

- Registrierung
- New feature 1
  - New feature 2
  - New feature 3
  - Kleinere Verbesserungen
  - Fehlerbehebungen

Aktualisieren



20.0



INFO

8:47

## 09-GUI-3-MVC

Objektorientierte Programmierung | Matthias Tichy



Software Engineering  
Programming Languages

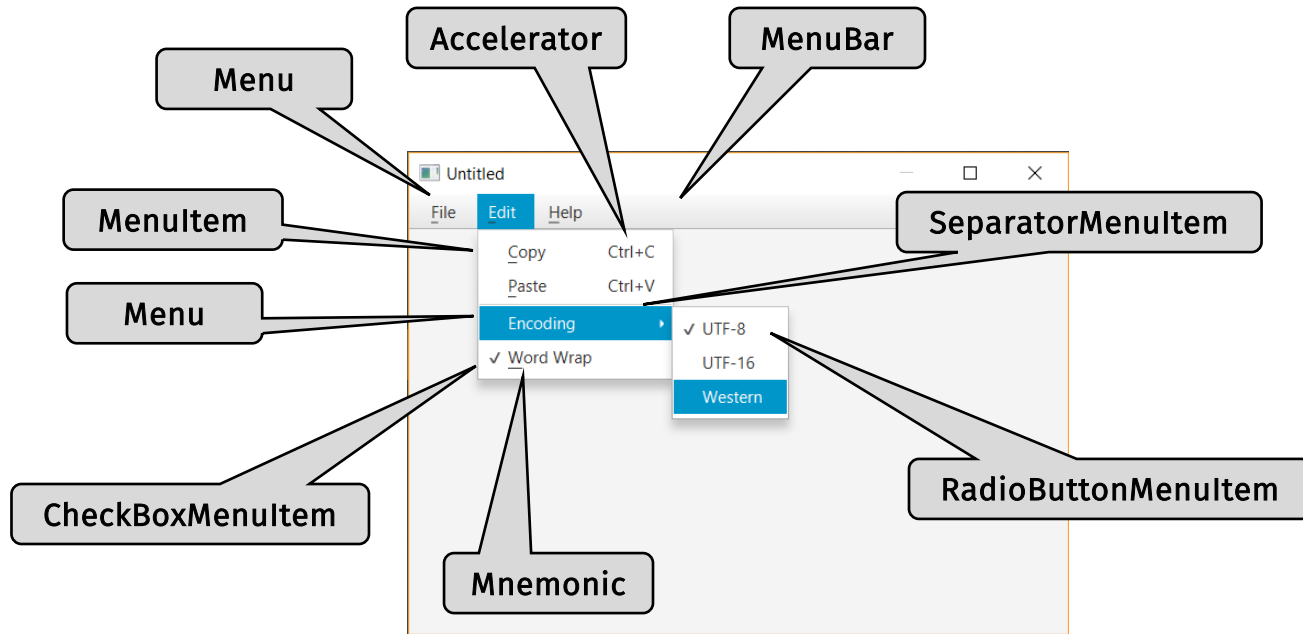


universität  
uulm

# Lernziele

- Weitere GUI-Elemente
- Zusammenhänge zwischen UI, Daten und Interaktionen

# Menü



# ToolBar

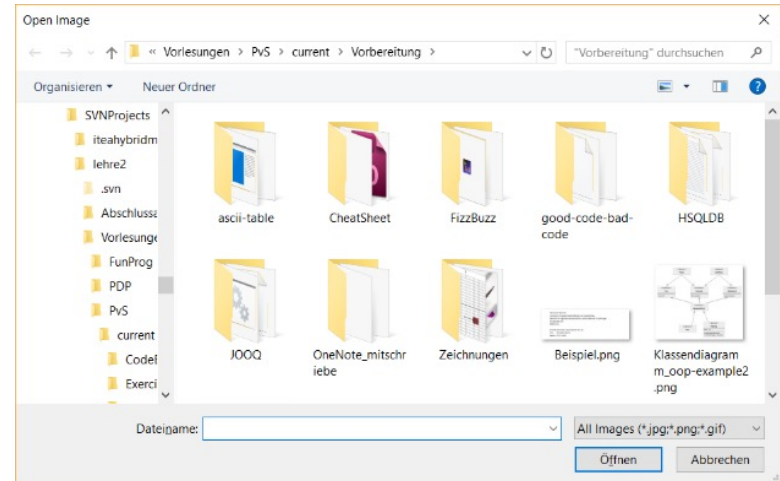
- kann beliebige Komponenten aufnehmen
  - Typischerweise mit Icons versehen
  - mehrere möglich
- 
- Oft nur Abkürzung für andere Aktionen → EventHandler wieder verwenden

# Dialog

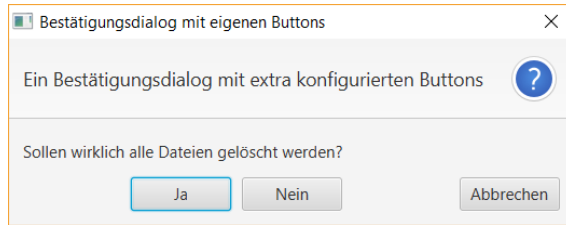
- Fenster, um einen bestimmten Bereich der Eingabe zu gruppieren
- Modal vs. Nicht-Modal
- einige vordefinierte Dialoge
  - FileChooser
  - ColorPicker
  - Alert, TextInputDialog, ChoiceDialog

# FileChooser

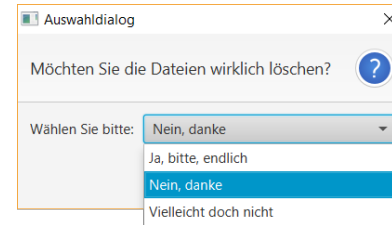
- nativer Dialog wird aufgerufen
- typische Einstellmöglichkeiten:
  - Startverzeichnis
  - Datei-Filter
  - Multiple-Selection erlaubt?
  - nur Ordnerauswahl



# Alert, Eingabedialoge

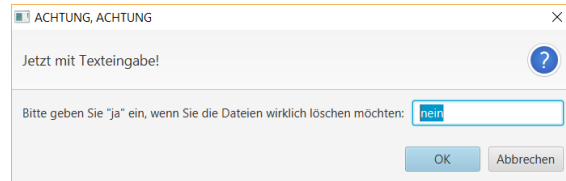


Alert

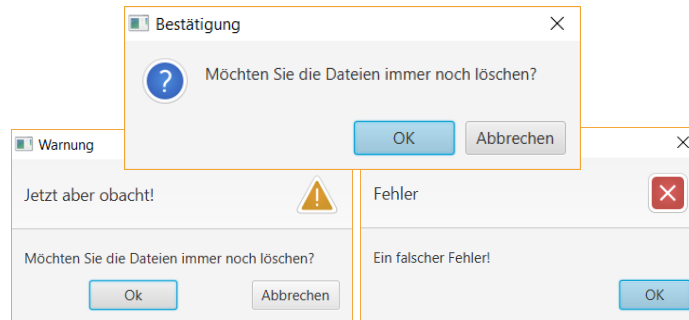


ChoiceDialog

TextInputDialog

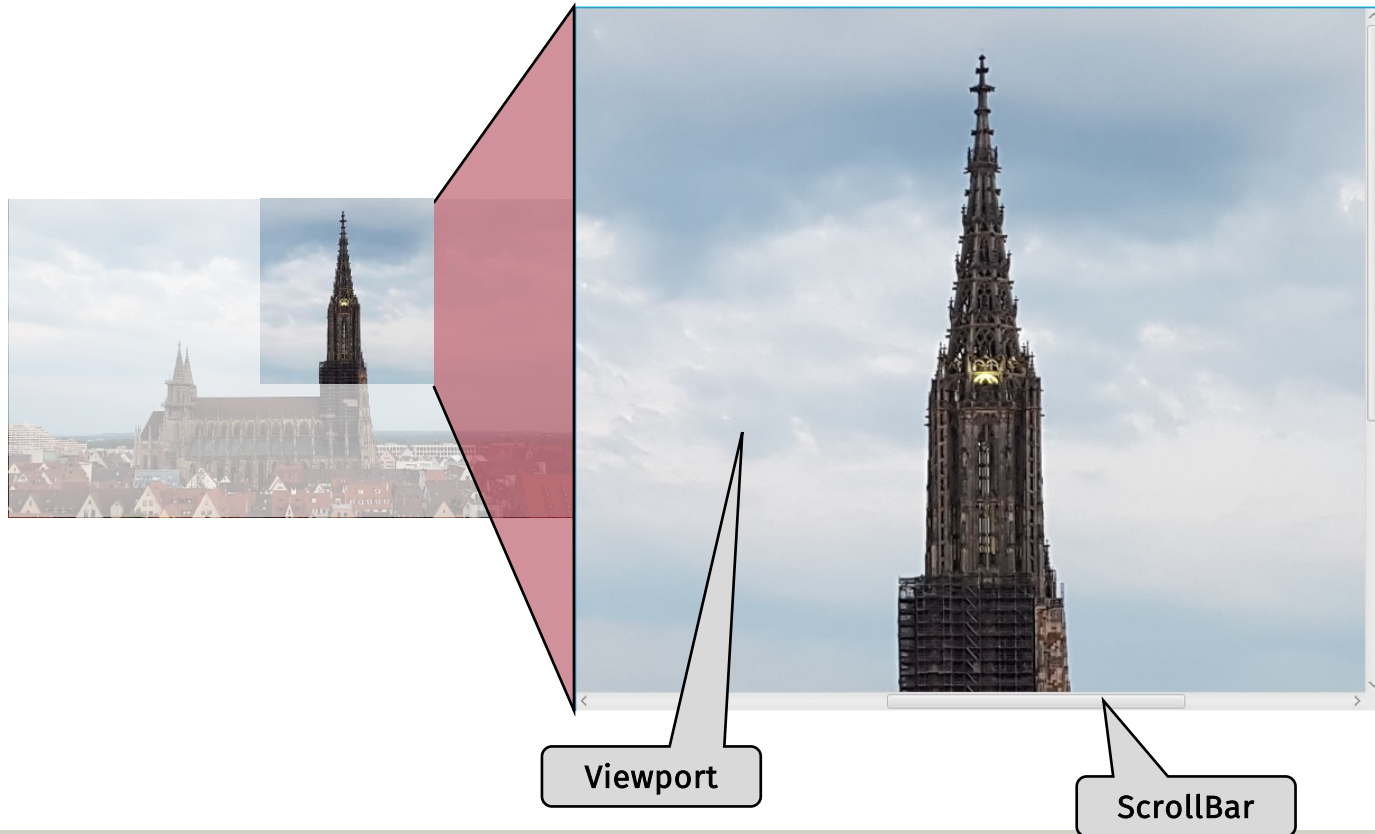


Alert



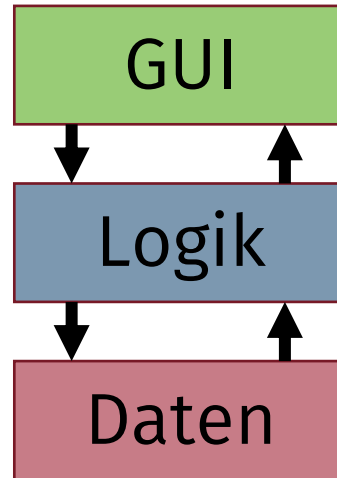
- Optionen:
  - Text, Titel, Typ, Icon
  - speziellere Optionen (Auswahl, Buttonbeschriftung) je nach Art

# ScrollPane





# UI vs. Daten vs. Interaktionen



# Problem

[illegible]

```

    @RequestMapping(method = RequestMethod.GET, value = "/api/v1/locations/{id}")
    public Location findById(@PathVariable("id") Long id) {
        Location location = locationRepository.findById(id)
            .orElseThrow(() -> new NotFoundException("Location not found"));
        return location;
    }

    @RequestMapping(method = RequestMethod.GET, value = "/api/v1/locations")
    public Page<Location> findAll(@RequestParam(value = "page", defaultValue = "1") int page,
                                @RequestParam(value = "size", defaultValue = "10") int size) {
        Page<Location> locations = locationRepository.findAll(PageRequest.of(page - 1, size));
        return locations;
    }

    @RequestMapping(method = RequestMethod.POST, value = "/api/v1/locations")
    public Location create(@RequestBody Location location) {
        Location savedLocation = locationRepository.save(location);
        return savedLocation;
    }

    @RequestMapping(method = RequestMethod.PUT, value = "/api/v1/locations/{id}")
    public Location update(@PathVariable("id") Long id, @RequestBody Location location) {
        Location existingLocation = locationRepository.findById(id)
            .orElseThrow(() -> new NotFoundException("Location not found"));
        existingLocation.setName(location.getName());
        existingLocation.setDescription(location.getDescription());
        existingLocation.setAddress(location.getAddress());
        existingLocation.setCoordinates(location.getCoordinates());
        return locationRepository.save(existingLocation);
    }

    @RequestMapping(method = RequestMethod.DELETE, value = "/api/v1/locations/{id}")
    public void delete(@PathVariable("id") Long id) {
        Location location = locationRepository.findById(id)
            .orElseThrow(() -> new NotFoundException("Location not found"));
        locationRepository.delete(location);
    }
}

```

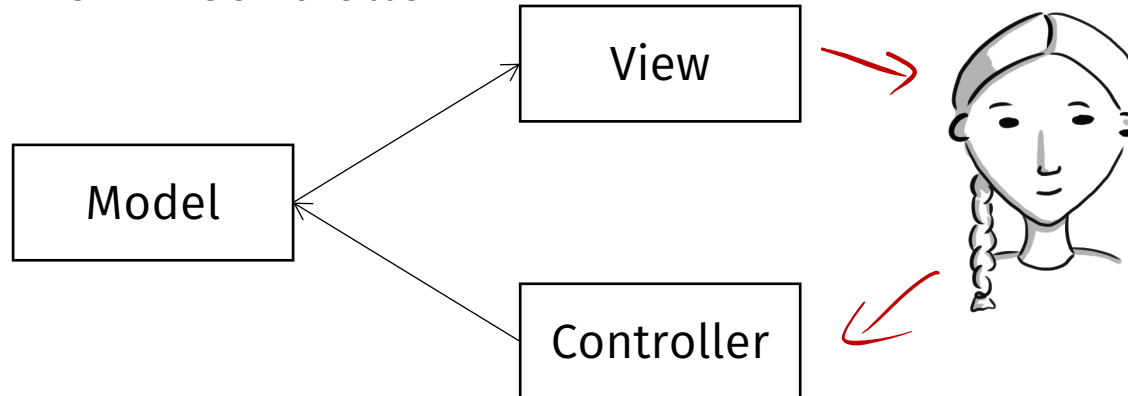
[illegible]

# Fragen

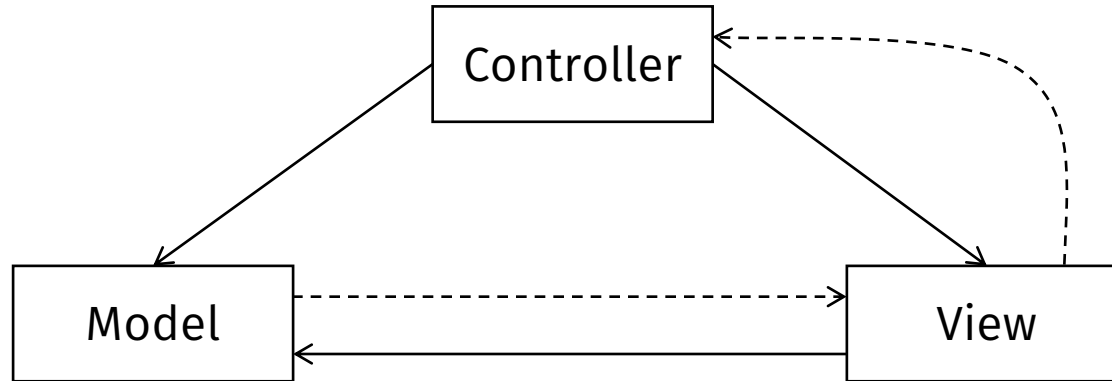
- Wie können wir den Code sauber separieren?
- Welche Teile gehören zusammen?
- Wie läuft die Kommunikation zwischen den einzelnen Teilen?

# Lösung

- Probleme schon sehr alt:  
1979 mit der ersten grafischen Oberfläche
- Vorschlag von damals:  
Model – View – Controller



# Model-View-Controller heute



- ← direkte Verbindung (Referenz)
- ←---- indirekte Verbindung (z.B. Observer)

# Rollen

## ■ Model

- darzustellende Daten (auch mehrere Klassen)
- Teil der "Logik" (Anwendungsschicht)
- "dümmer" Teil (kennt weder View noch Controller)

## ■ View

- verantwortlich für Darstellung der Daten
- ermöglicht Interaktion über Ereignisse
- Dialogablauf unbekannt

# Rollen

## ■ Controller

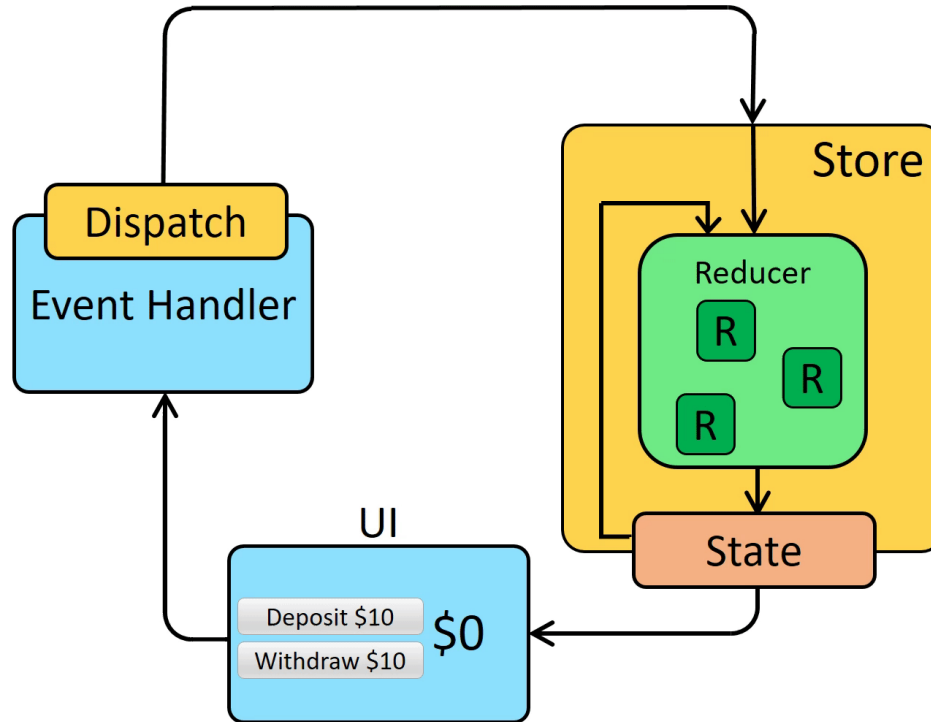
- Verbindung zwischen Model und View
- oft Teil der Präsentationsschicht
- empfängt Ereignisse der View
- Umformung zu Daten für Model
- leitet Ergebnisse zurück an View (evtl. auch direkt vom Model über Observer-Pattern)
- typischerweise ein Controller pro View

# offene Aufgaben

- Geschäftslogik
  - meist extra Klassen
- Validierung von Benutzereingaben
  - oft in der View
- Datenformatierung
  - oft auch in der View
  - extra ViewModel (→ Model - View- ViewModel)



# Web: Redux (State / Events / Actions)



<https://redux.js.org/tutorials/essentials/part-1-overview-concepts>

# Lernziele

- Weitere GUI-Elemente
- Zusammenhänge zwischen UI, Daten und Interaktionen