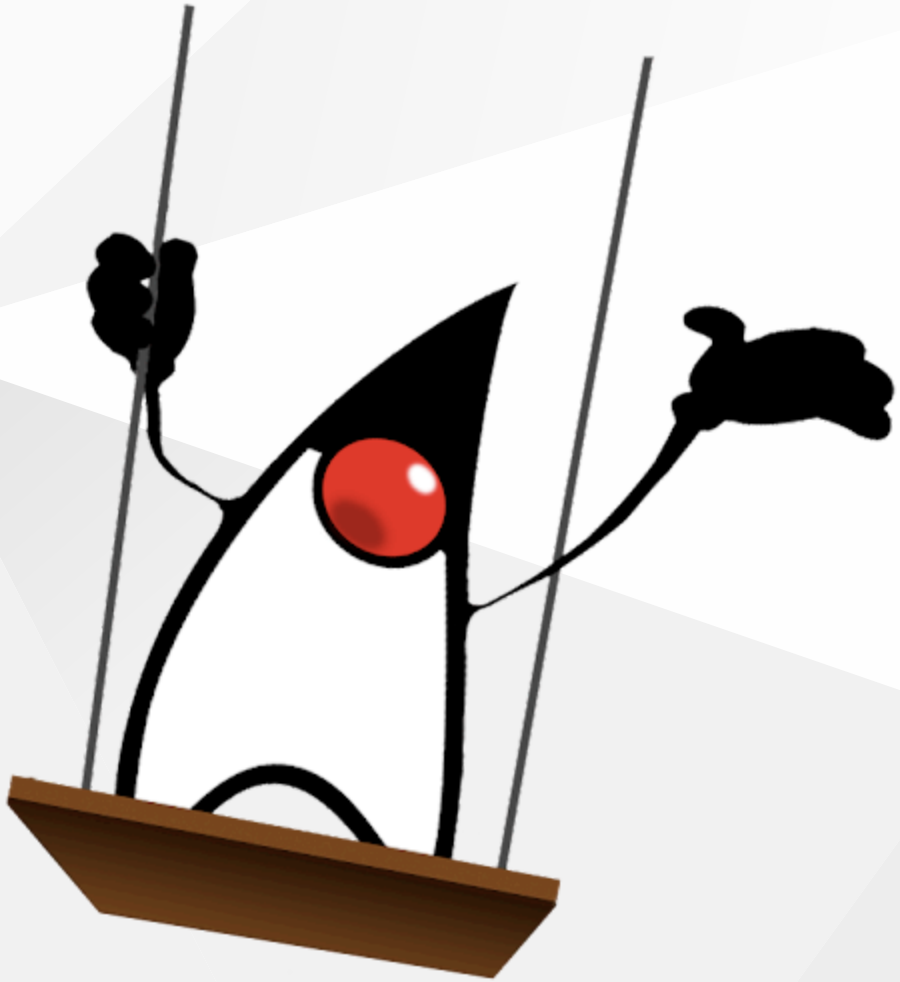
Wenn Listener-Schnittstellen mehr als eine Methode definieren (z.B. `getX()` und `getY()`) kann man nicht mehr mit Lambda-Funktionen arbeiten. Hier bieten sich dann sogenannte Adapter an.

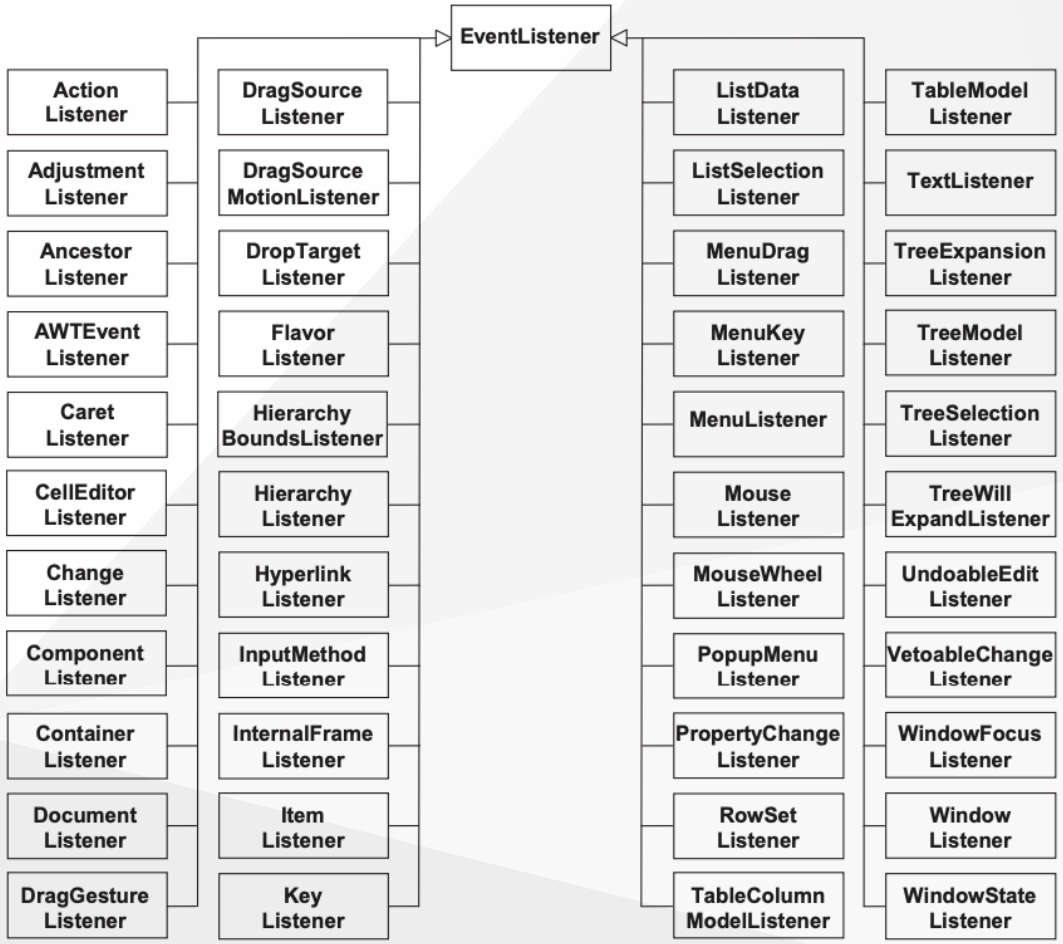



```java
JFrame frame = new JFrame("Mouse Position");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400, 300);
JPanel mousePanel = new JPanel();
mousePanel.setBackground(Color.YELLOW);
frame.add(mousePanel, BorderLayout.CENTER);

JTextField coordinates = new JTextField();
coordinates.setHorizontalAlignment(JTextField.CENTER);
coordinates.setEditable(false);
frame.add(coordinates, BorderLayout.SOUTH);

// Mouse anzeigen!
JLabel mouse = new JLabel("🐭");
mouse.setFont(new Font("Arial", Font.PLAIN, 24));
mousePanel.add(mouse);

// MouseMotionListener hinzufügen
frame.addMouseMotionListener(new MouseMotionAdapter() {
    public void mouseMoved(MouseEvent e) {
        // Die Schnittstelle hat mehr als eine Methode! Kein funktionales IF.
        int x = e.getX(); int y = e.getY();
        int w = mouse.getWidth(); int h = mouse.getHeight();
        coordinates.setText("X: " + x + ", Y: " + y); // Koordinaten anzeigen
        mouse.setBounds(x - w/2, y - h, w, h); // Mouse positionieren
    }
});
frame.setVisible(true);
```

Adapterklassen sind konkrete Klassen, die eine Listener-Schnittstelle mit leeren Methoden implementieren (z.B. `MouseMotionAdapter`). Sie werden hauptsächlich eingesetzt, wenn nur Teile von Methoden einer Schnittstelle implementiert werden (also nur auf einige aber nicht alle Ereignisse reagiert werden soll).

Überblick über Listener Schnittstellen (Eventarten) in Swing





Zu allen Schnittstellen gibt es auch entsprechende Adapterklassen.

Übungsaufgaben

Ein Ascii-Art Editor - Step by Step

1. Aufgabe: UI für einen ASCI Art Editor

Entwickeln Sie bitte ein Java Programm, dass folgende Oberfläche darstellt. Erst einmal nur die UI, ohne Funktion!

Hinweise: Folgende Komponenten wurden für die Vorgabe eingesetzt:

- JMenu (Frag KIRA!)
- JMenuItem (für die Font-Selektion)
- JPanel zur Gliederung
- JLabel (`¯\_(ツ)_/¯`)
- JTextField
- JTextArea
- Nested BorderLayout

```
Color THLRED = new Color(228, 1, 58);
```

2. Aufgabe: Ergänzen der UI um Event Handler

Ergänzen Sie die UI nun um Event Handler, die immer dann die Darstellung der `JTextArea` aktualisieren, wenn eine Änderung im Eingabe-`JTextField` gemacht wurde, oder wenn ein Font im Menu geändert wurde.

Eine Auswahl möglicher Fonts:

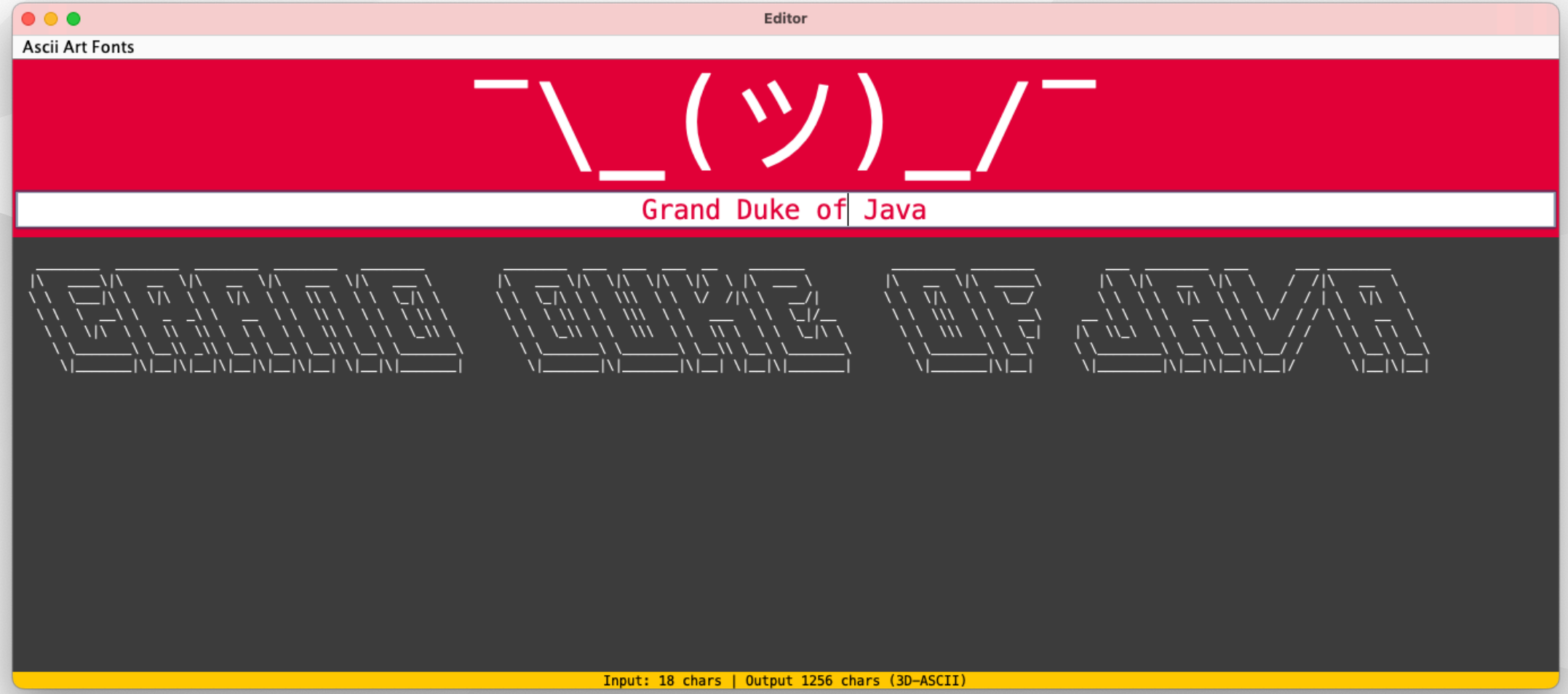
```
public static final String[] FONTS = {
    "3D-ASCII", "Alpha", "Bloody", "Straight",
    "Crazy", "Ghost", "Graffiti",  "Trek"
};
```

Generierung von Ascii Art Schriften:

```
public static String lastFont = "3D-ASCII";

public static String asAsciiArt(String text, String fontStyle) {
  try {
    Charset utf8 = StandardCharsets.UTF_8;
    String base = "https://asciified.thelicato.io/api/v2/ascii";
    String query = String.format("text=%s&font=%s",
      URLEncoder.encode(text, utf8),
      URLEncoder.encode(fontStyle, utf8)
    );
    URL url = new URL(base + "?" + query);
    String art = "\n" + new String((url)
      .openStream().readAllBytes(), utf8);
    return art.replaceAll("\\n", "\n  ");
  } catch (Exception e) {
      return "400 Bad Request: " + e.getMessage();
  } finally lastFont = fontStyle;
}

public static String asAsciiArt(String text) {
  return asAsciiArt(text, lastFont);
}
```

3. Aufgabe: Ergänzen Sie die UI nun um eine Statuszeile

Ergänzen Sie nun eine Statuszeile, die immer die aktuelle Anzahl der Input- und Outputzeichen wie hier gezeigt darstellt.



Zusammenfassung

- Graphical User Interfaces (GUI) *ermöglichen die Interaktion zwischen Benutzern und Softwareanwendungen über eine Oberfläche und der Darstellung visueller Interaktionselemente.*
- SWING *ist eine plattformunabhängige GUI-Bibliothek in Java, die leichtgewichtige Komponenten bietet, um ein einheitliches Erscheinungsbild auf verschiedenen Betriebssystemen zu gewährleisten.*
- Layout Managers *ordnen GUI-Komponenten auf der Oberfläche responsiv in Abhängigkeit von der Fenstergröße an. Gängige Layout-Managern in Swing: `FlowLayout`, `BorderLayout` und `GridLayout`*
- Event Handling *ermöglicht das Reagieren auf Benutzeraktionen, wie Mausklicks oder Tastatureingaben. Listener warten darauf, dass ein Ereignis eintritt, und führen dann eine registriere Methode für ein Ereignis aus. Lambda-Ausdrücke können verwendet werden, um einfachere Event-Handler zu definieren, während Anonyme Klassen für komplexere Ereignisse verwendet werden müssen, die in ihrer Schnittstelle mehr als eine Callback-Methode definieren.*
- Model-View-Controller Pattern (MVC) *ist ein Architekturansatz, der die Trennung von **Daten** und **Geschäftslogik** (Model), der **Präsentation** (View) und der **Benutzerinteraktion** (Controller) fördert.*
  *!!! **Wichtig** !!! Ein Model sollte niemals Abhängigkeiten zum View oder zum Controller eingehen!*

Kontakt

Prof. Dr. Nane Kratzke

Technische Hochschule Lübeck

Mail: nane.kratzke(at)th-luebeck.de

Code Strong!

Alle Icons stammen von iconify.design • Bilder von Pixabay

Lösungsvorschläge zu den Aufgaben 1 bis 3

Aufgrund des Umfangs finden Sie den Lösungsvorschlag ich in diesem Repository.
https://git.mylab.th-luebeck.de/nane.kratzke/prog-2-swing-examples

Sie können dieses Repository wie folgt auf Ihren lokalen Rechner klonen.

```
git clone https://git.mylab.th-luebeck.de/-/ide/project/nane.kratzke/prog-2-swing-examples.git
```

Dieses Projekt enthält auch alle Beispiele, die in der Unit 09 behandelt wurden. Auch den Lösungsvorschlag zum Ascii Art Editor.

Alle Programme sind mittels `java Programmname.java` ausführbar.

Z.B. startet

```
java AsciiArtEditor.java
```

den Ascii Art Editor.