

Bachelor of Science (BSc) in Informatik
Modul Software-Entwicklung 1 (SWEN1)

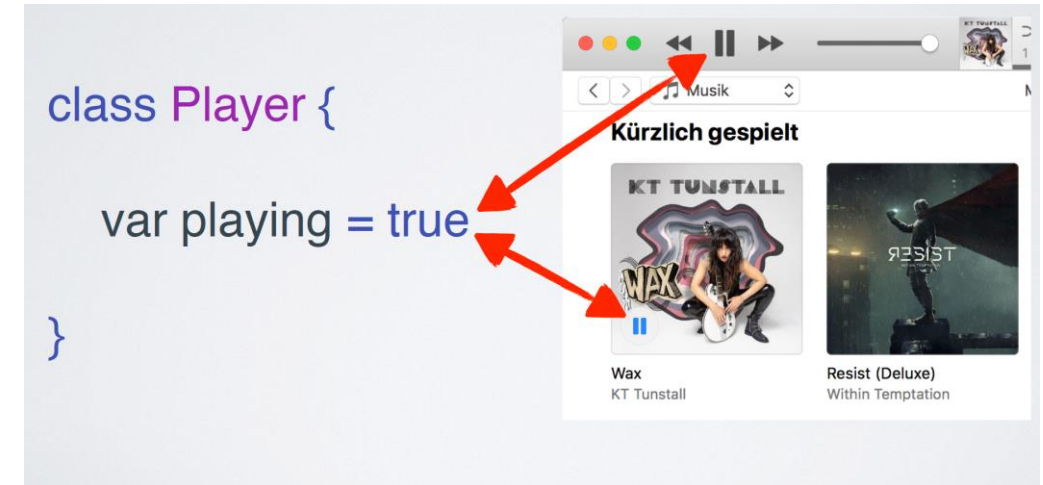
V2 – GUI-Architekturen

SWEN1/PM3 Team:
R. Ferri (feit), G. Wyder (wydg),
D. Lienhart (lieh), K. Bleisch (bles)

Ausgabe: HS23

Um was geht es?

- Was sind GUI-Architekturen?
- Was ist ereignisorientierte Programmierung?
- Wie wird das Observer Design Pattern in GUI-Applikationen verwendet?
- Wie werden gängige Architektur-Patterns für GUI-Applikationen wie MVC, MVP und MVVM verwendet?



Lernziele LE 11 – GUI-Architekturen

Sie sind in der Lage,

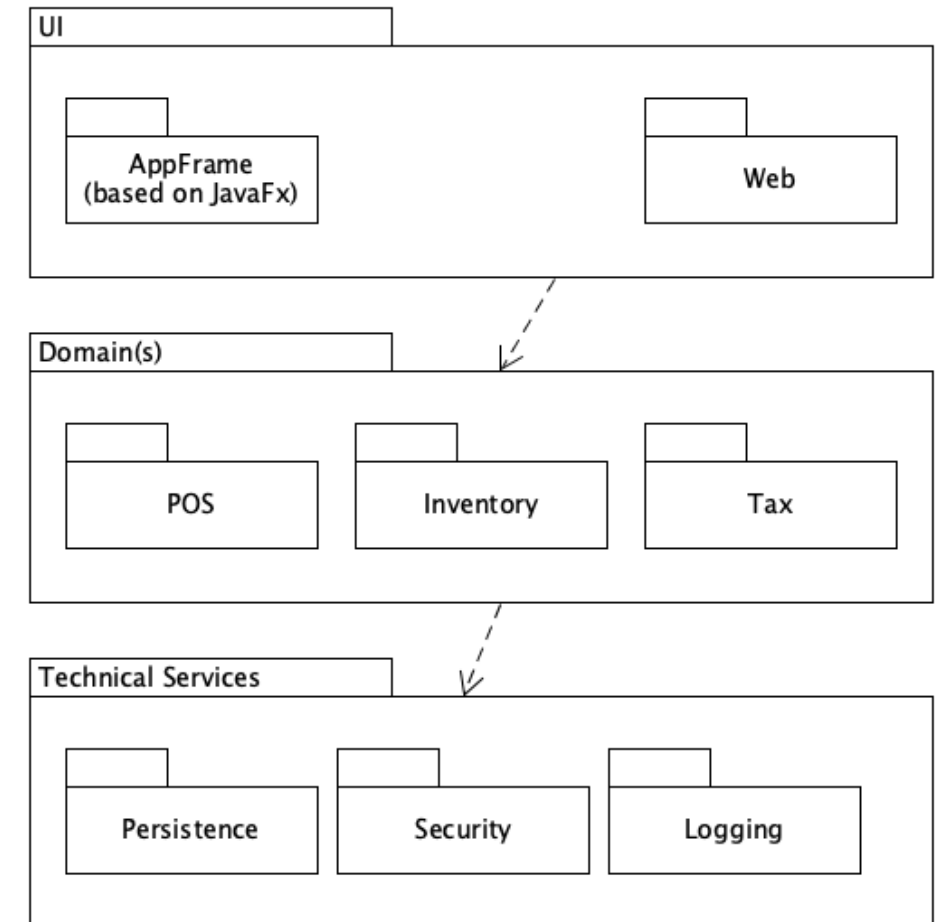
- zu erläutern, wie **GUI-Architekturen entworfen und umgesetzt** werden können,
- das **Observer-Pattern** in GUI-Applikationen einzusetzen,
- Beispiele für die Anwendung der **Architektur-Patterns MVC, MVP und MVVM** zu verstehen und zu erweitern,
- die Varianten für die **ereignisorientierte Programmierung** für JavaFX praktisch anzuwenden.

Agenda

1. **Einführung in GUI-Architekturen**
2. Patterns in GUI-Architekturen
3. Architektur-Patterns MVC, MVP und MVVM
4. Kurzer Recap JavaFX
5. Wrap-up und Ausblick

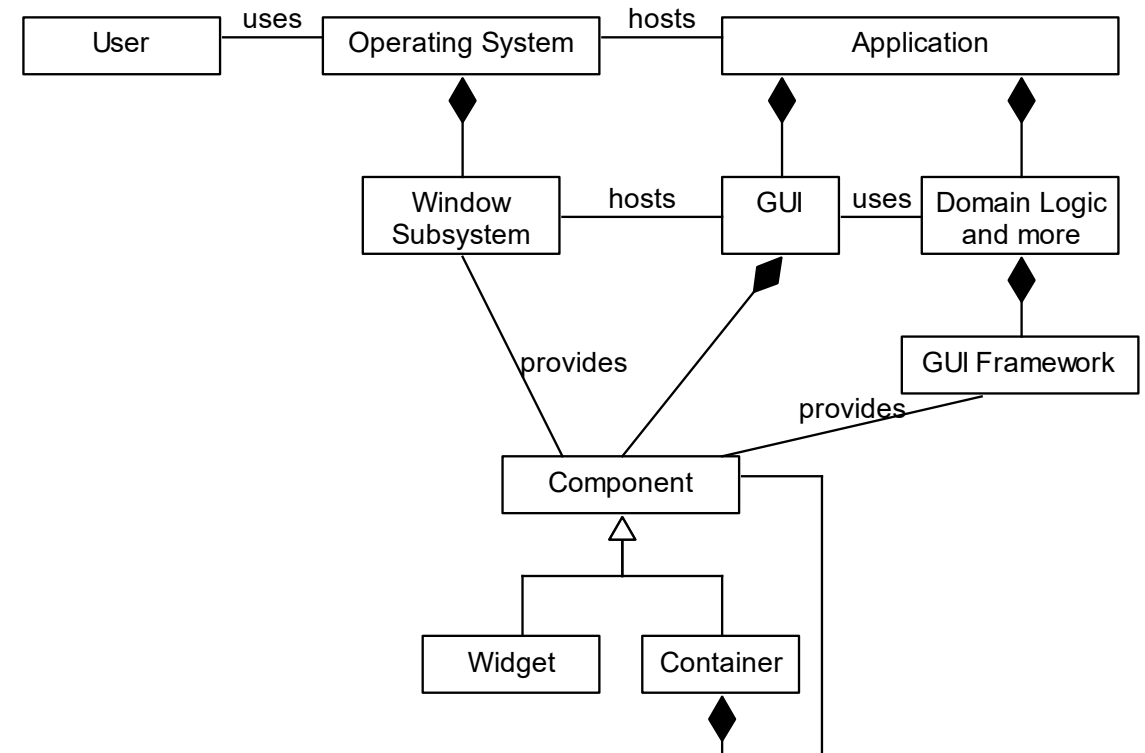
Recap - Schichtenarchitektur

- **UI (Presentation)**: Enthält die Präsentations-Logik
 - Technologie-gebunden (rascher Wechsel)
 - Desktop-App (Java: Swing oder JavaFX)
 - Mobile-App (iOS, Android, Hybrid-Frameworks)
 - Web-App (HTML, CSS, JS, Frameworks)
- **Domain**: Enthält den fachlichen Kern der Anwendungslogik.
- **Technical Services**: Enthält in Abhängigkeit der technischen Infrastruktur wichtige technologieabhängige Komponenten.

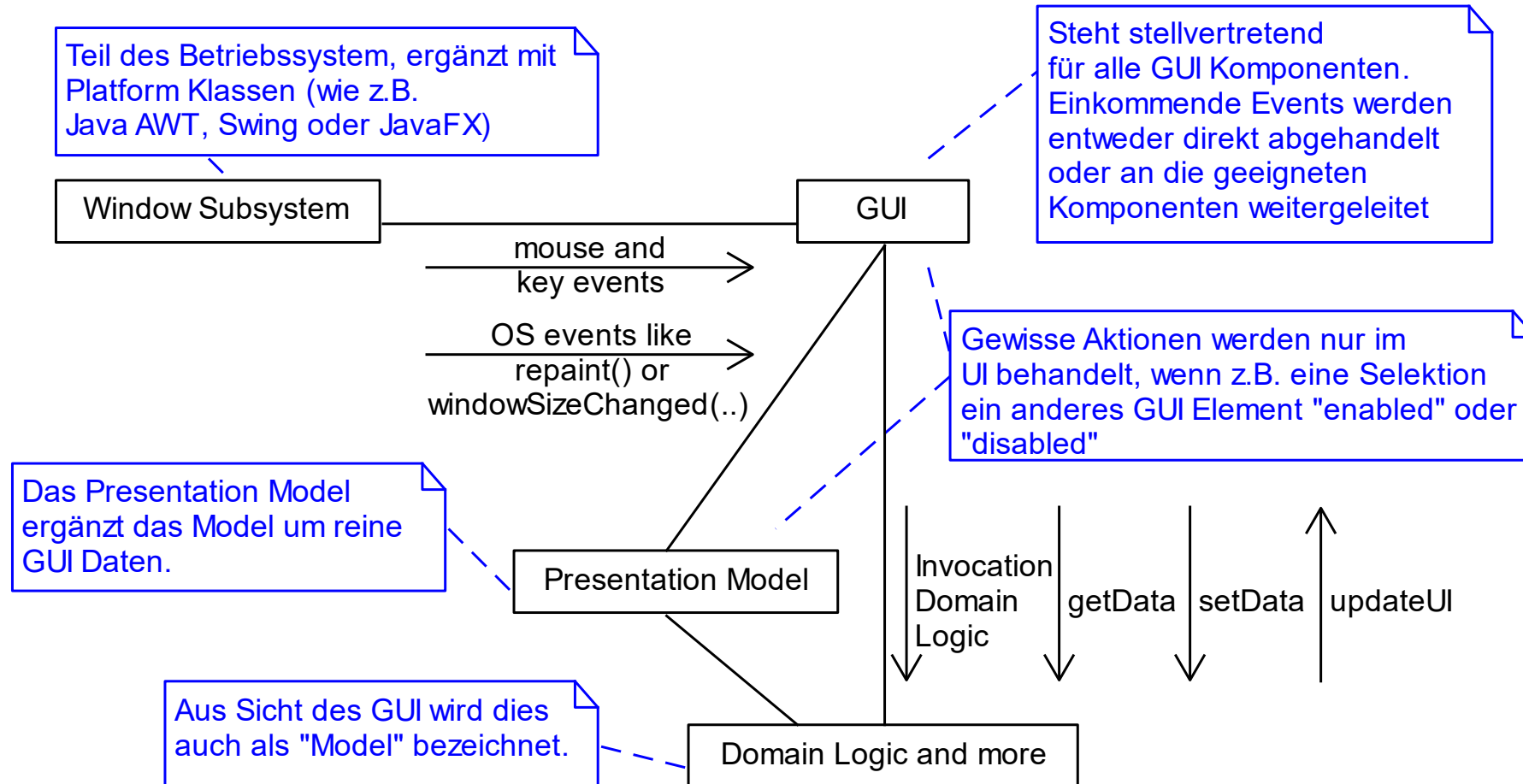


Domänen Modell GUI

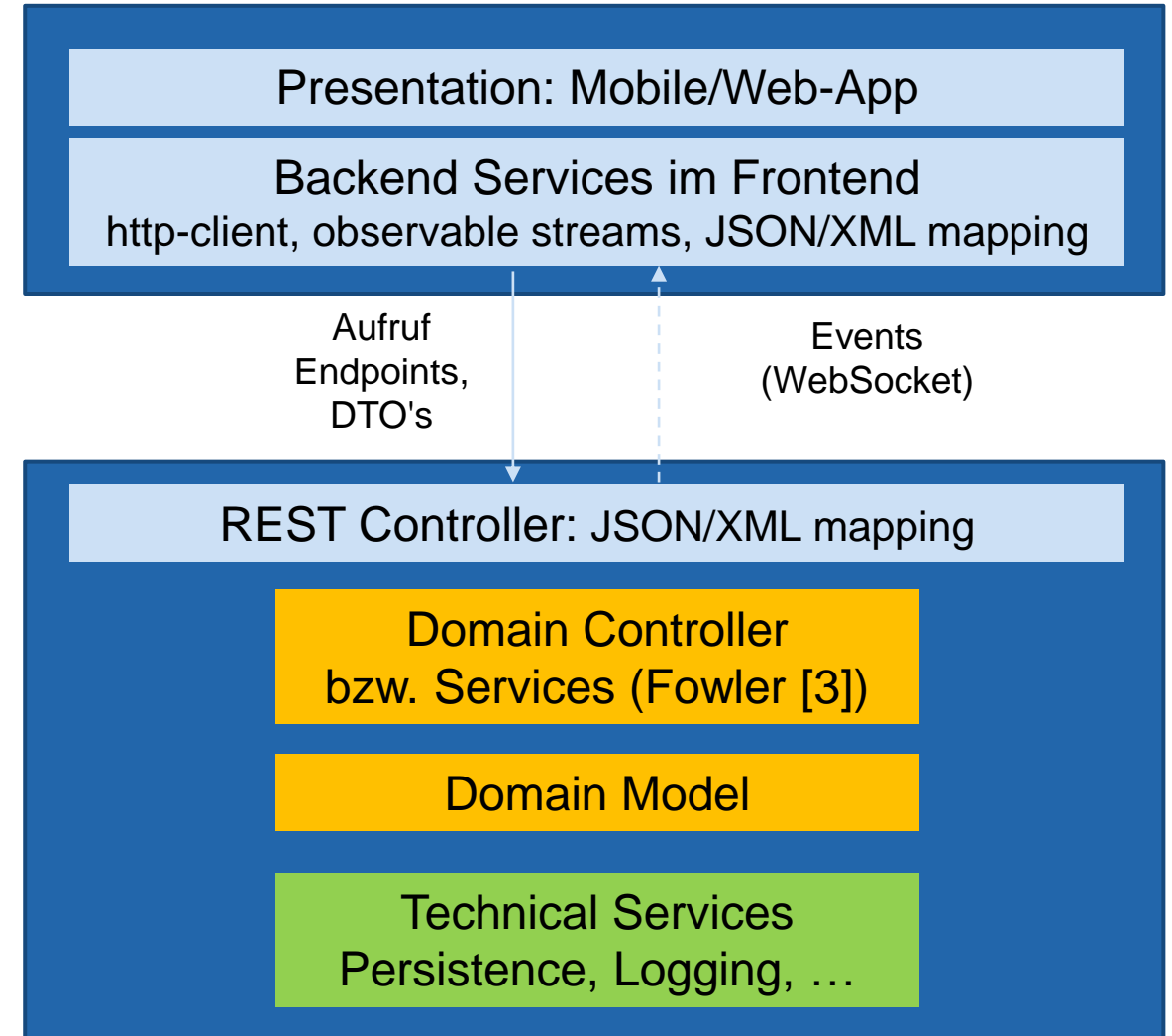
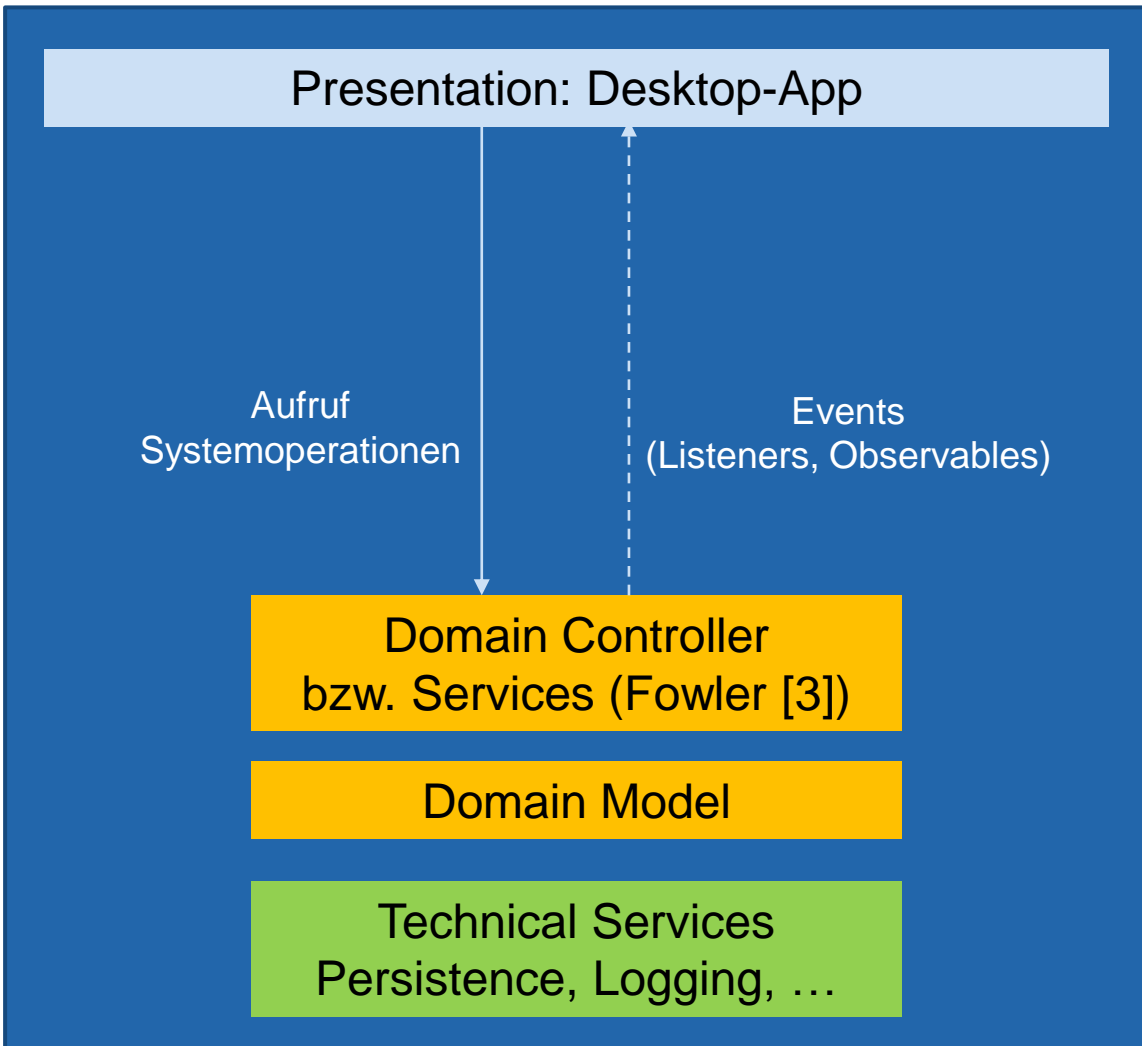
- Das **Domänenmodell** einer GUI Anwendung enthält alle relevanten Konzepte
- Das **GUI** ist typischerweise aus **Komponenten** aufgebaut. Diese werden vom Operating System und von GUI Frameworks zur Verfügung gestellt, können aber auch selber entwickelt werden.



Zusammenarbeit GUI mit Betriebssystem und Domänenlogik



Schichtenarchitektur Vergleich



Agenda

1. Einführung in GUI-Architekturen
- 2. Patterns in GUI-Architekturen**
3. Architektur-Patterns MVC, MVP und MVVM
4. Kurzer Recap JavaFX
5. Wrap-up und Ausblick

Patterns für GUIs (1/2)


- Was und wie können Patterns verwendet werden, um GUIs zu strukturieren?
- Folgende Fragen sollen beantwortet werden:
 - Wer besitzt welche Teile der Daten?
 - Wer hat Zugriff auf die Daten? Wie?
 - Wer verarbeitet die Daten?
 - Wer präsentiert welche Teile der Daten?
 - Wer kann Daten verändern?
 - Wer wird informiert, nachdem Daten verändert wurden?

Patterns für GUIs (2/2)

- Observer
 - Wird über Änderungen an Objekten informiert ohne observiertes Objekt im GUI zu instanziiieren.
- Model View Controller (MVC) & Friends
 - Separierung von Zustand, Präsentation und Logik.
- Composite
 - Die gesamte View-Hierarchie wird in einer Composite-Struktur gerendert. Die View-Fläche muss unter den vorhandenen Elementen aufgeteilt werden (z.B. über *Responsive Layout Manager*).

Observer Pattern in Java : Property Change Listener

```
public class StudentModel {  
    private String name;  
  
    private final PropertyChangeSupport changes = new PropertyChangeSupport( this );  
  
    public void setName(String name) {  
        String oldName = this.name;  
        this.name = name;  
        changes.firePropertyChange( "name", oldName, name );  
    }  
  
    public void addPropertyChangeListener( PropertyChangeListener listener ) {  
        changes.addPropertyChangeListener( listener );  
    }  
}  
  
studentModel.addPropertyChangeListener( e -> {  
    studentViewModel.onStudentNameChanged(e.getNewValue().toString());  
});
```



Weitere Anwendungen von Observer Pattern

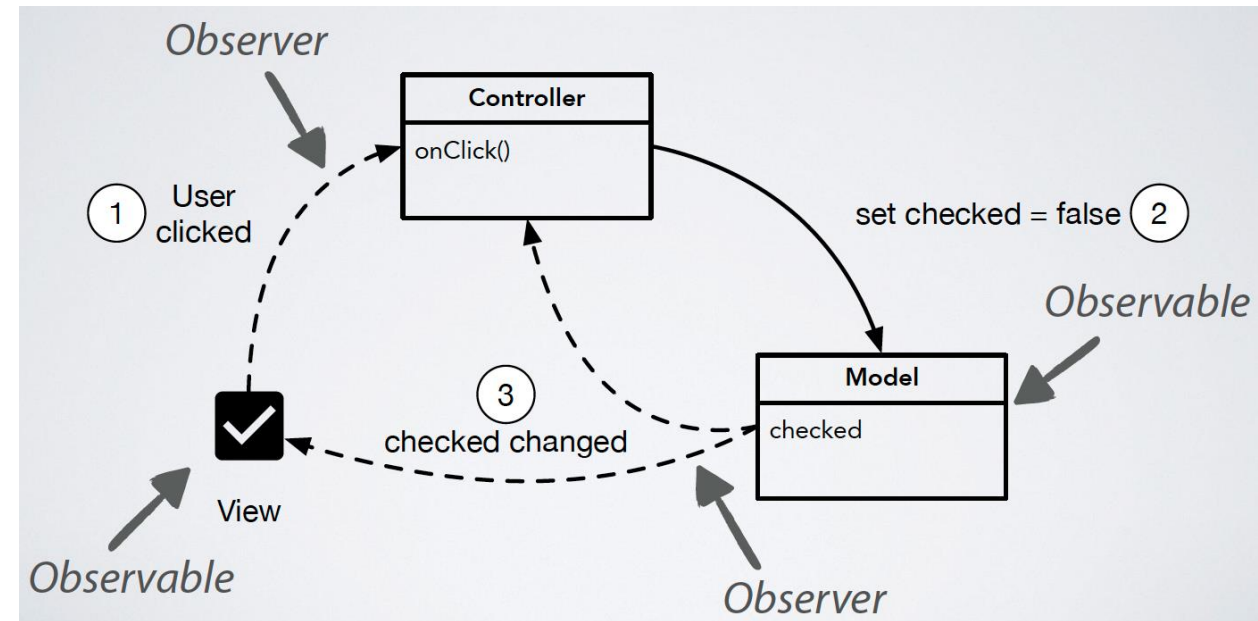
- Event Handling
 - Jeder `onClickListener()` etc. ist eine Anwendung des Observer Patterns.
- RxJava
 - Die Datenquellen werden aus diesem Grund Observables benannt.
- Event Bus
 - Mehrere Observables publizieren Events auf den gleichen Kanal.
 - Mehrere Observer werden mit dem Kanal verbunden und reagieren auf die für sie relevanten Events.
- Data Binding
 - JavaFX Properties, Java Beans Property Change Listener.

Agenda

1. Einführung in GUI-Architekturen
2. Observer Pattern in GUI-Architekturen
3. **Architektur-Patterns MVC, MVP und MVVM**
4. Kurzer Recap JavaFX
5. Wrap-up und Ausblick

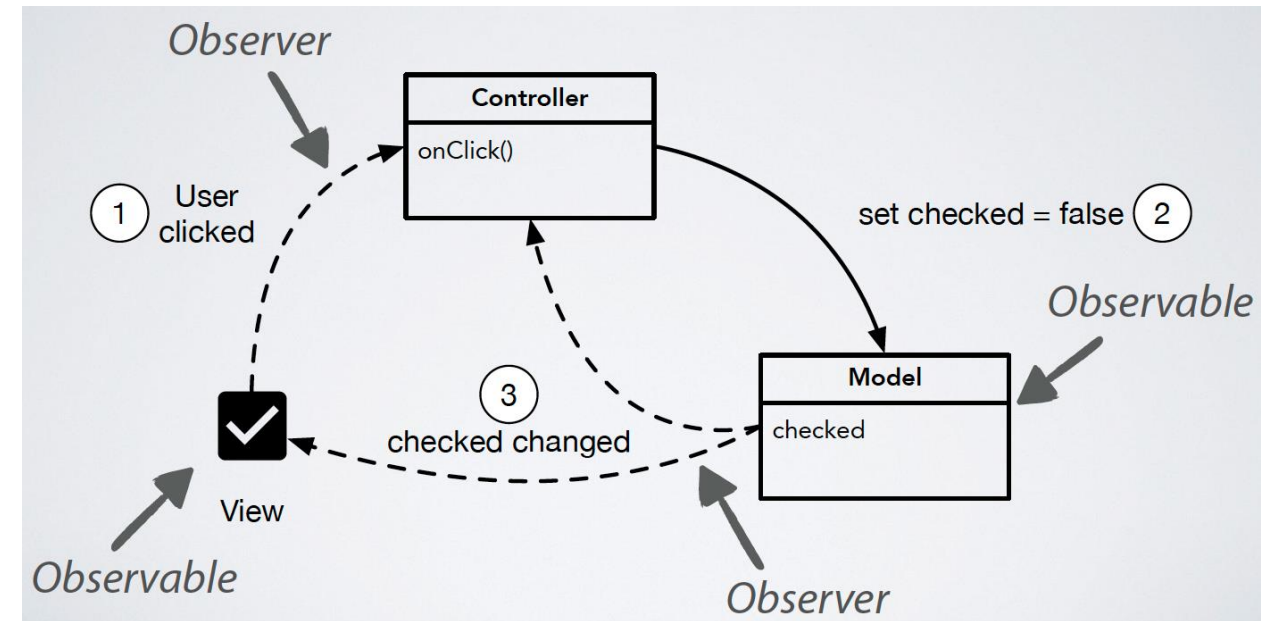
Model View Controller (MVC) (1/2)

- Trennung von Zustand, Präsentation und Logik
 - Bessere Testbarkeit
 - Viele unterschiedliche Interpretationen
- View: Was der Benutzer sieht
 - z.B. Checkbox
- Model: Anzuzeigende Daten
 - checked = true/false



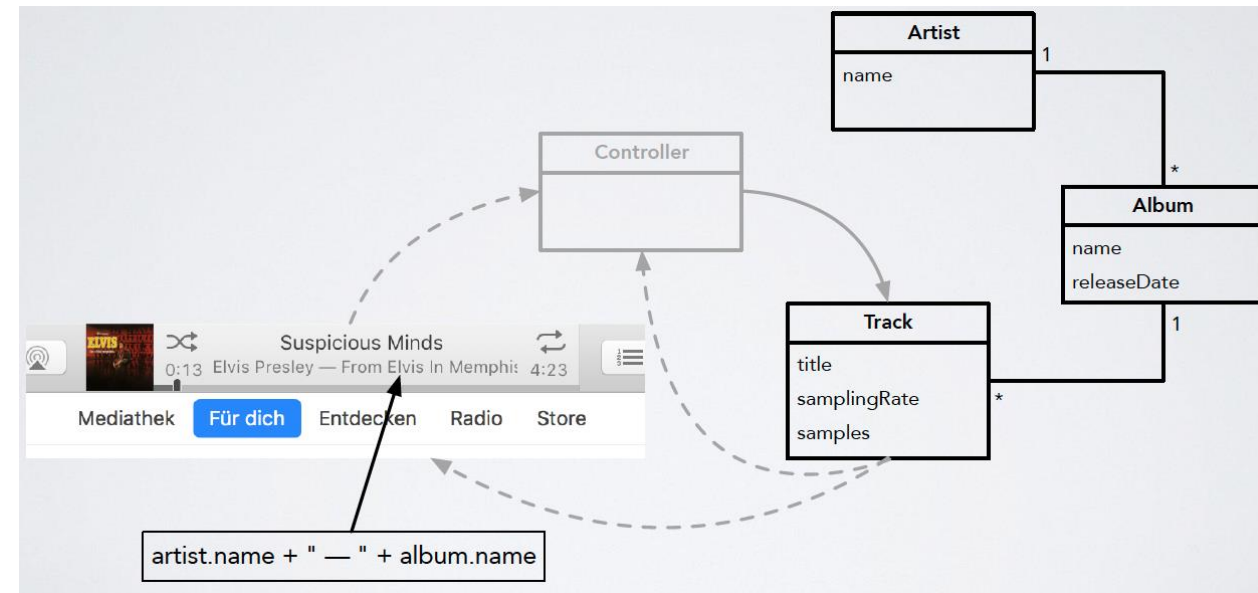
Model View Controller (MVC) (2/2)

- Controller
 - Behandelt Benutzer-Ereignisse
 - Koordiniert Aktionen mit anderen Controllern in der Anwendung
 - Enthält die Logik für die Veränderung des Models (über Domain Controller)
- Observer-Pattern ist integraler Teil von MVC
 - Das Model (Observable) informiert die View (Observer) über Änderungen
 - Der Controller sollte nicht direkt mit der View kommunizieren, sondern indirekt über die Aktualisierung des Model.



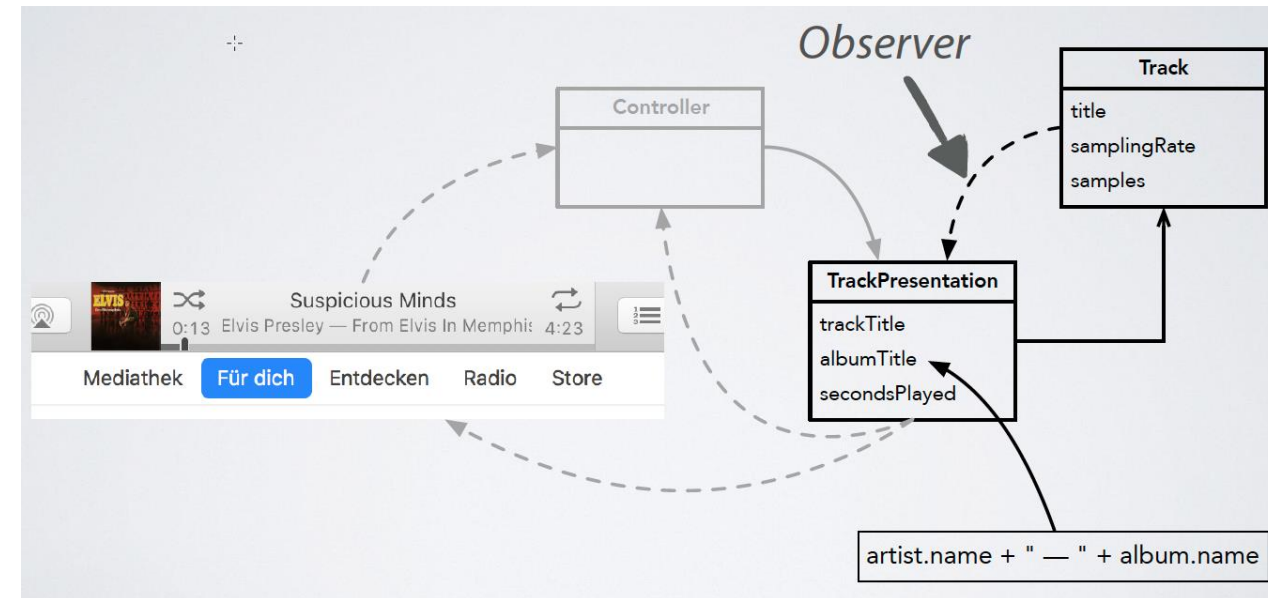
Das Modell

- Model -> Domain Model
 - Ein Model pro Anwendung
- Wird schnell unübersichtlich
 - Wo sollen berechnete Werte gespeichert werden?
 - Wo der Zustand des UI's?
 - Wo I18n (Internationalisierung)?
- Diese UI-Daten ruinieren das Domain Model.
 - Das Domainmodel soll nur fachliche Daten enthalten.



Das Presentation Model

- Das Presentation Model löst das Problem
 - Wird vor dem Domain Model positioniert
- Enthält alle darzustellenden Daten plus den Zustand des UI's
 - Minimiert die Logik des Views
 - Hält das Domain Model sauber

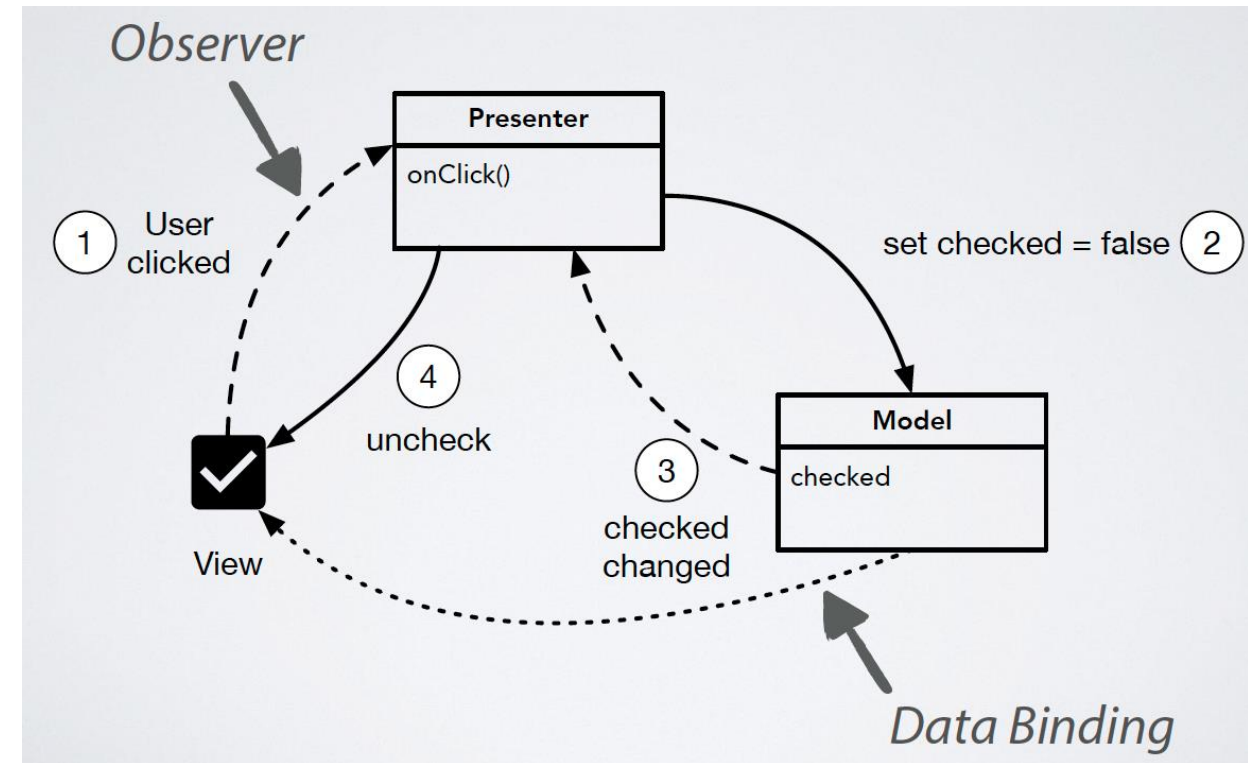


Model View Presenter (MVP)

- MVP ist die ursprüngliche Bezeichnung
- Zwei Varianten
 - Supervising Controller
 - Passive View

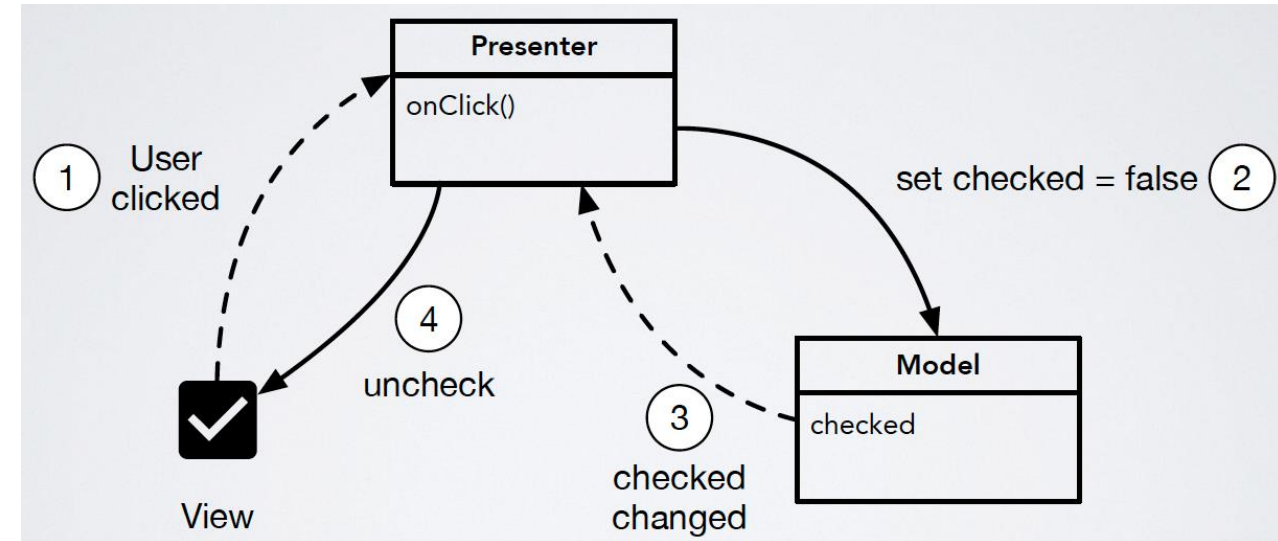
MVP - Supervising Controller

- Unterschied MVC zu Supervising Controller
 - View wird aktualisiert mittels Data Binding anstelle von Observer
 - Presenter manipuliert die View direkt
- Verwendung eines Presentation Models ist empfohlen.
- Erhöht die Testbarkeit, weil sich die Logik nicht mehr in der View befindet.



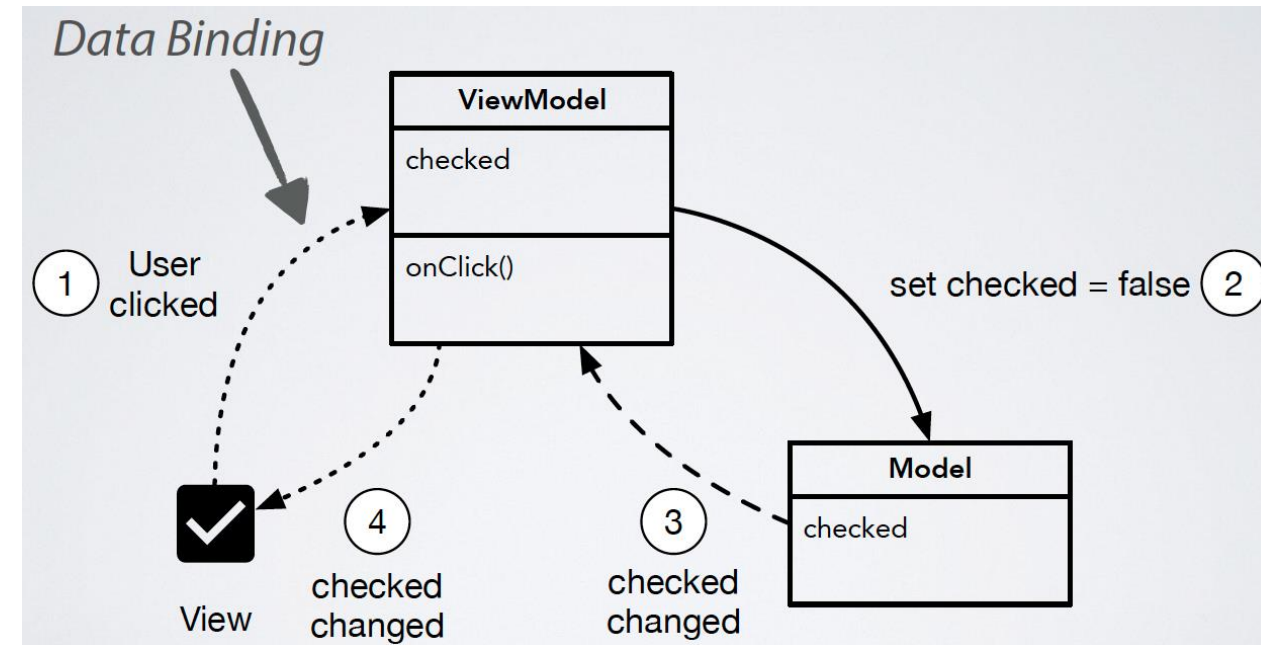
MVP - Passive View

- Im Vergleich zu Supervising Controller
 - Die View enthält keine Logik
 - Kann sich selber nicht aktualisieren
 - Der Presenter enthält alle Logik für die Aktualisierung des Views
- Verwendung eines Presentation Models ist empfohlen
- Erhöht die Testbarkeit, weil sich die Präsentations-Logik nicht mehr in der View befindet



Model View ViewModel (MVVM)

- Eng verwandt mit dem Supervising Controller
- ViewModel
 - Ersetzt den Supervising Controller
 - Enthält ein Presentation Model
 - View kommuniziert mit dem ViewModel durch DataBinding



```
private void initBindings() {
    label.textProperty().bind(studentViewModel.labelTextProperty());
    input.textProperty().bindBidirectional(studentViewModel.inputTextProperty());
    button.disableProperty().bind(studentViewModel.buttonDisabledProperty());
}
```

Agenda

1. Einführung in GUI-Architekturen
2. Patterns in GUI-Architekturen
3. Architektur-Patterns MVC, MVP und MVVM
4. **Kurzer Recap JavaFX**
5. Wrap-up und Ausblick

JavaFX und JDK

- JavaFX 8 ist/war ein Teil von Oracle JDK 8, 9 und 10
- JavaFX-Library muss separat ins Projekt eingebunden werden.
 - Ab Open JDK 8
 - Ab Oracle JDK11
- JavaFX Download (OpenFX) von
 - <https://gluonhq.com/products/javafx/>
 - JavaFX 11 LTS -> Long Term Support

JavaFX kann **direkt** gestartet werden

Oracle JDK8 -10

JavaFX 8
Teil des JDK8-10

JavaFX benötigt einen **Launcher** bei non-modularen Projekten

OpenJDK 8--xx
und OracleJDK
11 – xx

JavaFX Version xx
JavaFX Libraries
müssen eingebunden
werden

Agenda

1. Einführung in GUI-Architekturen
2. Patterns in GUI-Architekturen
3. Architektur-Patterns MVC, MVP und MVVM
4. Kurzer Recap JavaFX
- 5. Wrap-up und Ausblick**

Wrap-up

- GUI-Architekturen werden in der **Presentation-Schicht** bzw. UI angewendet.
- Die gängigen GUI-Architekturen basieren auf **bewährten Patterns**.
- Die Patterns erlauben die Umsetzung von «**Separation of Concern**». Das bekannteste Pattern ist das **MVC**. Weitere Pattern sind das **MVP** und **MVVM**.
- Die Reaktion auf Ereignisse kann durch das **Observer-Pattern** hergestellt werden. Das Pattern wird in vielen Frameworks angewendet.
- Bei der Frontend-Programmierung steht die **Reaktion auf Ereignisse** im Vordergrund. Die Umsetzung kann mit Hilfe von **anonymen Klassen** oder vereinfacht durch **Lambda-Ausdrücke** erreicht werden.
- Bei JavaFX kann ein **View imperativ** durch die Instanziierung der Steuerelemente erstellt werden. Mit Hilfe von FXML steht eine **deklarative Möglichkeit** zur Verfügung.

Ausblick

- In der nächsten Lerneinheit werden wir:
 - das Thema der Persistenz vertiefen.

Quellenverzeichnis

- [1] Martin Fowler, GUI Architectures, <https://martinfowler.com/eaDev/uiArchs.html>, 2006
- [2] Gluon, JavaFX, <https://gluonhq.com/products/javafx/>
- [3] Fowler, M.: Patterns of Enterprise Application Architecture, Addison Wesley, 1. Auflage, 2002