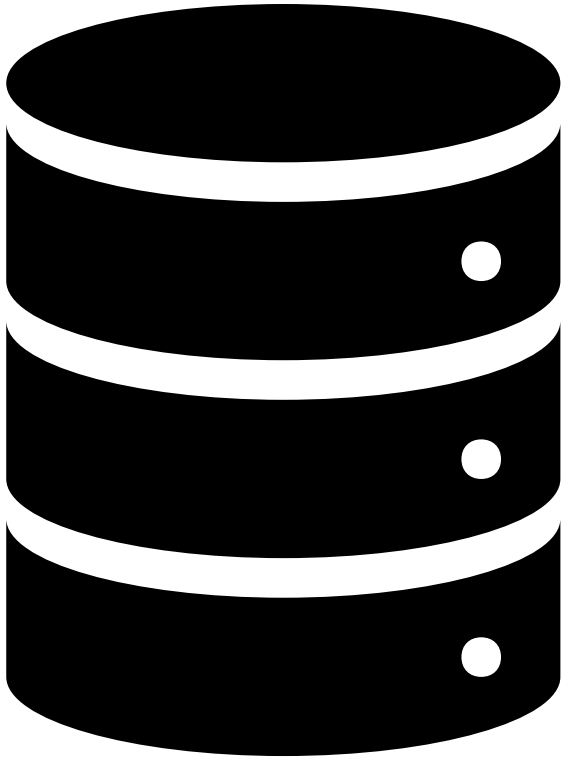


MVC und GUI

März 2025

Plan für die Woche

Montag	Dienstag	Mittwoch	Donnerstag	Freitag
• Aggregate • JDBC • SQL-Injection	• Implementierungs-weisen • BestPractises • MVC-Architektur • ER-Modelle	• Wiederholung	• MVC-Pattern • Grundlagen graphische Oberfläche	• Projekteinleitung

SMART INDUSTRY CAMPUS

Plan für heute

- MVC – Pattern
- Grundlagen graphische Oberfläche

SMART INDUSTRY CAMPUS

MVC-Pattern

SMART INDUSTRY CAMPUS

Einleitung

- MVC-Architektur steht für Model-View-Controller
- ist ein **Entwurfsmuster** in der Softwareentwicklung, das die Strukturierung von Anwendungen erleichtert
- es trennt folgende Hauptkomponenten:
  - Model (Datenstrukturen wie Buch, Autor, ...)
  - View (Benutzeroberfläche)
  - Controller (Steuerung der Benutzeroberfläche und Verarbeitung der Benutzereingaben)
- es gibt verschiedene Abwandlungen des Entwurfsmusters

Modell

- repräsentiert ein Objekt aus der realen Welt (Buch, Autor, Patient,...)
- enthält Logik (Berechnungen, Validierungen, ...)
- ist unabhängig von der graphischen Oberfläche
- bei kleinen Anwendungen werden hier auch die Daten aus der Datenbank abgerufen
  - D.h., dass hier ein Teil der DAO-Schicht liegt
- bei größeren Anwendungen werden DAO- und Modell-Schicht wie zuvor erarbeitet getrennt

View

- ist für die Darstellung der Daten und das User Interface (UI) verantwortlich
  - Daten visuell darstellen
  - Benutzerinteraktionen erfassen (z.B. Klicks, Formulareingaben, …)
- zeigt Informationen an, die vom Model bereitgestellt werden
  - Änderungen vom Model werden empfangen und in der UI aktualisiert

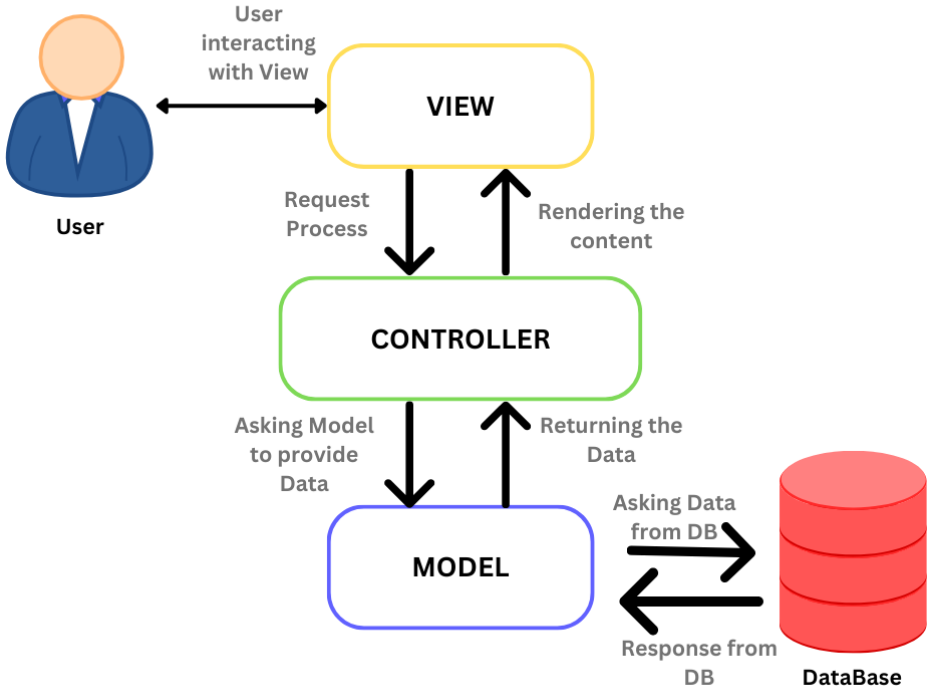
Controller

- Vermittler zwischen Model und View
- verarbeitet Benutzereingaben und entscheidet, wie Model und View darauf reagieren sollen
  - Nimmt Benutzereingaben entgegen
  - Aktualisiert das Model je nach Nutzereingabe
  - Wechselt die View entsprechend der Benutzerinteraktion
- Beispiel:
  - Nutzer klickt auf „Buch ausleihen“, der Controller ruft die DAO-Methode im Model auf und aktualisiert die View

Beispiel

1. Der Benutzer klickt auf den „Suche“-Button, nachdem er ein Titel eines Buches eingegeben hat
2. Der Controller empfängt die Eingabe, ruft das Model auf und holt sich die benötigten Daten. In größeren Anwendungen würde er die Service-Schicht befragen, die wiederum mit DAO- und Modellschicht kommuniziert
3. Das Model verarbeitet die Daten und gibt sie zurück
4. Der Controller leitet die Daten an die View weiter
5. Die View zeigt die gewünschten Daten

Einleitung



https://medium.com/@sadikarahmantanisha/the-mvc-architecture-97d47e071eb2

SMART INDUSTRY CAMPUS

DEMO MVC-Pattern

SMART INDUSTRY CAMPUS

GUI-Programmierung

GUI

- steht für Graphic User Interface (Graphische Benutzeroberfläche)
- gibt verschiedene Möglichkeiten eine Benutzeroberfläche zu erzeugen
- Desktopanwendungen werden immer seltener, gesetzt wird auf Webtechnologien
  - Desktop: Swing und JavaFX
  - Web : Angular, React

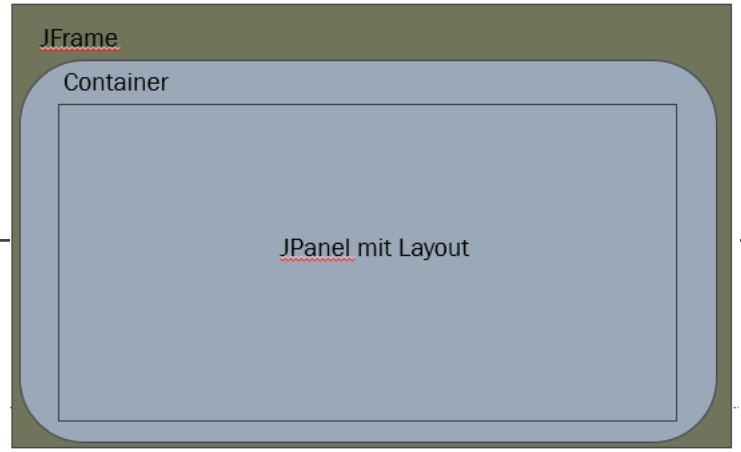
Swing

- ist eine Bibliothek in Java zur Erstellung grafischer Benutzeroberflächen
- bietet vordefinierte Komponenten:
  - Hauptfenster
  - Layouts
  - Buttons
  - Labels
  - Textfelder
  - ...

SMART INDUSTRY CAMPUS

Aufbau einer Oberfläche

Swing



- JFrame:
  - Hauptfenster der Swing-Anwendung
  - In ihm werden weiter Komponenten platziert
- Container:
  - Ist eine Komponente, die andere Komponenten enthalten kann
  - Ein JFrame selbst ist ein Container
- JPanel:
  - Wird innerhalb eines JFrames oder Containers verwendet
- Layouts:
  - Definieren die Anordnung der Komponenten innerhalb eines Containers

JFrame

- ist ein Hauptfenster
- Hauptfenster können einen Titel, eine Größe, eine Position, … haben
- dazu gibt es entsprechende Methoden
  - setTitle(String s)
  - setSize(int width, int height)
  - setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE) – setzt, wie das Hauptfenster geschlossen werden soll
  - setLocationRelativeTo(null); - setzt die Position, wo das Hauptfenster geöffnet werden soll
  - setVisible(boolean b);
  - getContentPane(); - holt sich den Container des Hauptfensters

SMART INDUSTRY CAMPUS

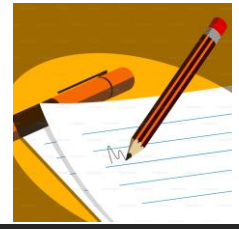
DEMO JFrame

Aufgaben

```
setTitle(String s)
setSize(int width, int height)
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
setLocationRelativeTo(null);
setVisible(boolean b);
getContentPane();
```

SMART INDUSTRY CAMPUS

- Pulle auf Git
- Öffne den Ordner unter Unterricht _20250327GUIProgrammierung/view
- Öffne die Klasse MyFirstJFrame
- Gebe dem Frame einen Namen, eine Größe, eine Position und setzte es auf sichtbar
- Erzeuge eine main-Methode und erzeuge ein Objekt der Klasse MyFirstJFrame



JPanel

- kann man sich wie eine Leinwand vorstellen, die im Hauptfenster immer wieder gewechselt wird
- hat einen Container, in den weitere Komponenten hinzugefügt werden können:
  - Weitere Panels
  - Buttons
  - Labels
  - ...
- Syntax:

```
JPanel p = new JPanel();
```

- die Komponenten werden üblicherweise nach einem Layout ausgerichtet

```
JPanel p = new JPanel(LayoutManager lM);
```

SMART INDUSTRY CAMPUS

JButton, JLabel, JTextField

- ein Button in Swing kann mit Hilfe der Klasse JButton erzeugt werden
- Syntax:

```
JButton b = new JButton("Hallo");
```

JButton, JLabel, JTextField

- ein Label in Swing kann mit Hilfe der Klasse JLabel erzeugt werden
- Syntax

```
JLabel l = new JLabel("Ich bin ein Label");
```

JButton, JLabel, JTextField

- ein Textfeld in Swing kann mit Hilfe der Klasse JTextField erzeugt werden
- Syntax:

```
JTextField tf = new JTextField();
```

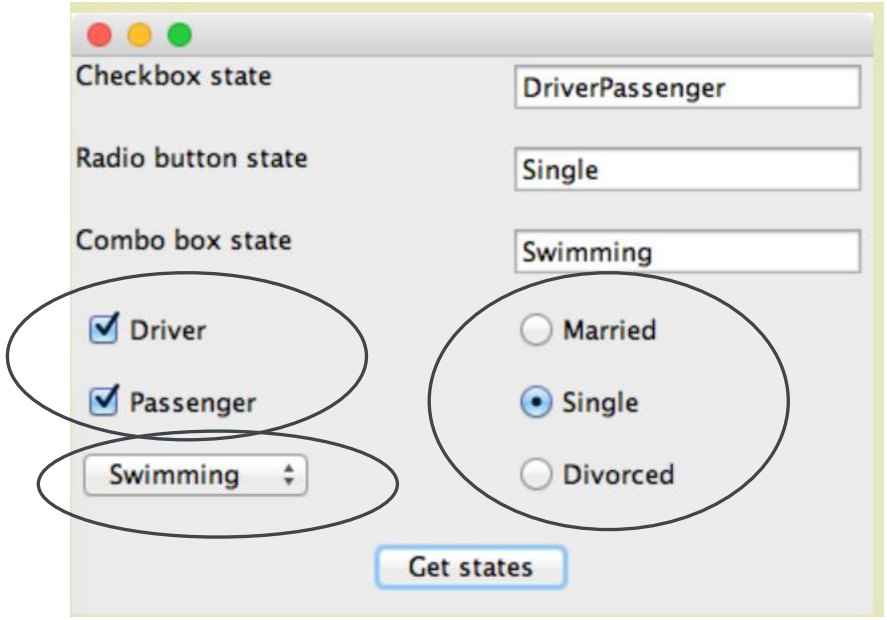
- hat eine Methode .getText() , die den Inhalt des Feldes zurückgibt
- hat eine Methode .setText(), die den Inhalt des Feldes setzt

JComboBox, JRadioButton, JCheckBox

SMART INDUSTRY CAMPUS

https://docs.oracle.com/javase/tutorial/uiswing/examples/components/index.html

- Swing hat noch viele weiter definierte Komponenten



https://kennethalambert.com/BreezySwing/radiobuttons.html

SMART INDUSTRY CAMPUS

DEMO JPanel und JButton/JLabel/JTextField

SMART INDUSTRY CAMPUS

Aufgaben

- erstelle in dem Konstruktor der MyFirstJFrame-Klasse einen JPanel
- füge dem JPanel einen Button hinzu
- füge das JPanel dem Container hinzu
- setze nach der Operation den Frame auf sichtbar

Layouts

- um Komponenten innerhalb eines Containers zu strukturieren
- gibt viele verschiedene vordefinierte Layouts:
  - GridLayout
  - BorderLayout
  - GridBagLayout
  - FlowLayout
  - ...

GridLayout

- wie der Name schon impliziert, wird nach Rastern sortiert

Spalten

Zeilen	S: 0, Z: 0	S: 0, Z: 1
	1,0	1,1
	2,0	2,1

- Syntax:

```
JPanel p = new JPanel(new GridLayout(int rows, int column));
```

- beim Hinzufügen der Komponenten in das Panel, werden diese automatisch sortiert

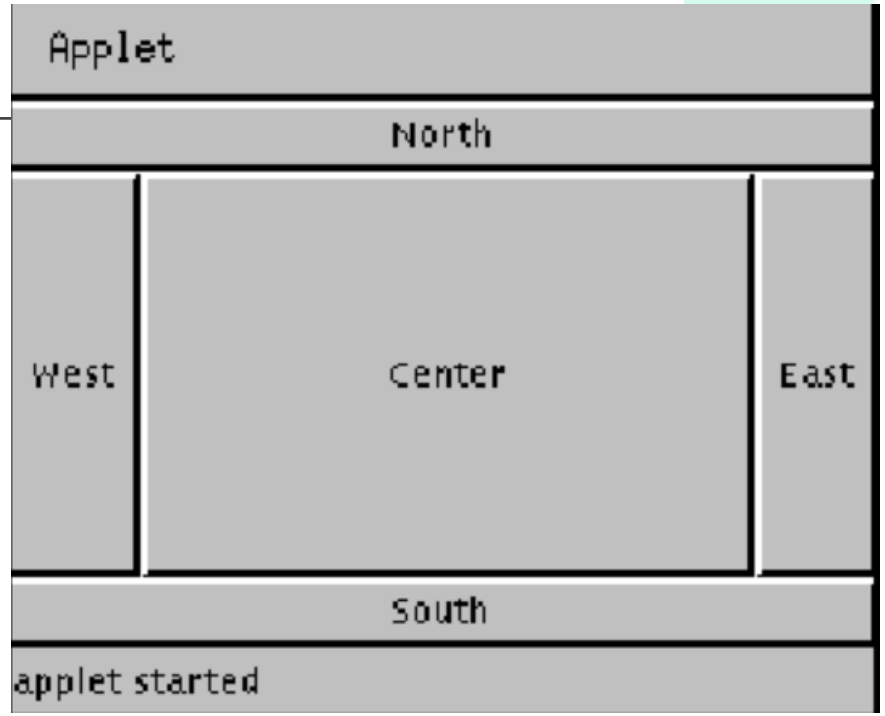
BorderLayout

- wie der Name schon impliziert, wird nach Grenzen sortiert
- Syntax:

```
JPanel p = new JPanel(new BorderLayout());
```

- Beim Hinzufügen einer Komponente muss man angeben, an welche Position diese gesetzt werden soll

```
JPanel p = new JPanel(new BorderLayout());
JButton b = new JButton("Hallo");
p.add(b, BorderLayout.EAST);
```



https://docs.oracle.com/javase/7/docs/api/java/awt/BorderLayout.html

SMART INDUSTRY CAMPUS

Aufgaben

- füge deinem zuvor erzeugten Panel ein Layout hinzu
- füge dem Panel nun Komponenten hinzu

SMART INDUSTRY CAMPUS

Controller-Seite

SMART INDUSTRY CAMPUS

ActionListener

- ist ein funktionales Interface -> Lambdas können angewandt werden!
- um die Benutzereingaben und Klicks zu verarbeiten, braucht man einen sogenannten ActionListener, der auf eine Aktion (=Buttonklick,...) wartet
- dieser wird je Komponente hinzugefügt
- Syntax:

```
public void addActionListenerToButton(ActionListener aL) {
   button.addActionListener(aL);
}
```

Aufgaben

- Schreibe deinen Code so um, sodass du eine public Methode „addActionListenerToButton(ActionListener l)“ schreiben kannst, um dem Button einen ActionListener hinzuzufügen

```
public void addActionListenerToButton(ActionListener aL) {
    button.addActionListener(aL);
}
```

Controller

- beim Drücken des Buttons, soll nun etwas passieren
- für die Verarbeitung von Benutzereingaben ist der Controller zuständig
- die Methode, die einer Komponente hinzugefügt wurde, wird im Controller aufgerufen
  - Im Konstruktor notwendig, da die Listener einmal am Anfang hinzugefügt werden müssen
- mittels Lambda wird eine Antwort auf die Aktion zugewiesen:

```
this.myFirstJFrame.addActionListenerToButton(e-> System.out.println("Ich wurde gedrückt"));
```

Aufgaben

- Öffne den Ordner unter Unterricht _20250327GUIProgrammierung/controller
- Instanziiere dein erzeugtes Jframe
- rufe im Konstruktor die addActionListenerToButton- Methode des JFrames auf
- nutze Lambdas, um eine Reaktion auf das Klicken des Buttons zu erhalten
  - Z.B. System.out.println(...);
  - Oder: eine eigene Methode, die im Controller geschrieben wurde (z.b: eine Berechnung) und diese dann aufrufen.
- die Main-Methode aus der View entfernen
- eine Main-Methode im Controller hinzufügen, die den Controller instanziiert

SMART INDUSTRY CAMPUS

Best Practices

- es existiert ein Hauptframe (JFrame)
- der Container des Hauptframes wird immer wieder mit JPanels gefüllt
  - Das heißt wir haben bei einer großen Anwendung mehrere Panels (WelcomePage, LoginPage, …)
  - Diese Panels werden im JFrame immer wieder ausgewechselt (je nachdem, was der Nutzer drückt)
- Controller werden normalerweise in Themen gruppiert:
  - Es gibt z.B. einen UserController, der Dinge wie, „User löschen“, „User anlegen“, „User suchen“, … übernimmt
- wir haben Model-Klassen, mit denen der Controller kommuniziert
  - Z.B. User

Nachmittagsaufgabe

- erweitere dein DB-Projekt um das Package View und Controller
- erstelle eine View zum Holen eines Buches anhand seiner ID:
  - Ein JFrame, mit einem Panel und einem Label, einem Textfeld und einem Button
  - Erstelle eine Methode, die einen ActionListener dem Button zufügt
- erstelle im Controller eine Methode, welche die View instanziiert
  - Die ActionListener-Methode aus der View soll aufgerufen werden
  - Welche Methode existiert bereits, die ein Buch anhand der ID holt? Nutze die Service-Klasse.
  - Gebe das Buch in der Konsole aus