Swing: Daten binden

Karsten Lentzsch
www.JGoodies.com

Ziel

Ansätze kennenlernen
wie man in Swing
Fachdaten
mit der Oberfläche verbinden kann

Lösungen bewerten können

Vorstellung

- Ich baue Swing-Anwendungen,
  die viele Leute elegant finden
- arbeite seit 1990 mit Objekten
- helfe Anderen über und unter der Haube
- biete Bibliotheken, die Swing ergänzen
- biete Swing-Beispiele zu Architekturen
- und schreibe über Desktop-Themen

Gliederung

- Einleitung
- Grundlagen
- Werte binden
- Listen binden
- 3-Schichten-Architektur
- Erfahrungsbericht

Swing-Bausteine

Swing-Bausteine

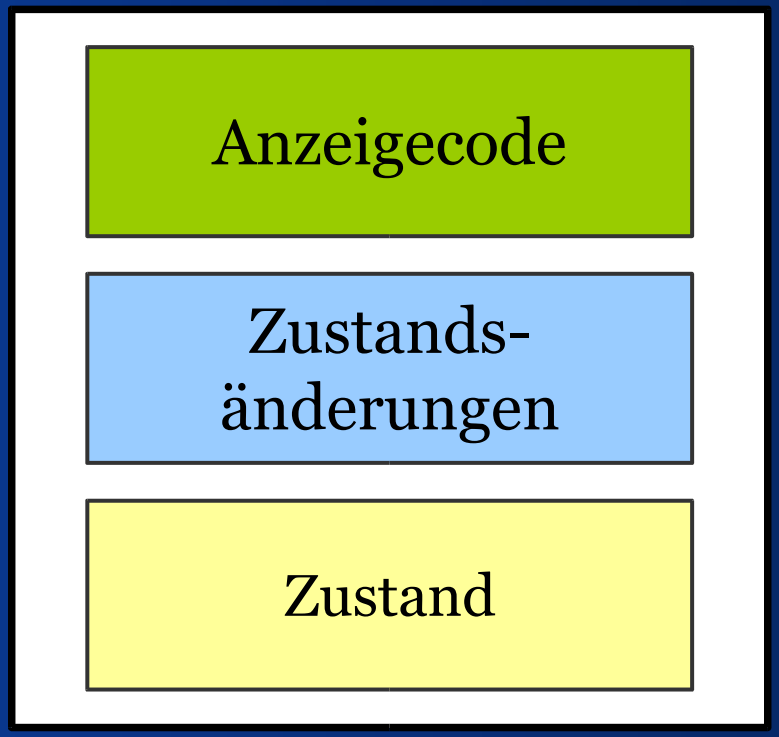
Fragen

- Wie hängen MVC und Swing zusammen?
- Wie gliedere ich meine Anwendung?
- Was gehört ins Modell?
- Wie teile ich Modelle auf?
- Wie baue ich einen View?
- Was tut ein Controller?
- Brauche ich einen Controller?

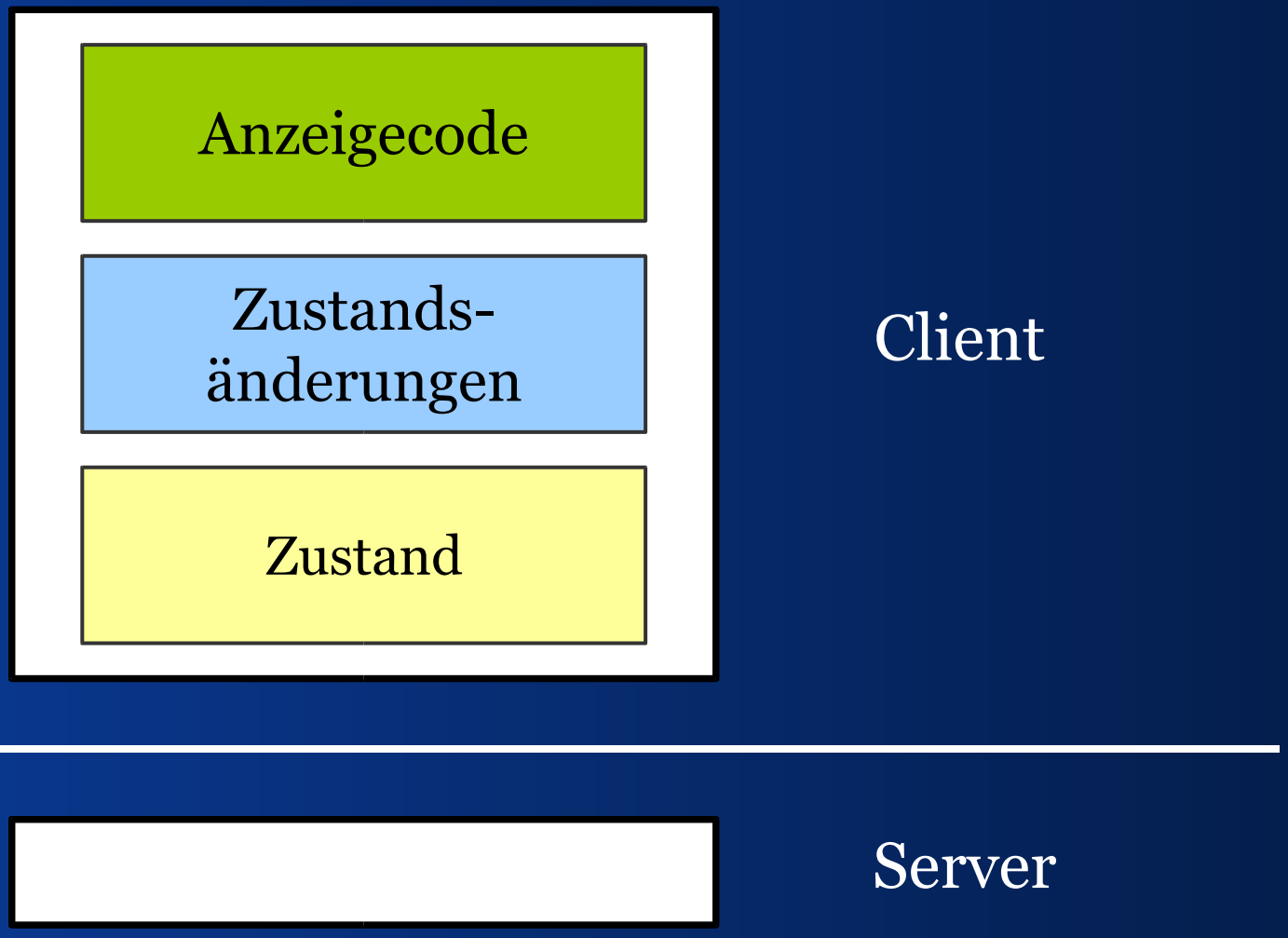
I - Grundlagen

MVC und Swing

Vor MVC



Vor MVC: 2 Schichten

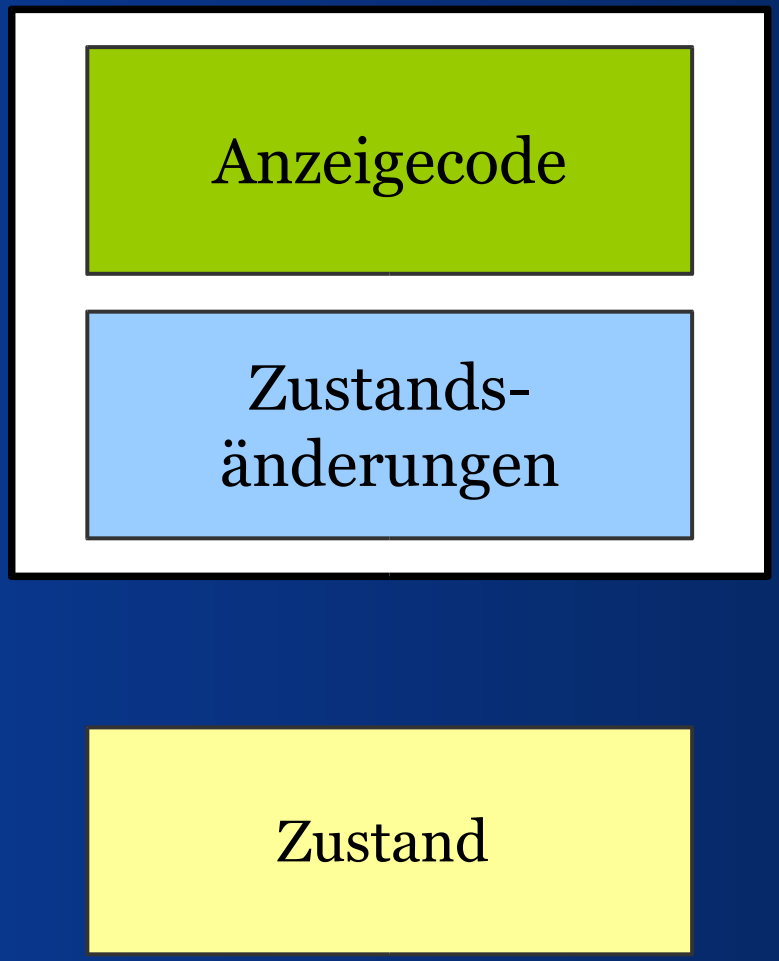


Fachlogik abtrennen

- Fachlogik enthält keinen GUI-Code
- Präsentation handhabt die Oberfläche

- Vorteile:
  - Die Teile sind leichter zu verstehen
  - Die Teile sind leichter zu ändern

Fachschicht und Präsentation



Präsentationsschicht

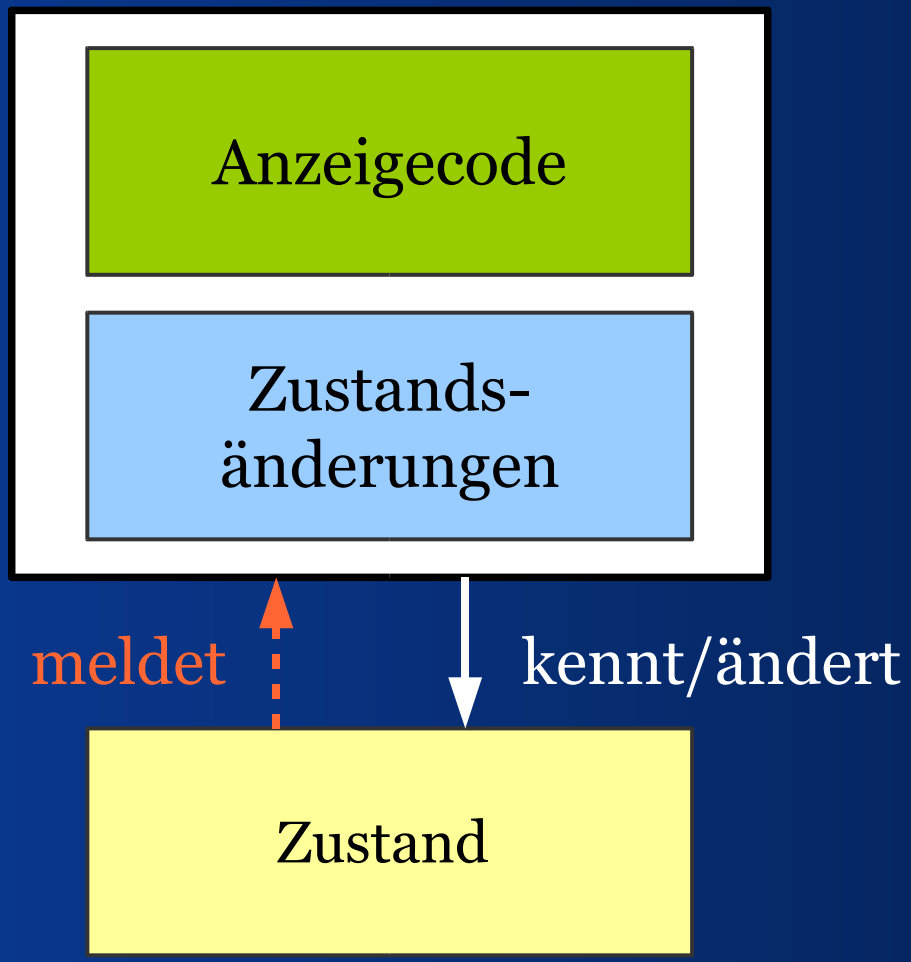
Fachschicht

Präsentation entkoppeln

- Fachlogik referenziert keinen GUI-Code
- Präsentation kennt und ändert die Fachlogik

- Vorteile:
  - Reduziert Komplexität
  - Erleichtert mehrere Präsentationen einer Fachlogik

Fachschicht und Präsentation

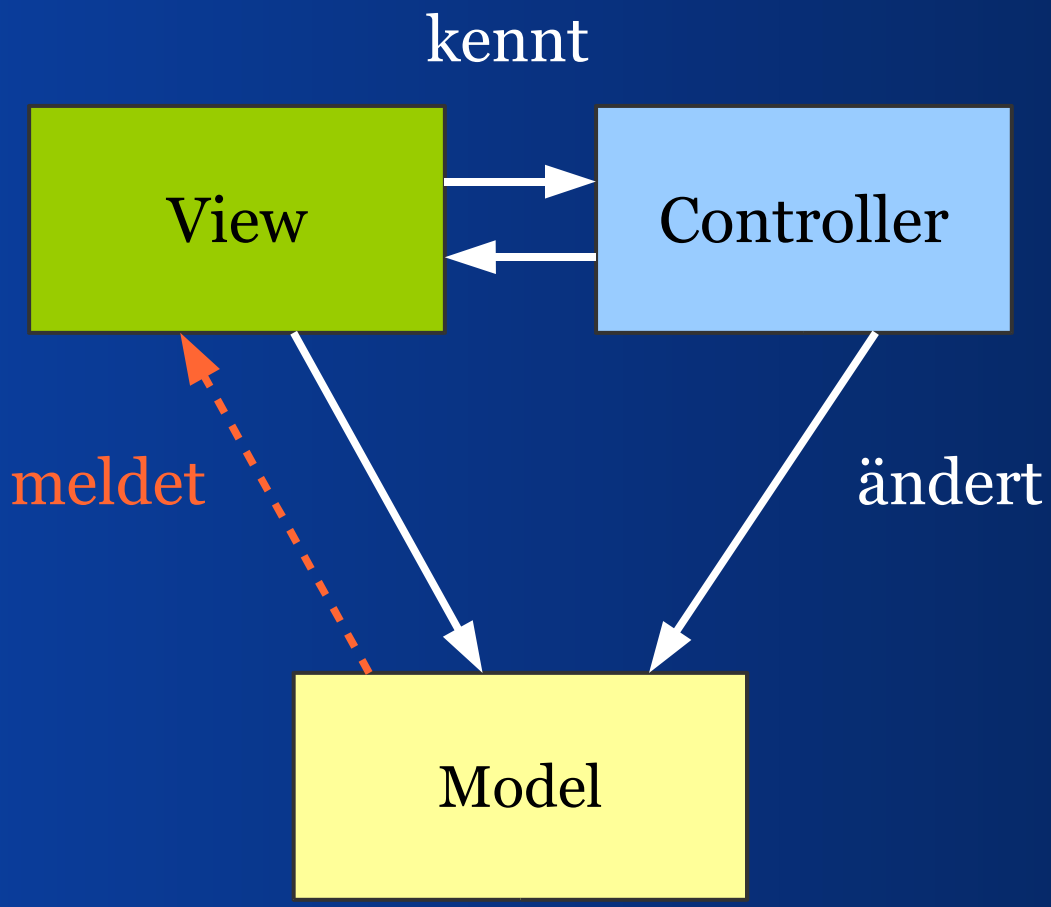


View und Controller trennen

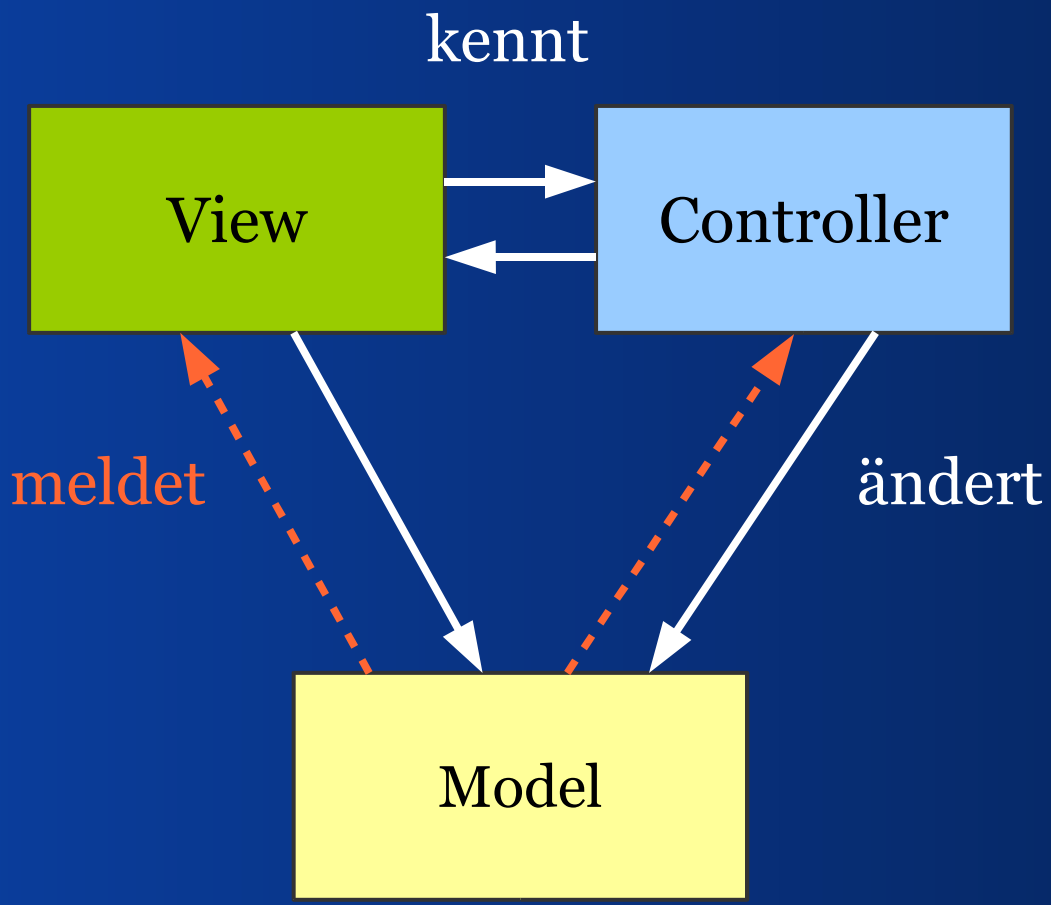
Trennt man den Zeichencode (View) von Operationen auf dem Zustand (Controller), dann kann man:

- View und Controller freier kombinieren
- Beide Teile leichter wieder verwenden

MVC



MVC

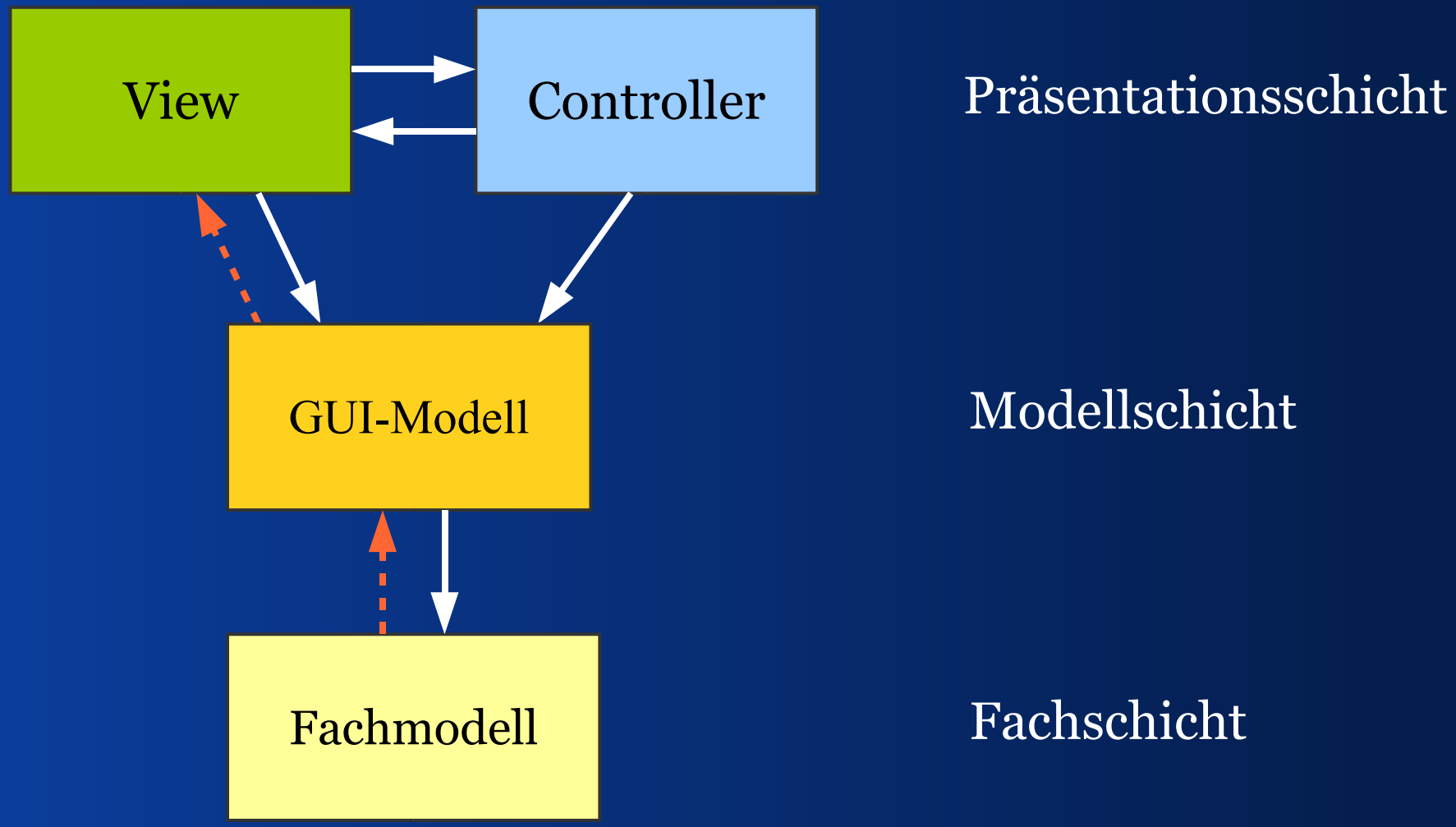


Oberflächenmodelle abtrennen

Man kann Modelle kategorisieren danach, ob sie Fachdaten betreffen oder die GUI.

Daraus entsteht eine weitere Schicht.

MVC mit Modellschicht



Kandidaten für Modellschicht

- Für die Oberfläche aufbereitete Fachobjekte, z. B. Baummodell aus File-Instanzen

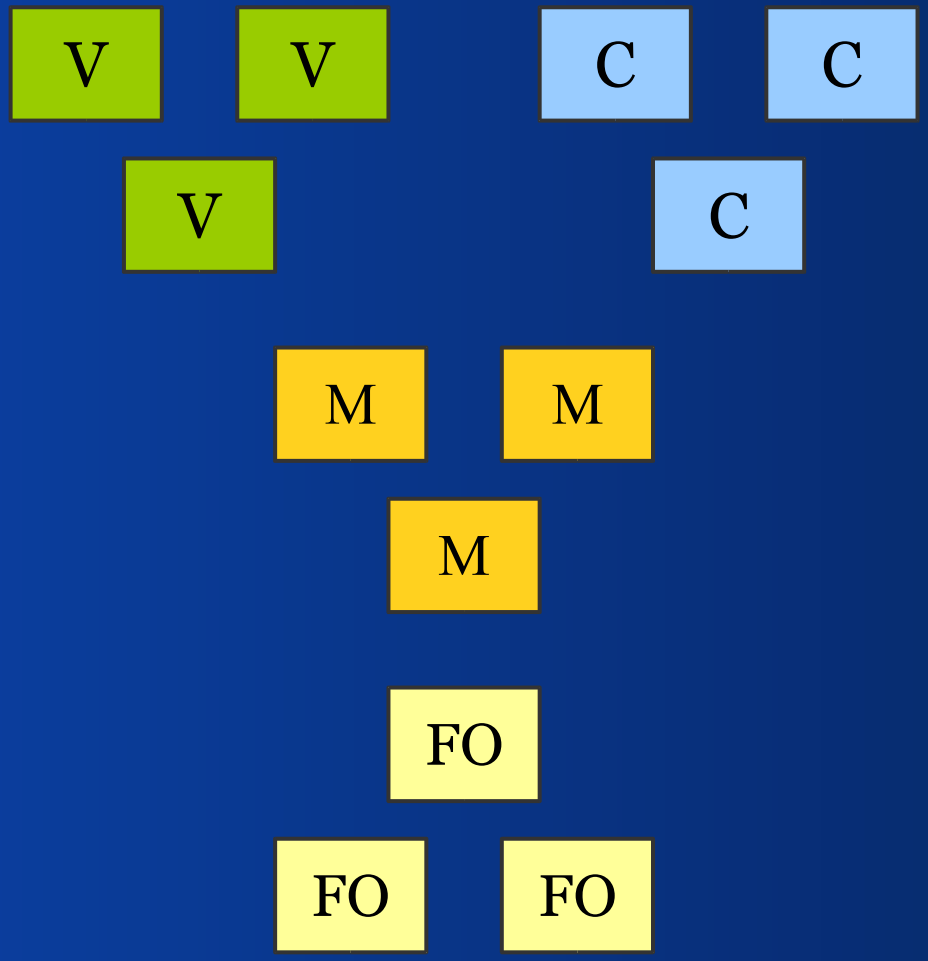
- Modelle, die nicht zur Fachlogik gehören:
  - Kennwort in einem Anmeldedialog
  - ein Suchbegriff
  - GUI-Zustand, z. B. Maus gedrückt

MVC-Triaden kombinieren

Eine MVC-Oberfläche kombiniert Triaden.

- Setzt Modelle als Graph zusammen
- Baut große Views aus kleinen Views
- Nutzt Untercontroller

MVC-Triaden mit Modellschicht



Präsentationsschicht

Modellschicht

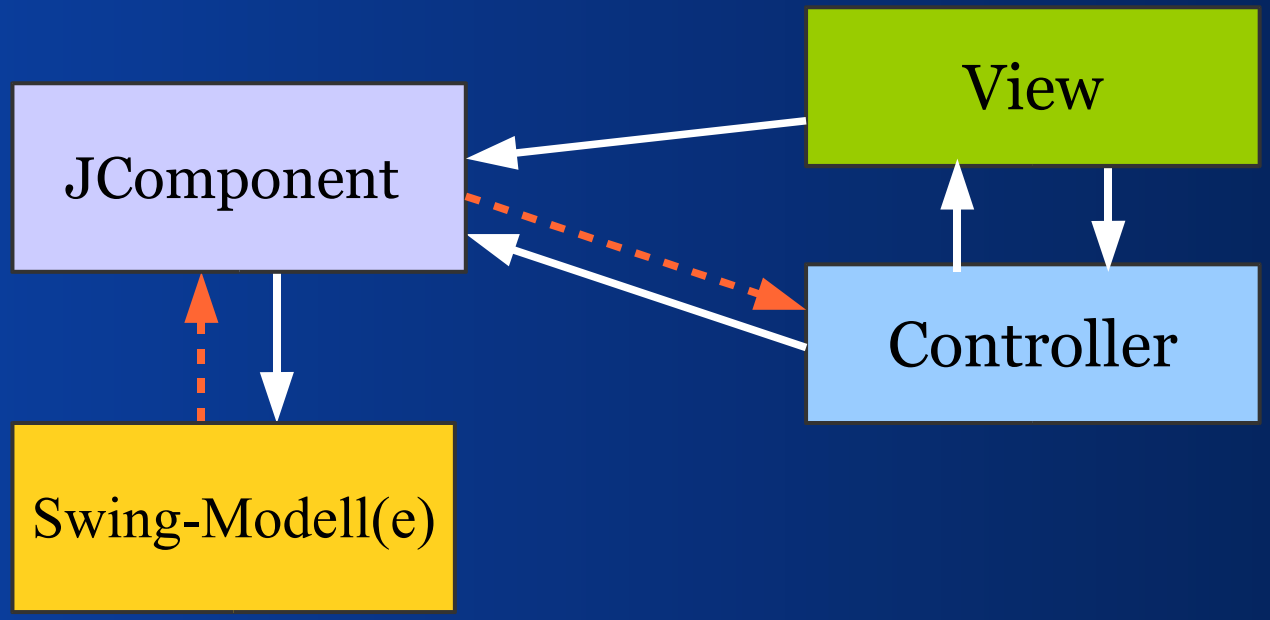
Fachschicht

Look&Feel abtrennen

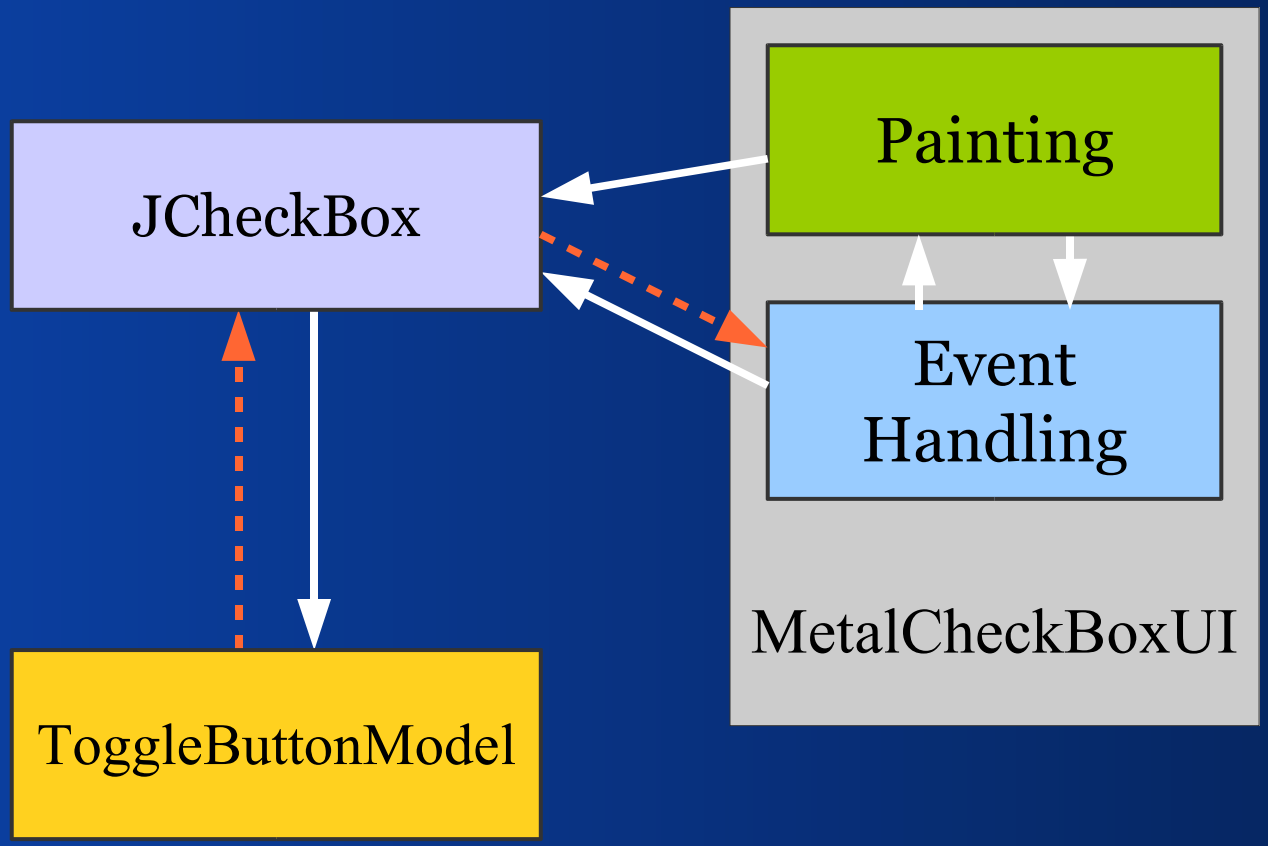
Swing kann Aussehen und Verhalten wechseln.

- Views und Controller sind abgetrennt.
- Views nutzen eine Basisimplementierung.

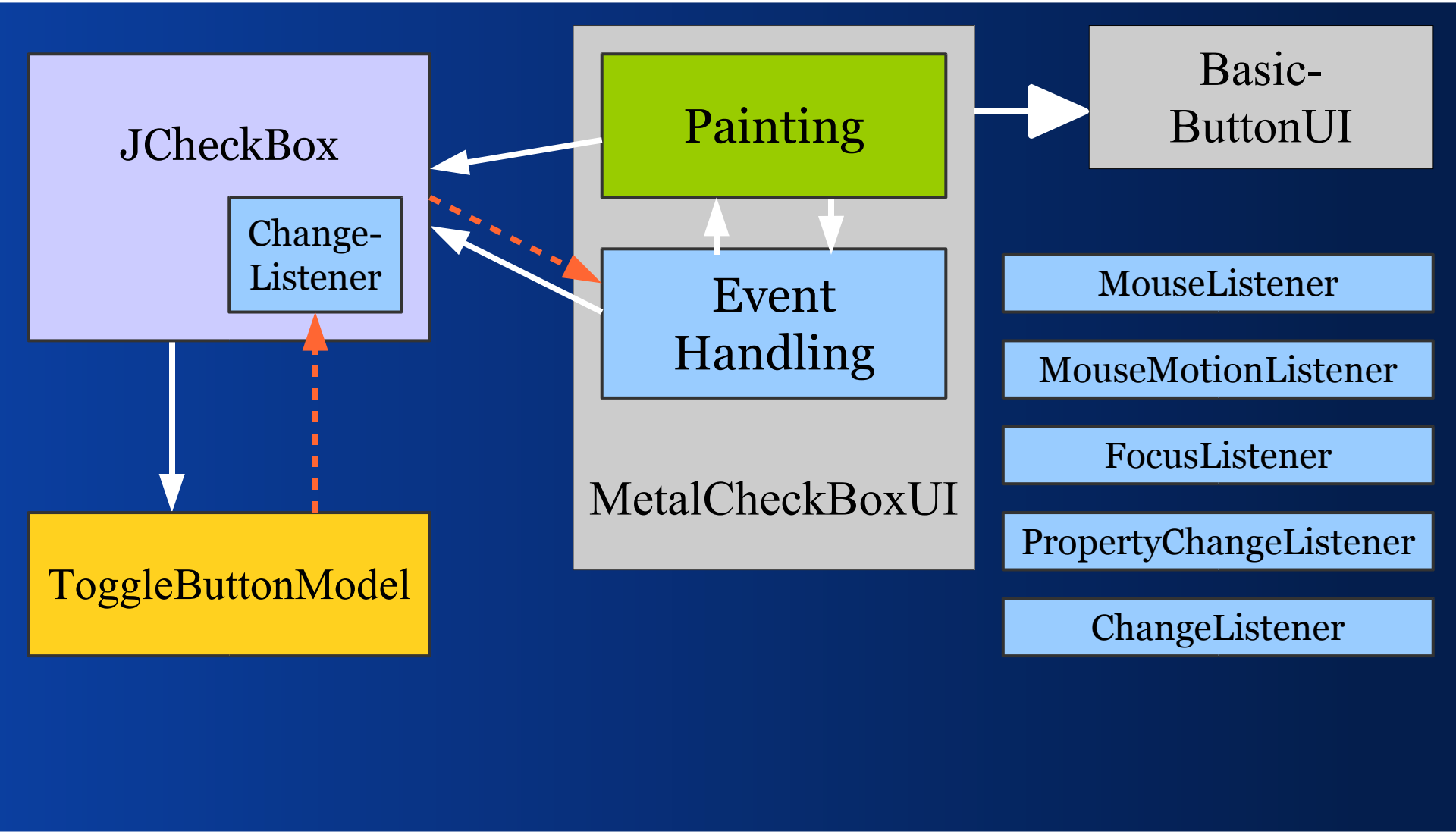
M-JComponent-VC



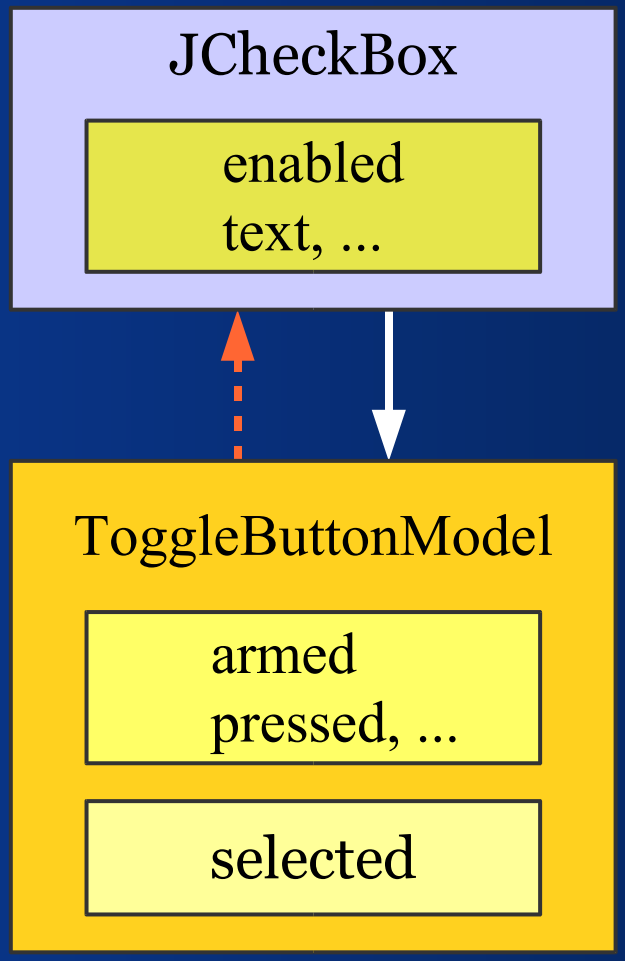
Beispiel: JCheckBox



JCheckBox: Einige Details



JCheckBox: Zustandsarten

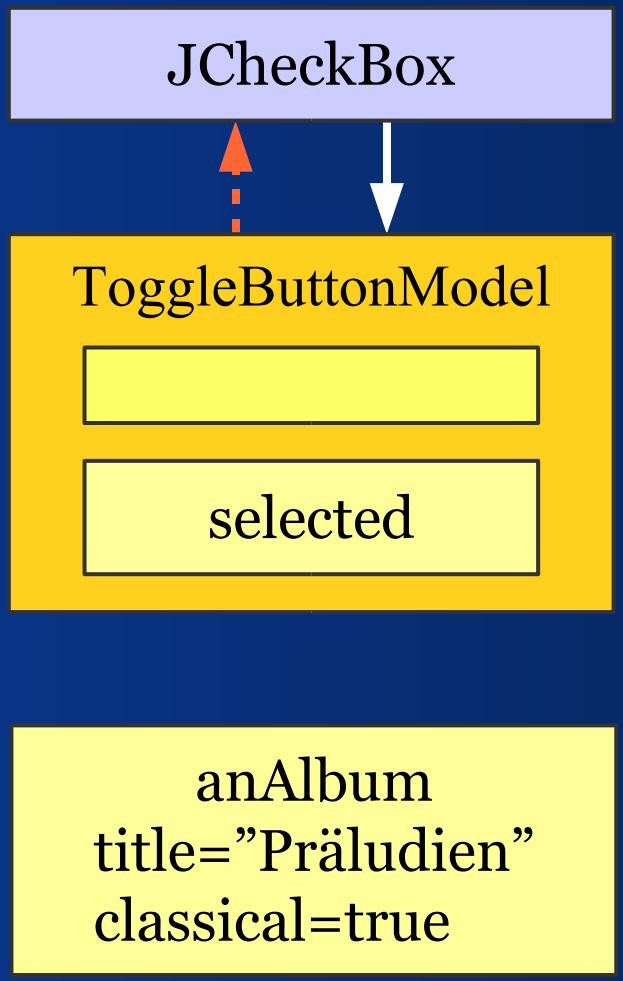


GUI-Eigenschaften

GUI-Zustand

Datenzustand

JCheckBox: Bindeaufgabe

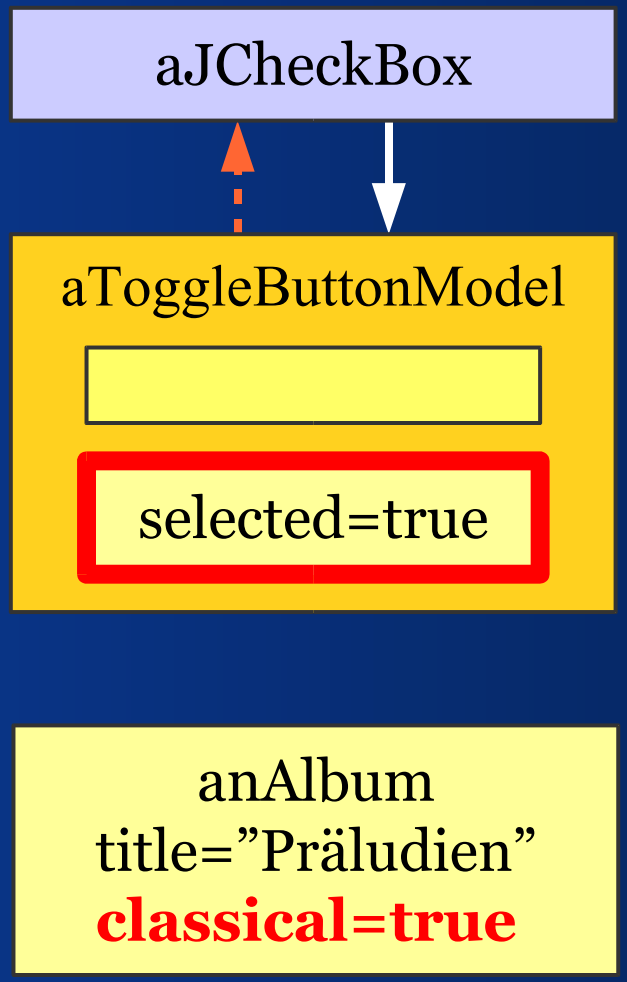


GUI-Komponente

Datenmodell

Fachdaten

JCheckBox: Bindeaufgabe



GUI-Komponente

Datenmodell

verbinden,
synchronisieren

Fachdaten

Fazit

- Swing verwendet nicht das Original-MVC
- Swing nutzt eine erweiterte MVC-Form
- Swing teilt aber die Motivation hinter MVC
- Swing bietet etwas mehr als MVC

Darum suchen wir eine Bindelösung
für Swing, nicht für MVC.

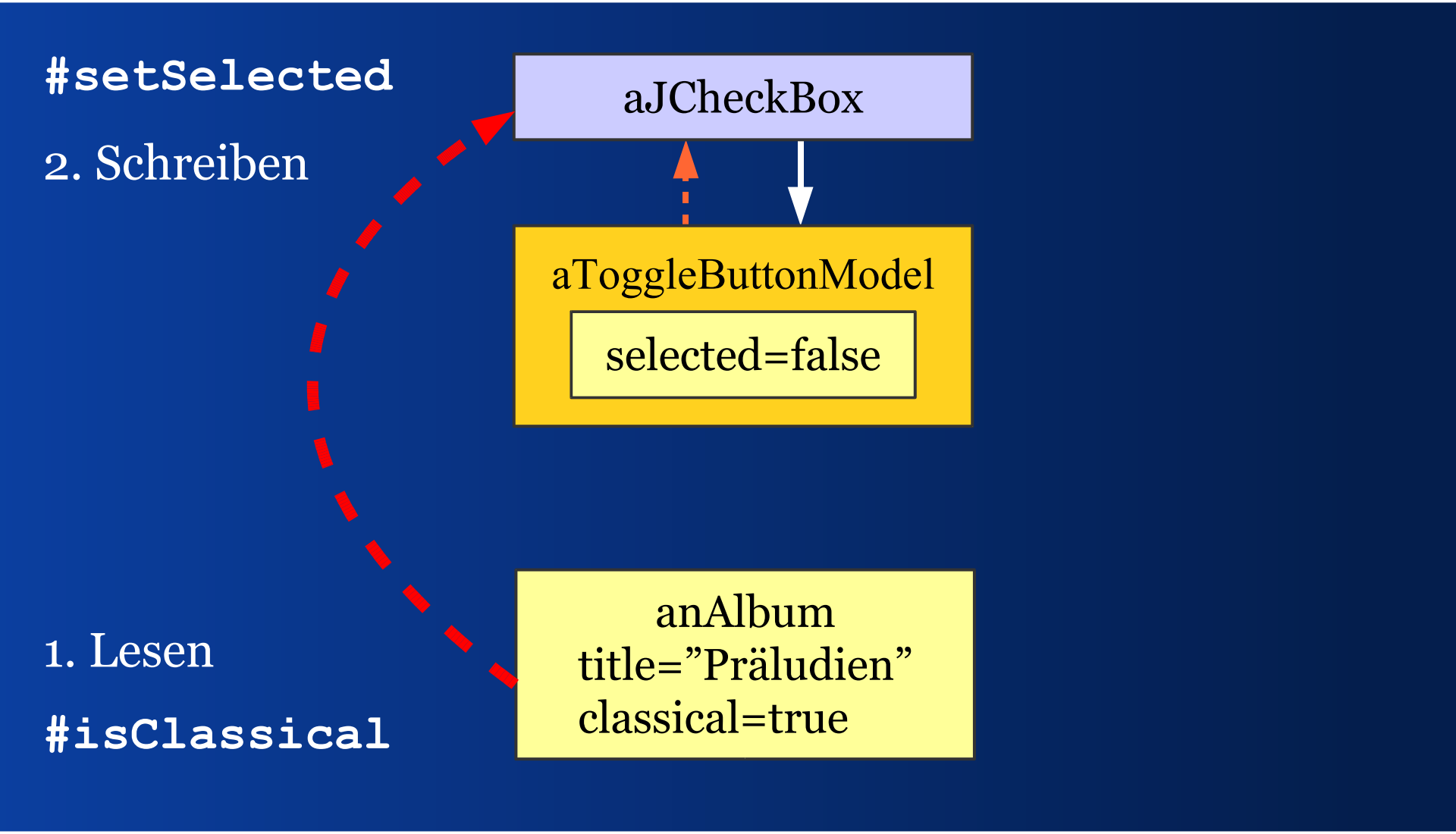
II - Werte binden

Wie verbinde ich Facheigenschaften mit Komponenten?

Bindeaufgaben

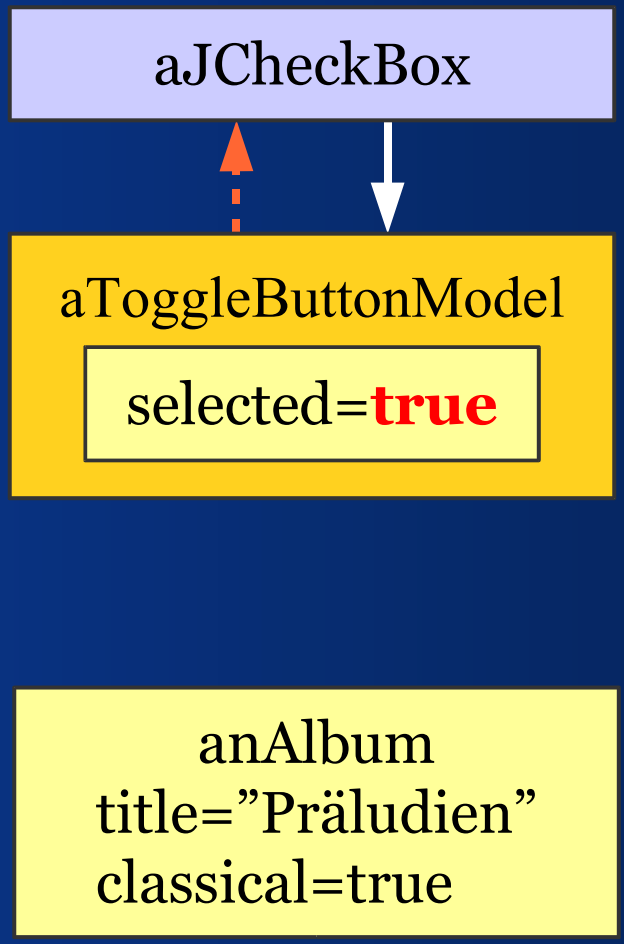
- Fachdaten lesen und schreiben
- GUI-Modellwerte lesen und setzen
- Fachdatenänderungen melden/behandeln
- Werte puffern – bis OK zurück halten
- Änderungsmanagement – OK nötig?
- Indirektion wie in Übersicht-Detail-Views
- Typen konvertieren, z. B. Datum nach Text

Werte zum View kopieren

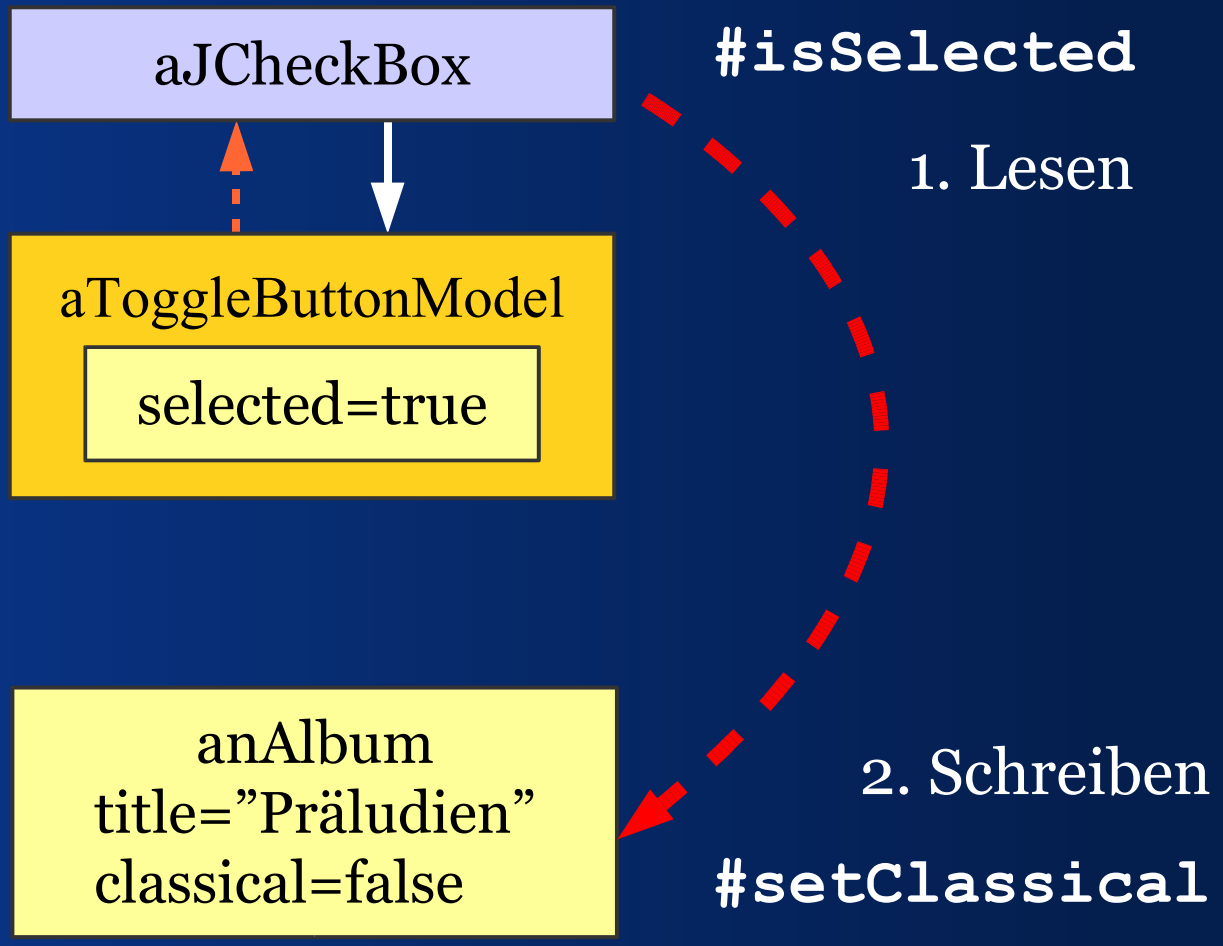


Werte zum View kopieren

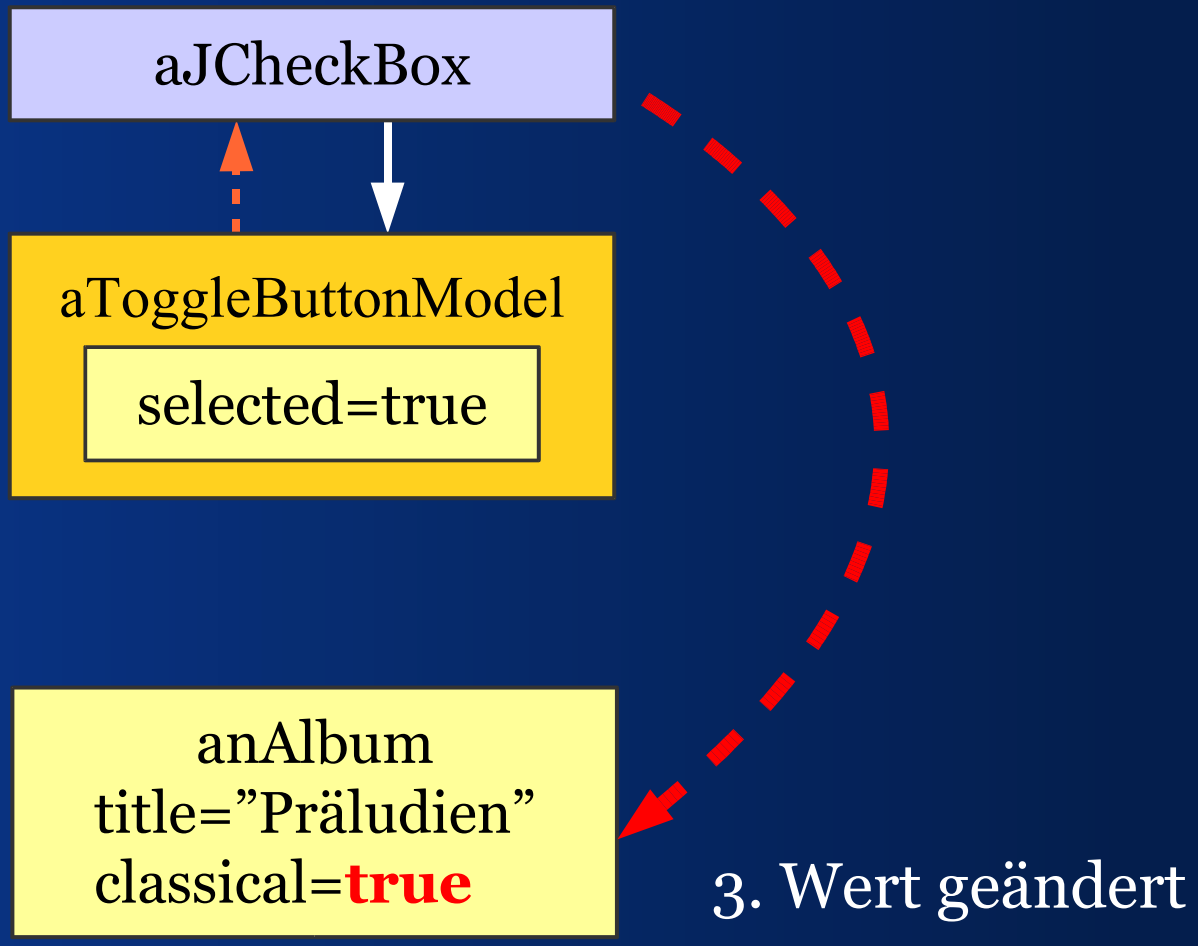
3. Wert geändert



Werte zurück kopieren



Werte zurück kopieren



Beispiel zum Kopieren

```
public void modelToView() {

    Album anAlbum = getEditedAlbum();

    classicalBox.setSelected(
        anAlbum.isClassical());

    titleField.setText(
        anAlbum.getTitle());
    ...
}
```

Beispiel zum Kopieren

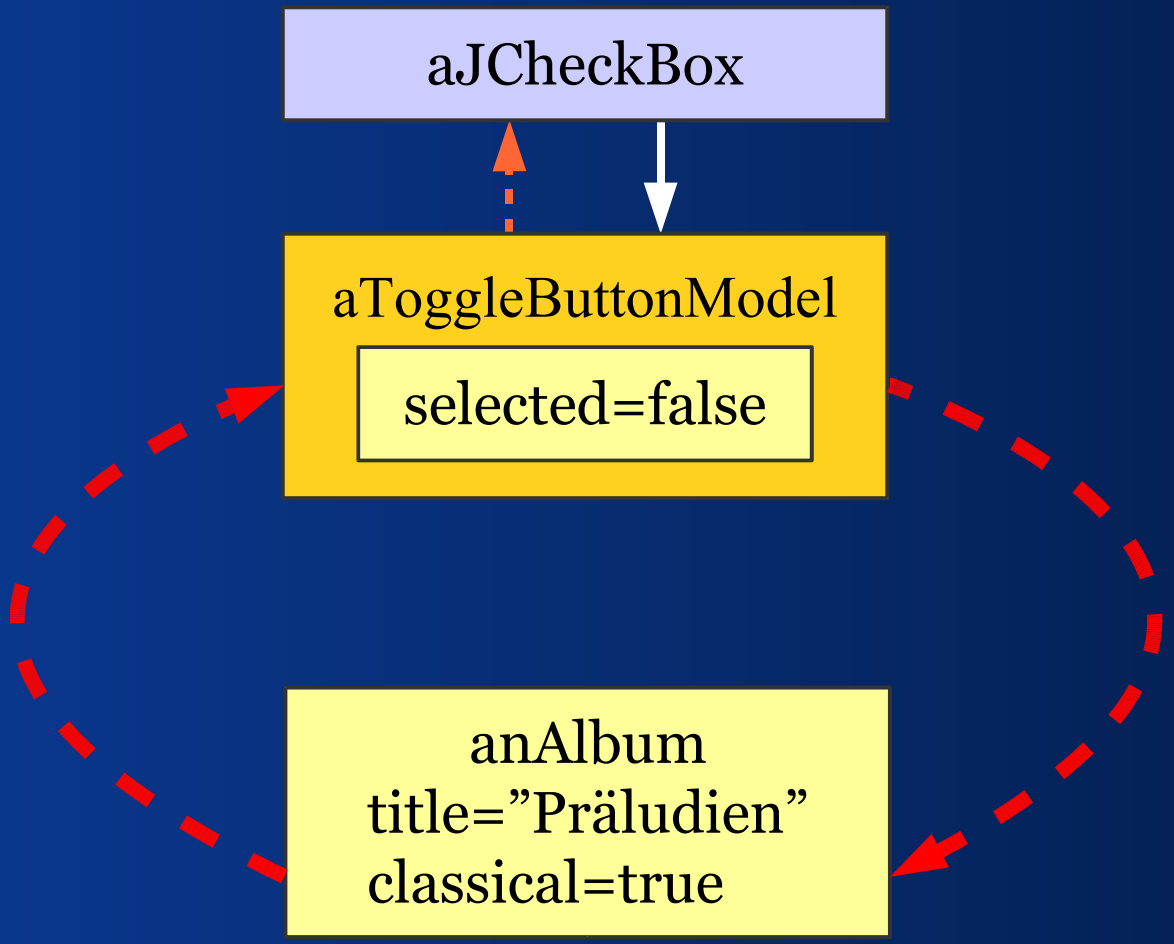
```
public void viewToModel() {

    Album anAlbum = getEditedAlbum();

    anAlbum.setClassical(
        classicalBox.isSelected();

    anAlbum.setTitle(
        titleField.getText();
    ...
}
```

Kopieren: Vor- und Nachteile

- Einfach zu verstehen und zu erklären
- Funktioniert in fast allen Situationen
- Einfach zu debuggen

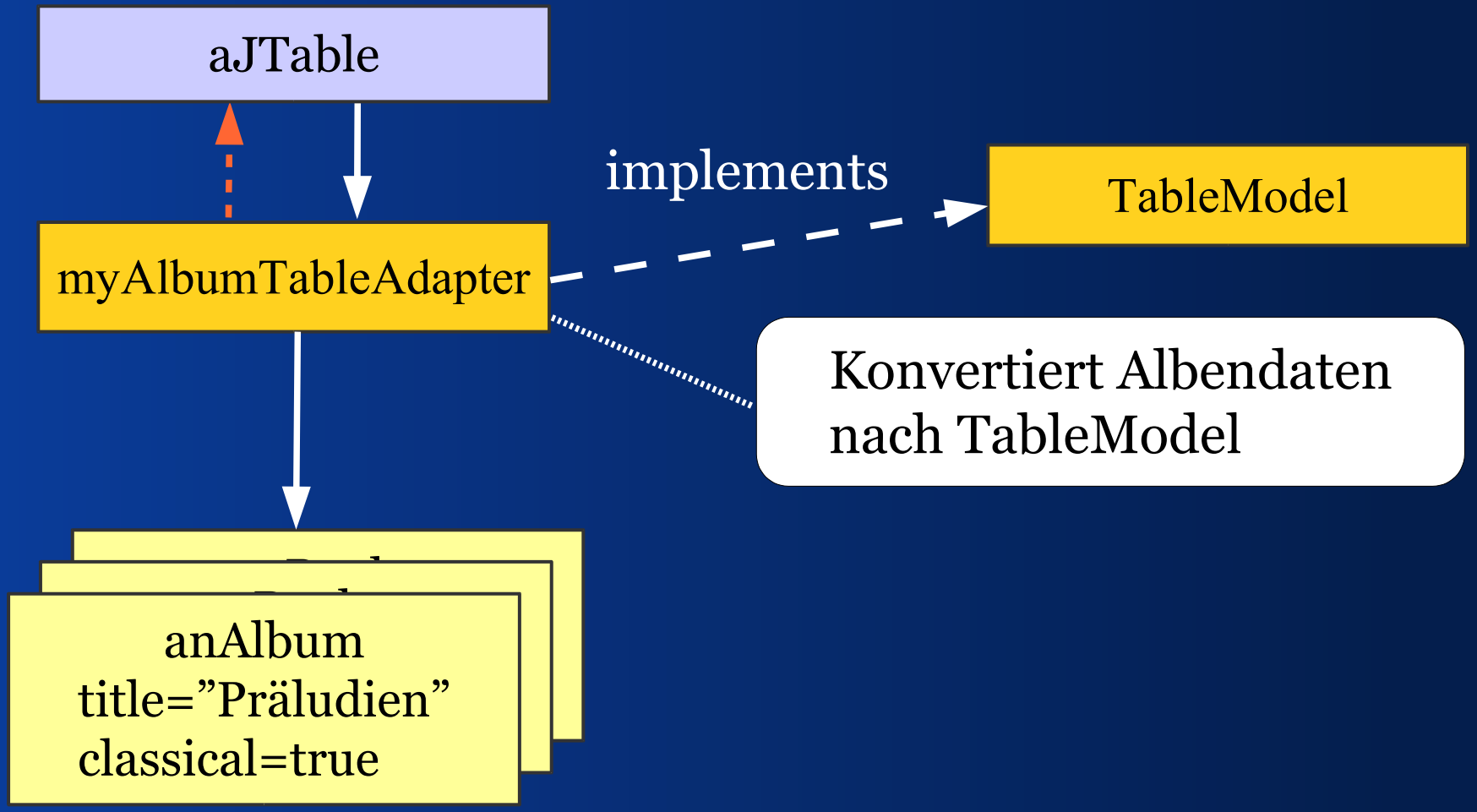
- Relativ viel Schreibarbeit
- Synchronisation aufwendig
- Reaktion auf Fachdatenänderung schwierig

Alternative

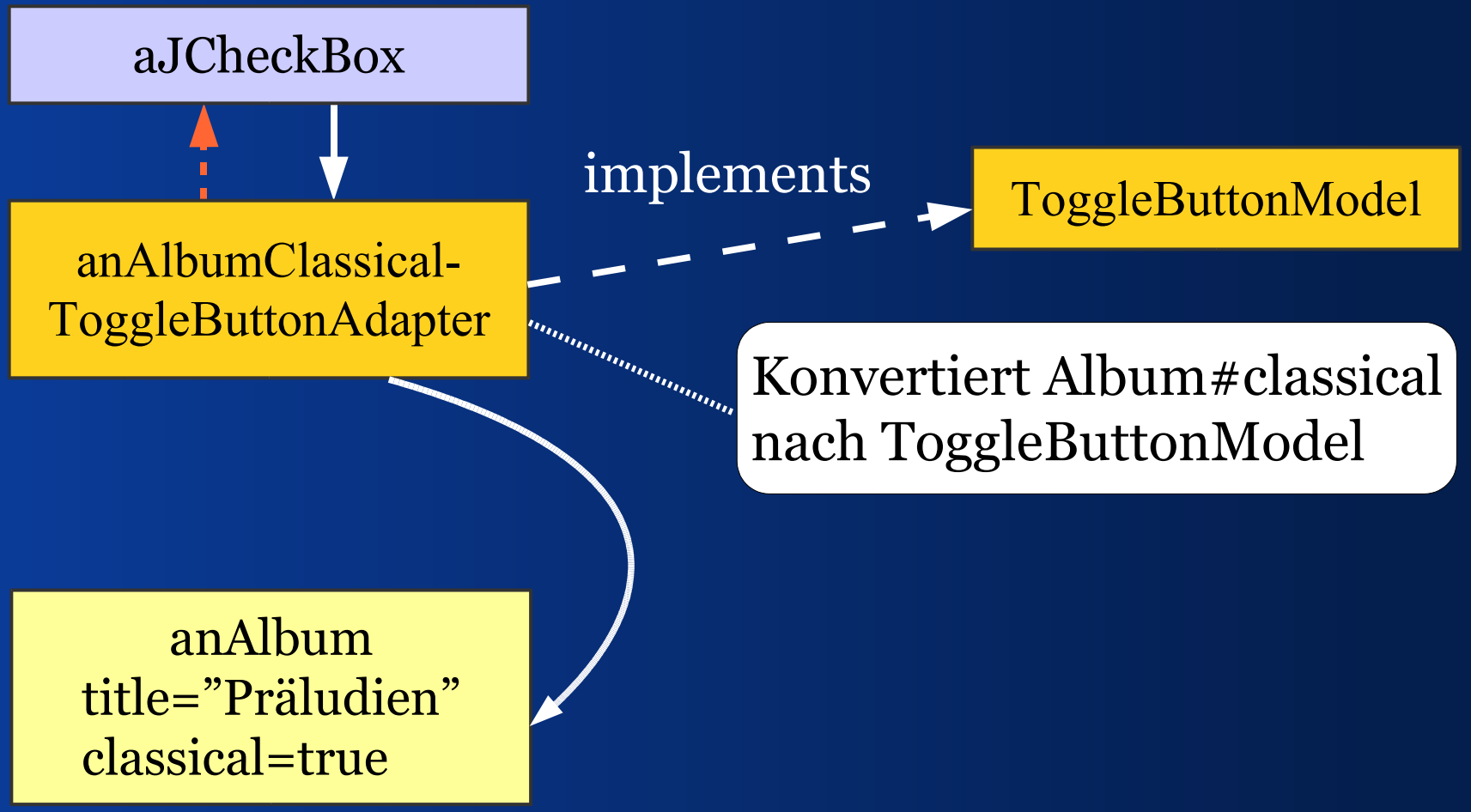


Beachte: man kann das Modell nicht teilen

Direktadapter: TableModel



Direktdapter: JCheckBox



Problem der Direktadapter

Für jede Fachdateneigenschaft muss ein individueller Adapter geschrieben werden.

Das ist in etwa so, als wenn alle Steckdosen in der Wand eine andere Form hätten.

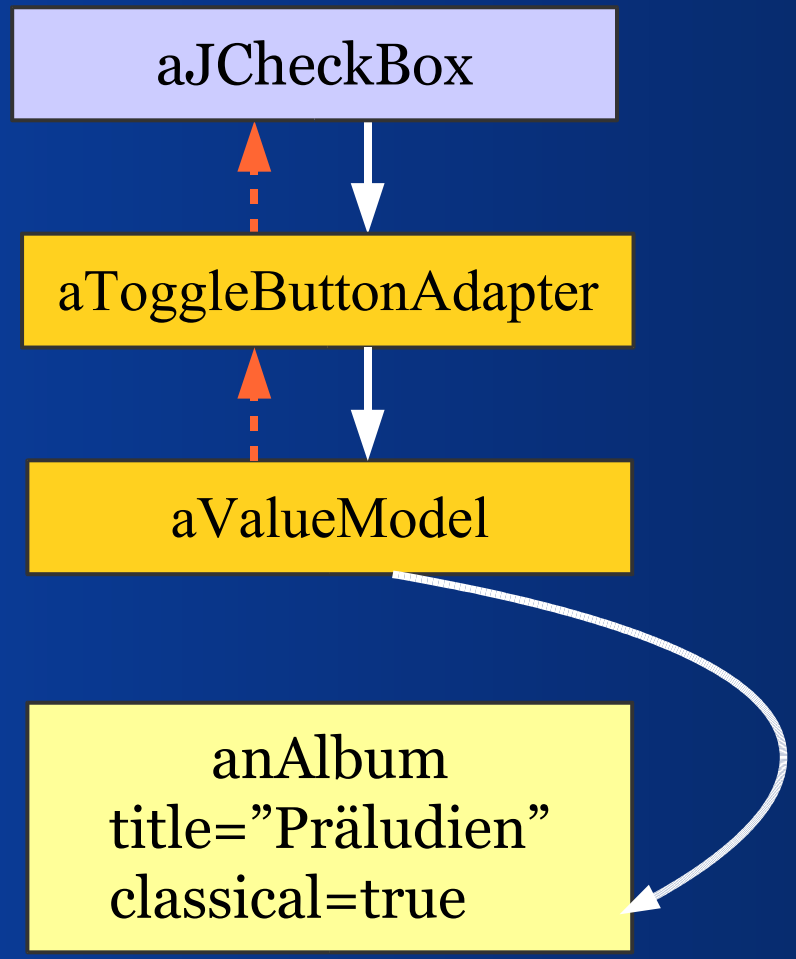
Code ist gleich, bis auf die Methoden zum Lesen und Schreiben der Facheigenschaften.

Konzept

- Nutze ein Universalmodell (ValueModel)
- Konvertiere Facheigenschaften nach ValueModel
- Baue Konverter von ValueModel zu den Swing-Modellen.

- Man braucht etwa 15 Klassen.

ValueModel und Adapter



ValueModel: Anforderungen

- Wir wollen einen Wert lesen
- Wir wollen einen Wert setzen
- Wir wollen Änderungen bemerken

Die Schnittstelle ValueModel

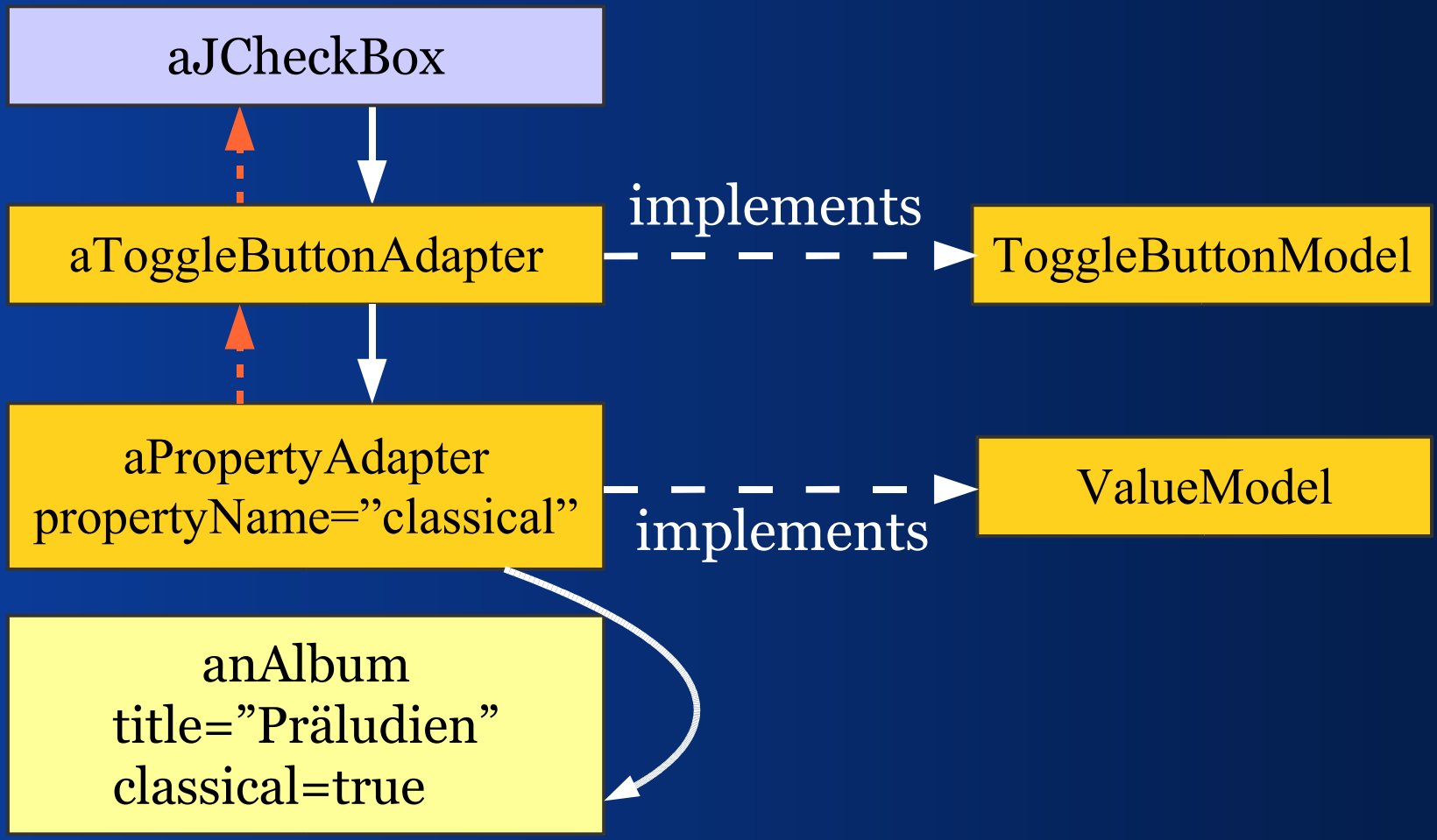
```
public interface ValueModel {

  Object getValue();

  void setValue(Object newValue);

  void addChangeListener(ChangeListener l);

  void removeChangeListener(ChangeListener l);
}
```

Welcher Event-Typ?

- ChangeEvent meldet keinen Wert;
  den muss man bei Bedarf holen.

- PropertyChangeEvent
  liefert den alten und neuen Wert;
  beide können aber **`null`** sein.

ValueModel & PropertyAdapter



Anforderungen an Fachobjekte

- Wir wollen Eigenschaften lesen und setzen
- Wir wollen das einheitlich tun
- Änderungen sollen gemeldet werden

Das bieten Java Beans.

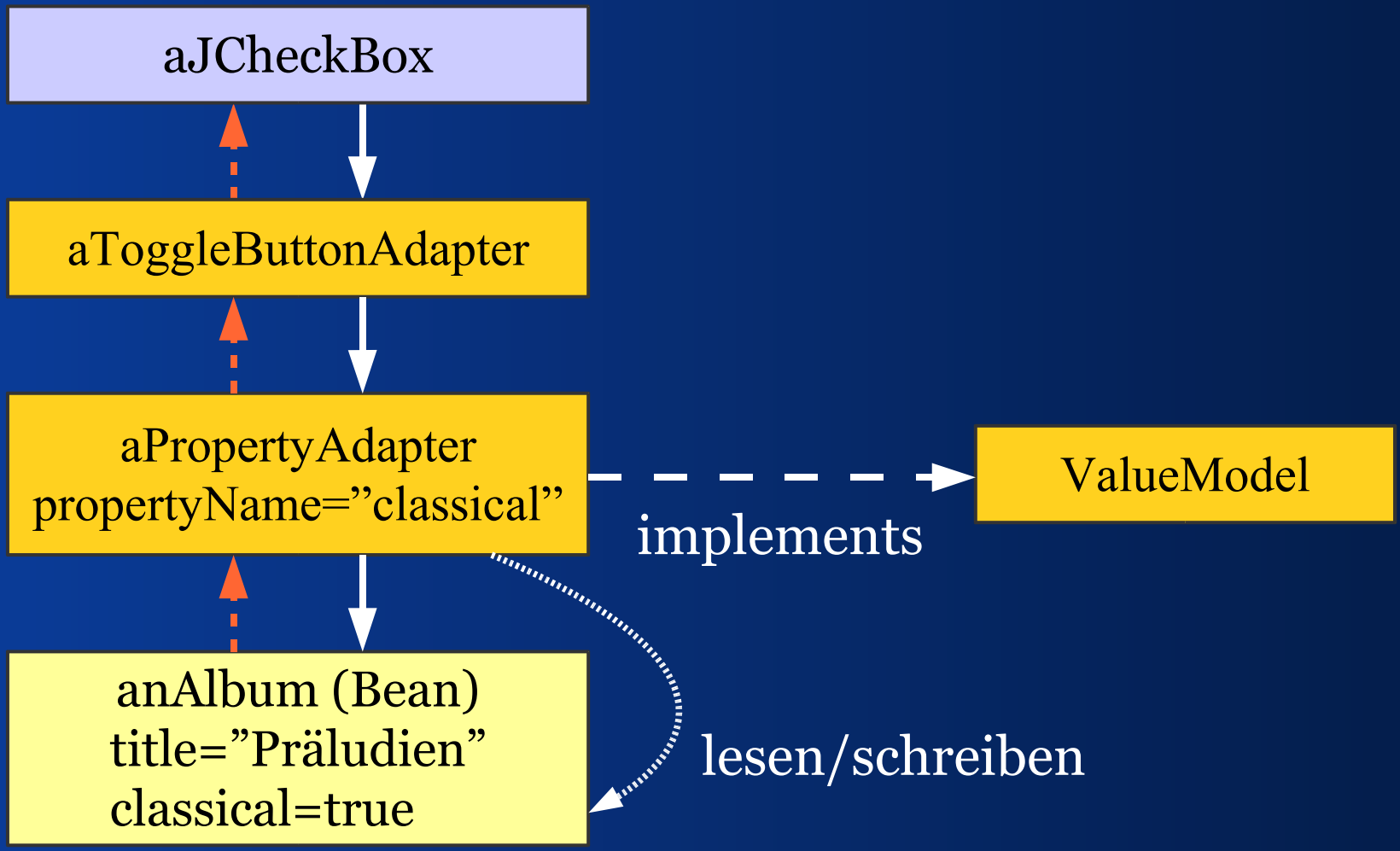
(Bound) Bean Properties

- Java Beans haben Eigenschaften, die man uniform lesen und schreiben kann.
- Bean-Eigenschaften heißen gebunden (bound), wenn man Änderungen mittels PropertyChangeListener beobachten kann.

PropertyAdapter

- `BeanAdapter` und `PropertyAdapter` konvertieren Bean-Eigenschaften nach ValueModel
- Beobachten Bound properties
- Nutzen Bean-Introspection, das wiederum Reflection nutzt, um Eigenschaften zu lesen und zu setzen

ValueModel & PropertyAdapter



Adapterkette bauen

```
private void initComponents() {

   Album album = getEditedAlbum()

   ValueModel aValueModel =
       new PropertyAdapter(album, "classical");

   JCheckBox classicalBox = new JCheckBox();
   classicalBox.setModel(
       new ToggleButtonAdapter(aValueModel));
}
```

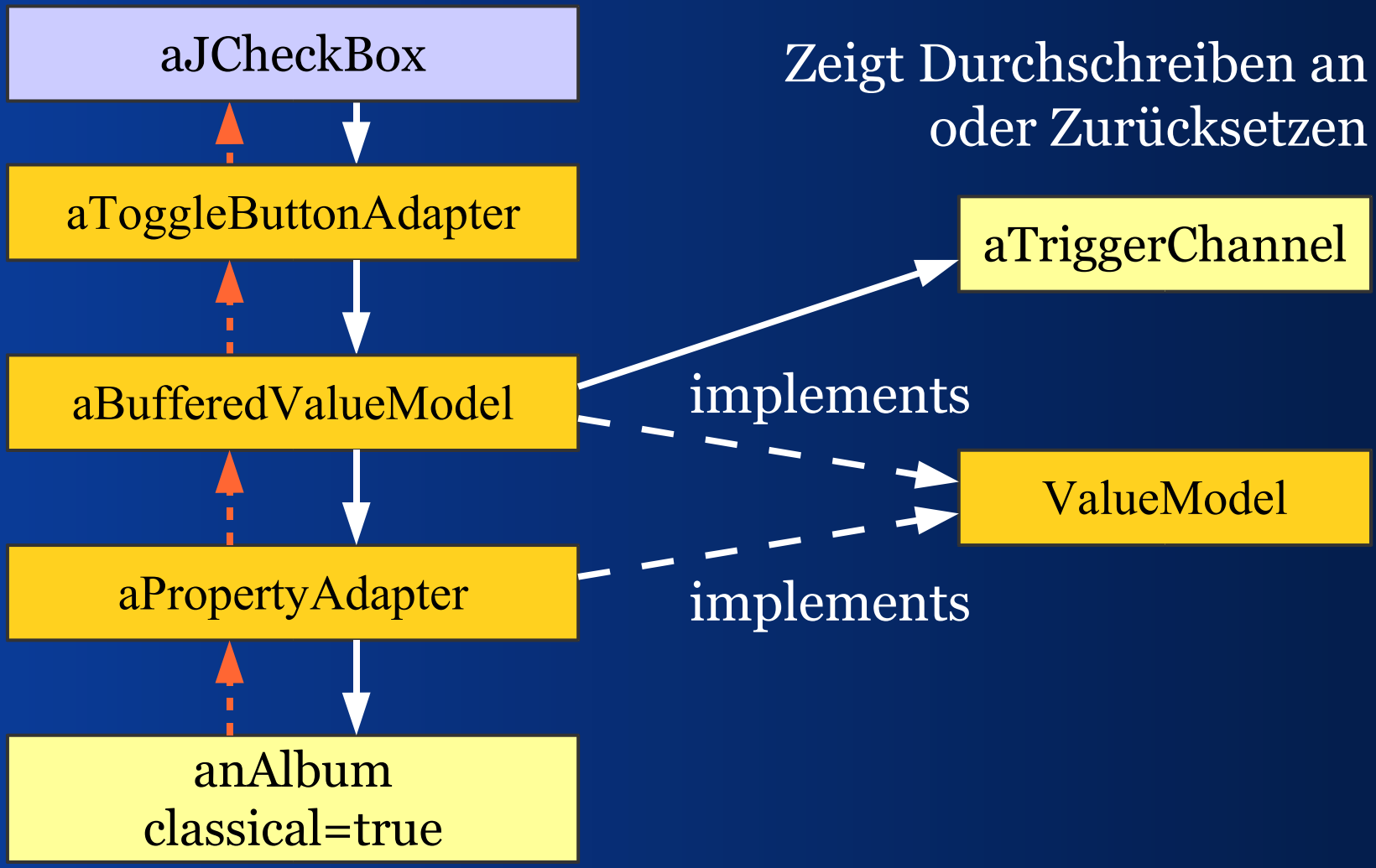
ComponentFactory

```
private void initComponents() {

   Album album = getEditedAlbum();

   JCheckBox classicalBox =
       ComponentFactory.createCheckBox(
           album,
           Album.PROPERTYNAME_CLASSICAL);
}
```

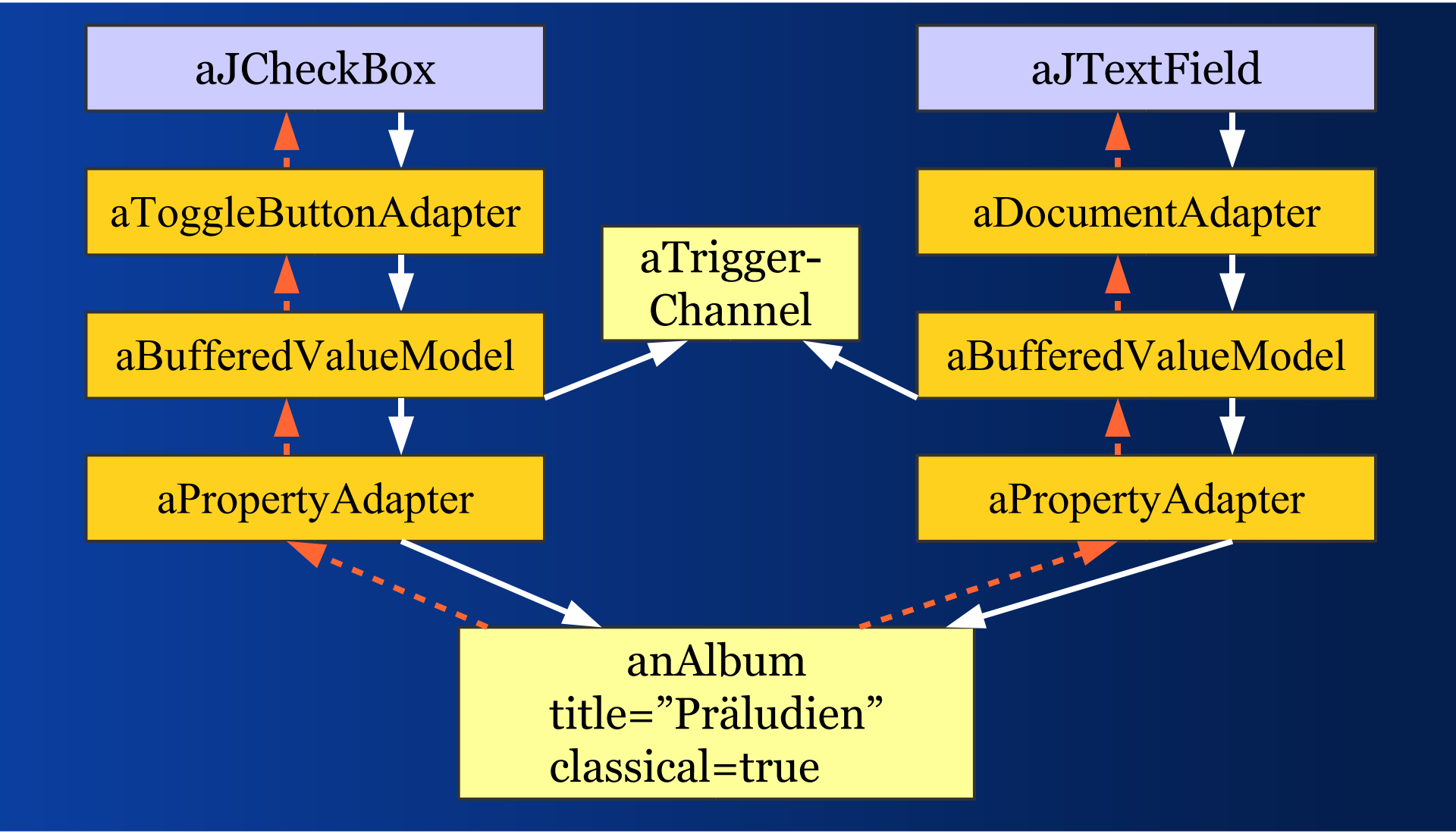
Puffern: Werte zurück halten

- Schalten wir die JCheckBox, wird die gebundene Facheigenschaft sofort geändert.
- Häufig will man Änderungen zurück halten, bis OK oder Übernehmen gedrückt wurde.
- Wir können in der Bindekette puffern oder in der Fachschicht.

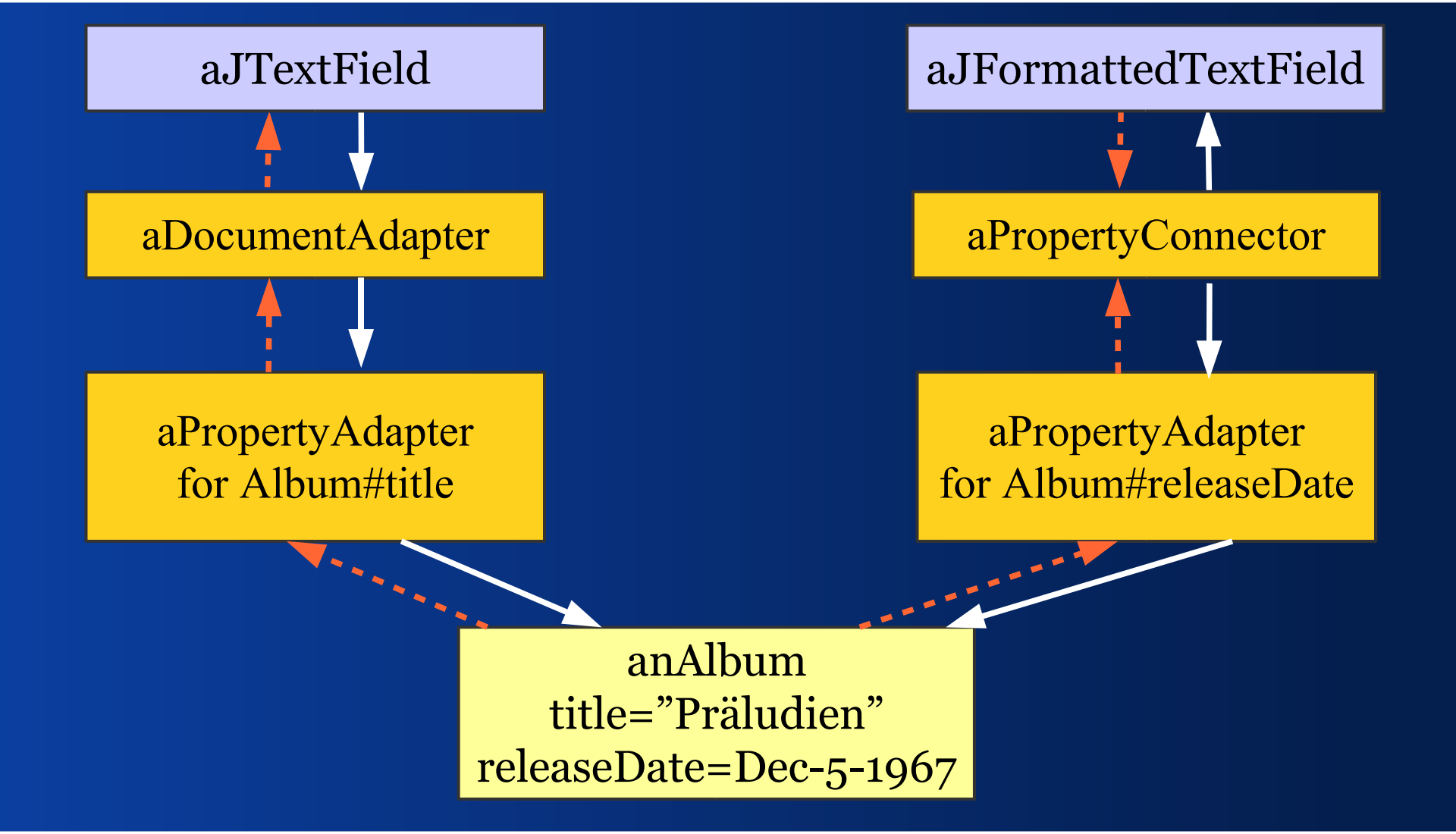
BufferedValueModel



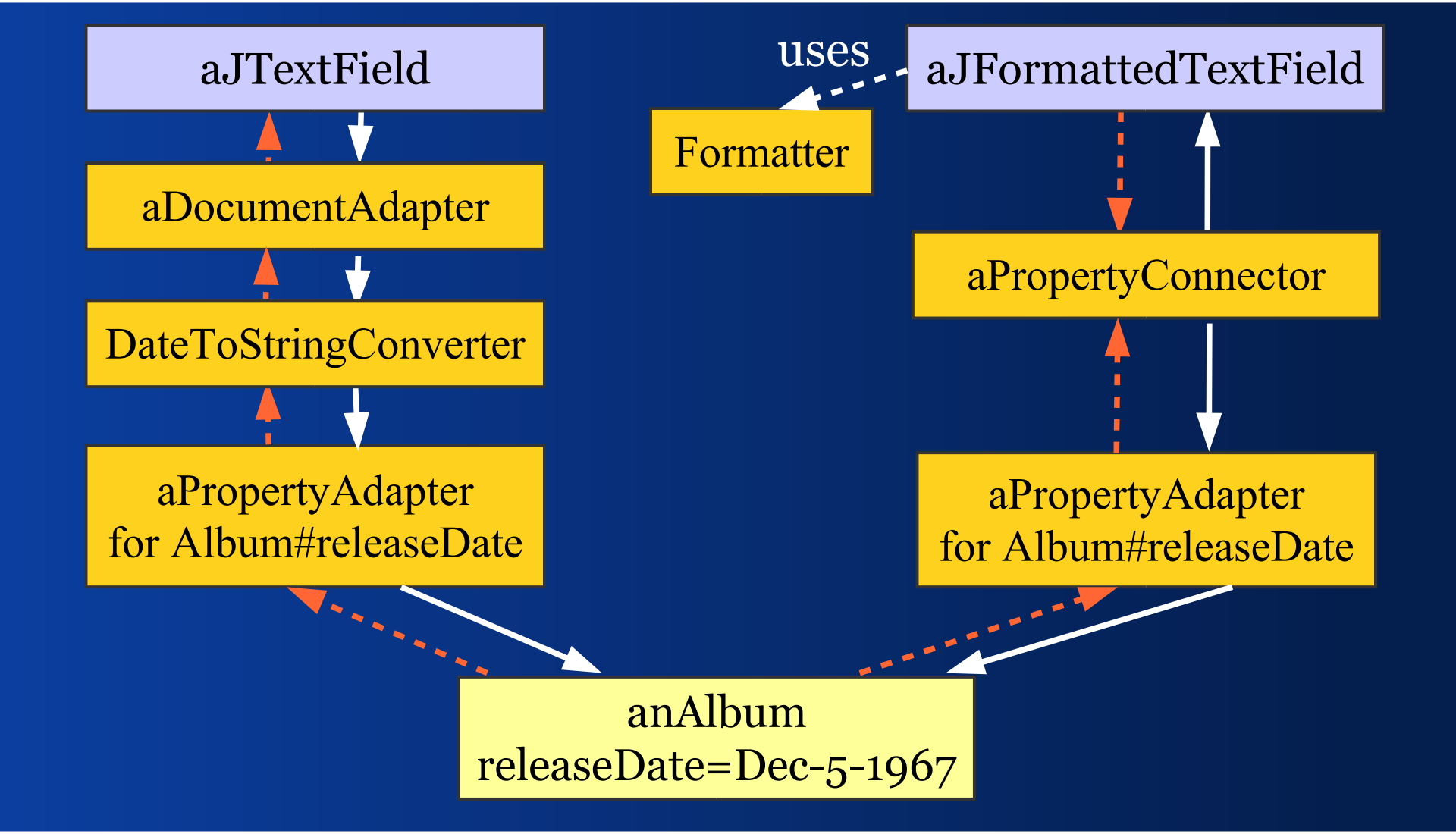
Puffer-Trigger teilen



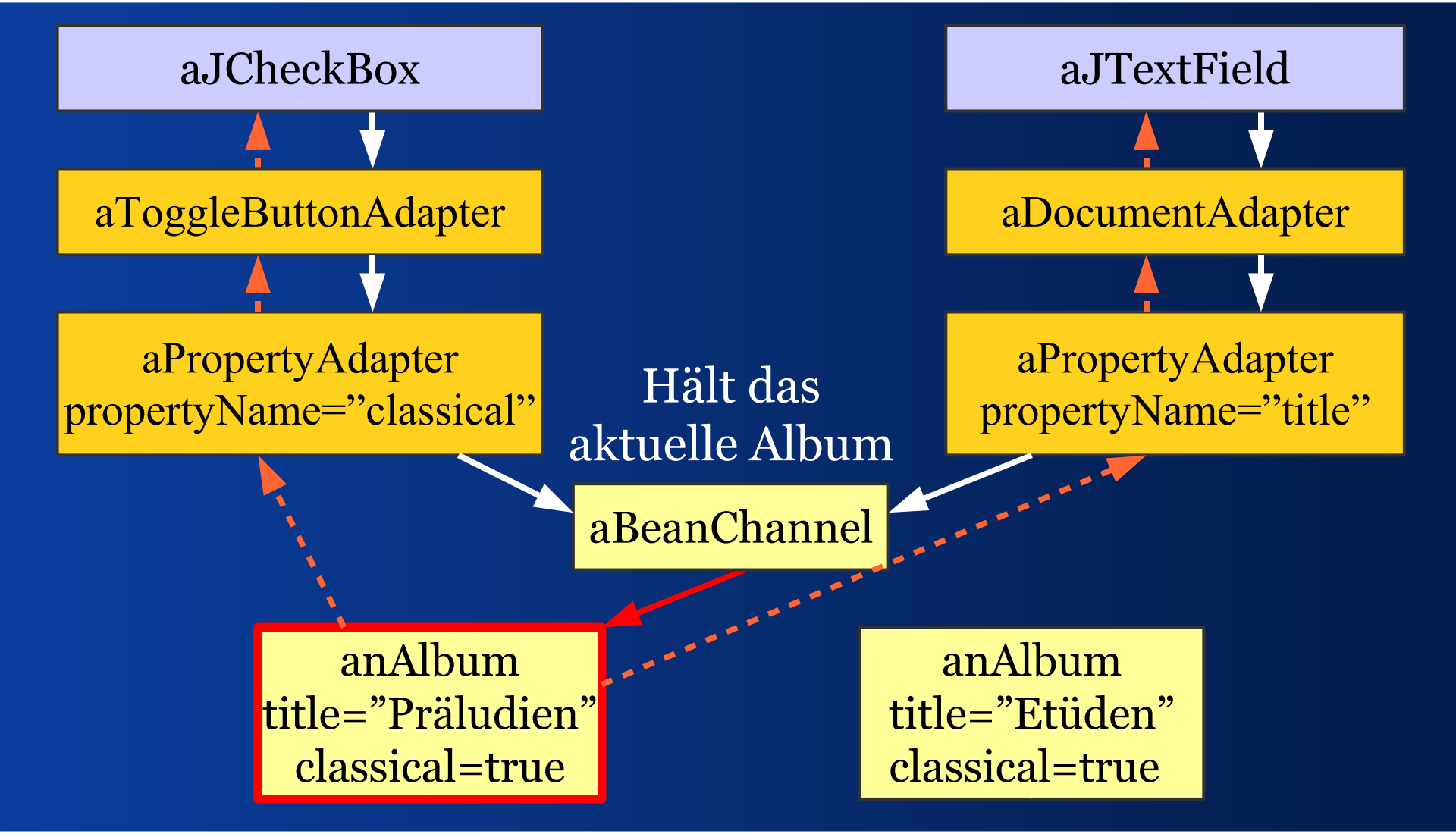
Adapter vs. Connector



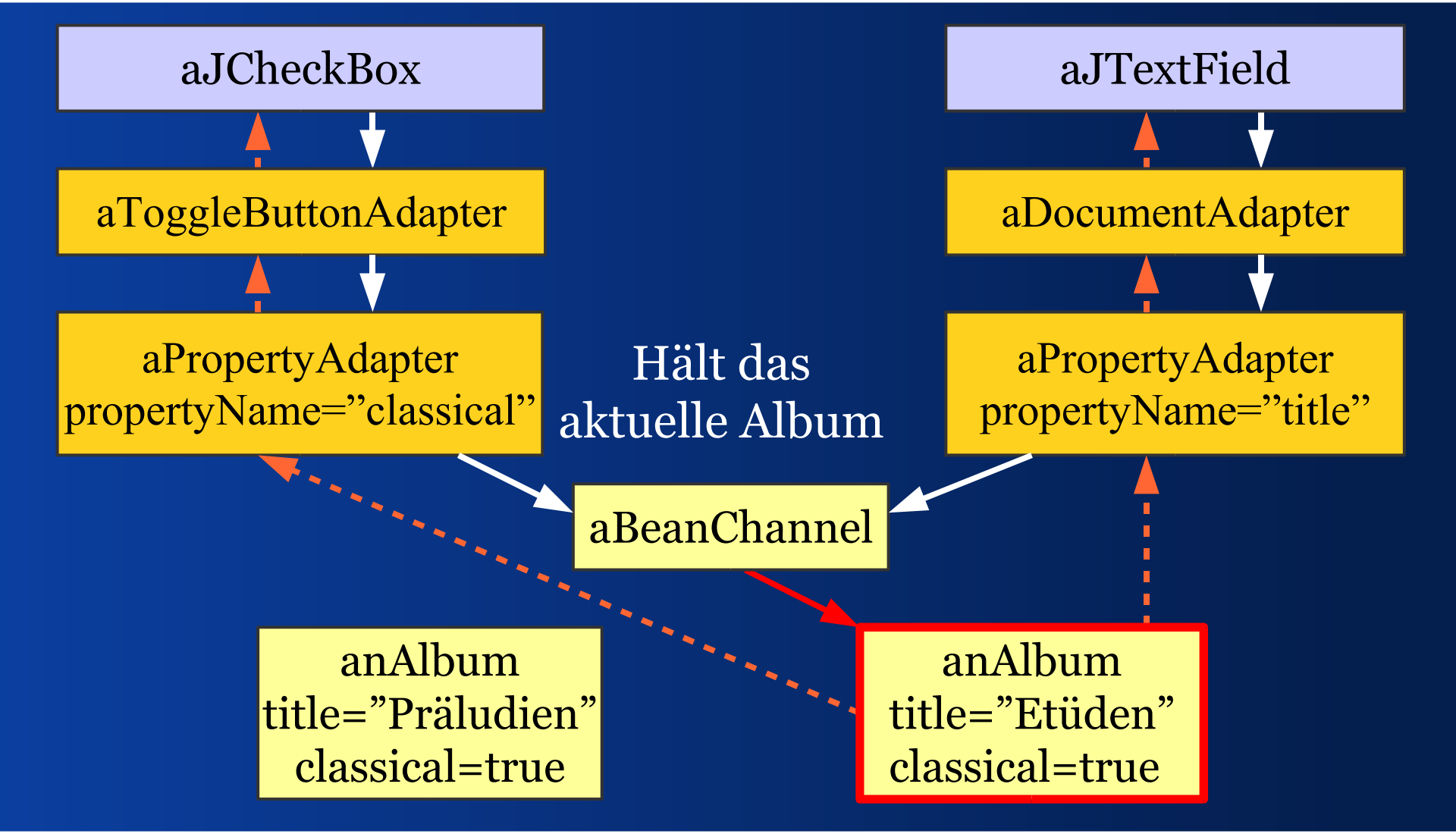
Converter vs. Formatter



Indirektion



Indirektion



III - Listen binden

Wie verbinde ich
Listen von Eigenschaften mit Komponenten?

Problem der Listenbindung

Wir wollen Listeninhalte beobachten,
nicht nur, dass eine Liste komplett anders ist.

Sonst verlieren wir Selektion und Rollzustand.

Ein universelles Listenmodell

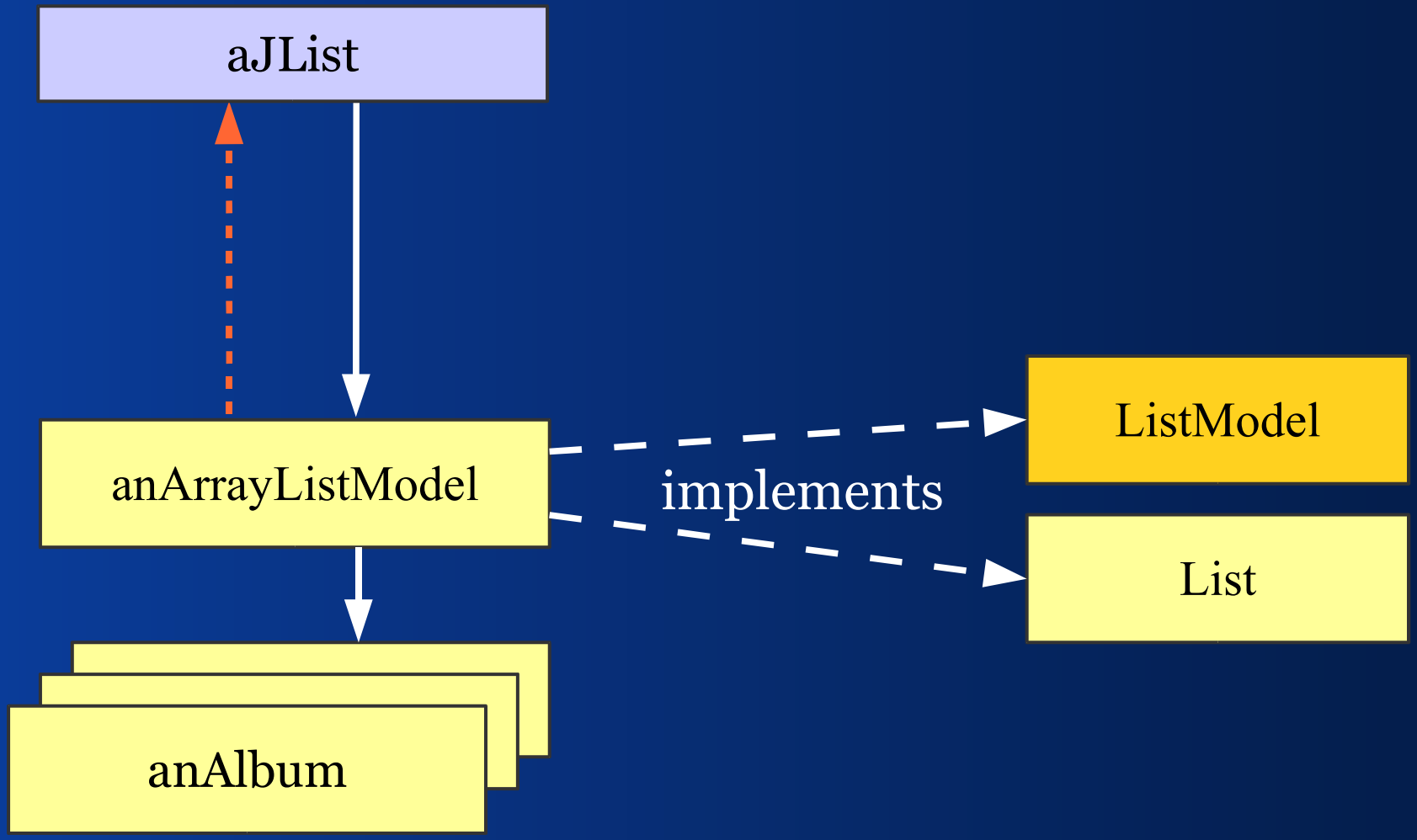
- Wir wollen Elemente lesen.
- Wir wollen die Größe wissen.
- Änderungen bemerken:
  - wenn Elemente geändert werden
  - wenn Elemente zugefügt werden
  - wenn Elemente entfernt werden

Die Swing-Klasse `ListModel` bietet das.

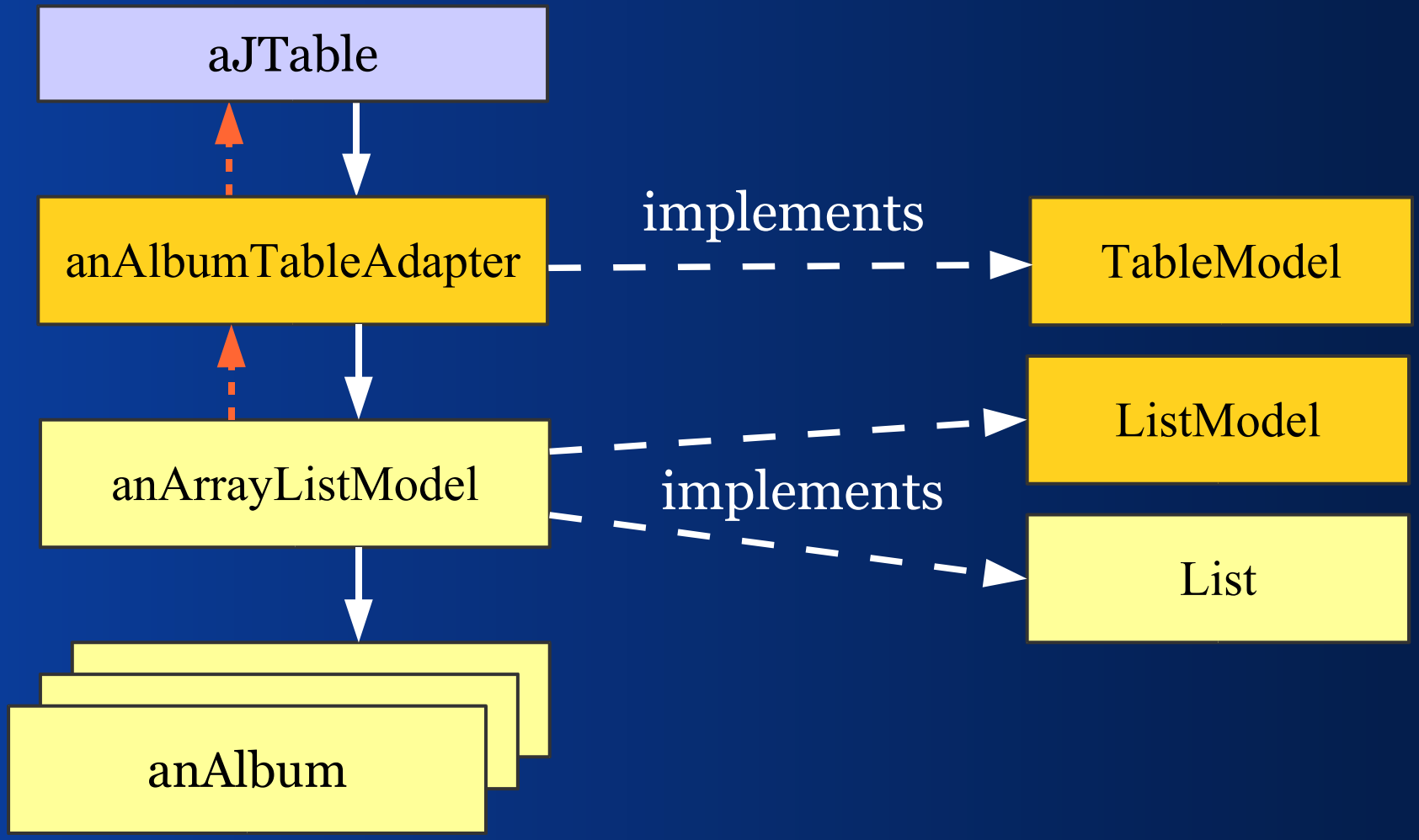
ListModel-Implementierungen

- ArrayListModel erweitert ArrayList, implementiert ListModel
- LinkedListModel erweitert LinkedList, implementiert ListModel
- Man kann wie auf List operieren und bekommt Änderungen gemeldet.

Liste an JList binden



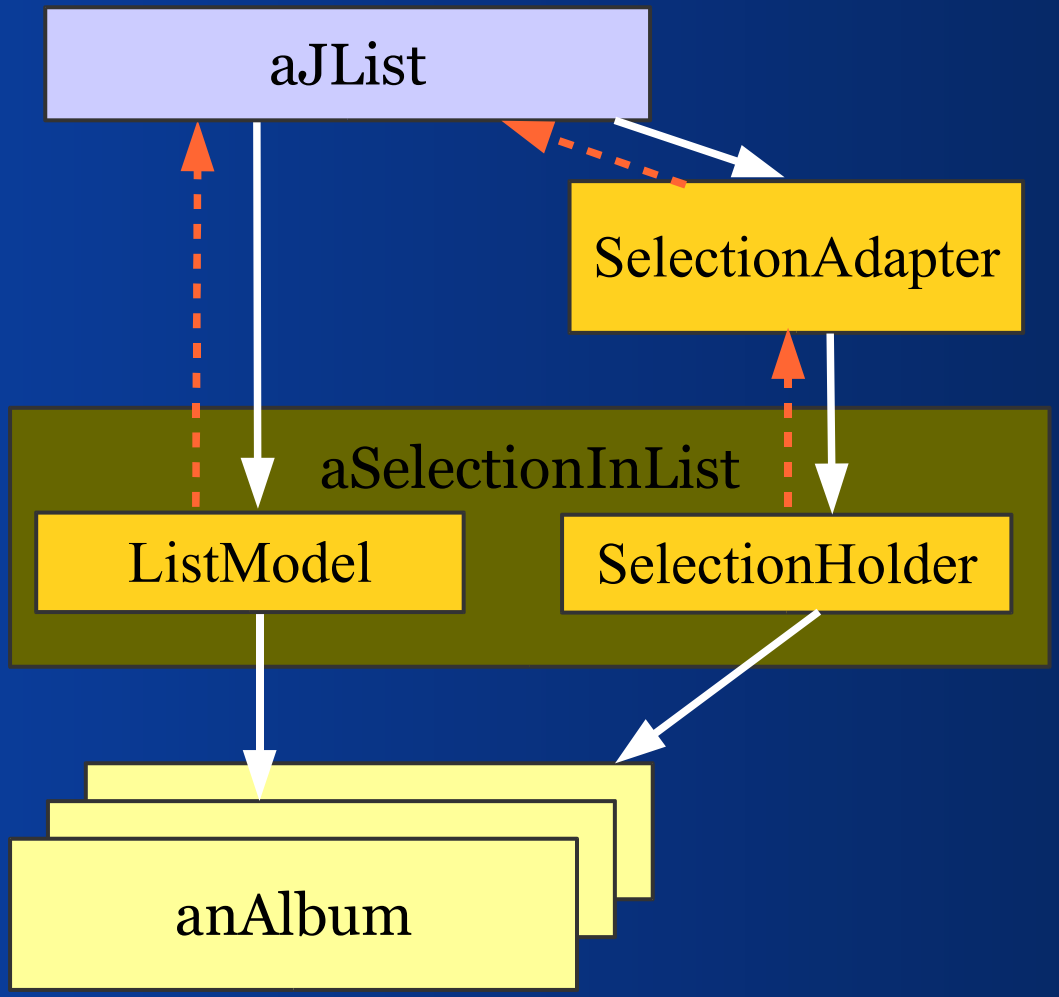
Liste an JTable binden



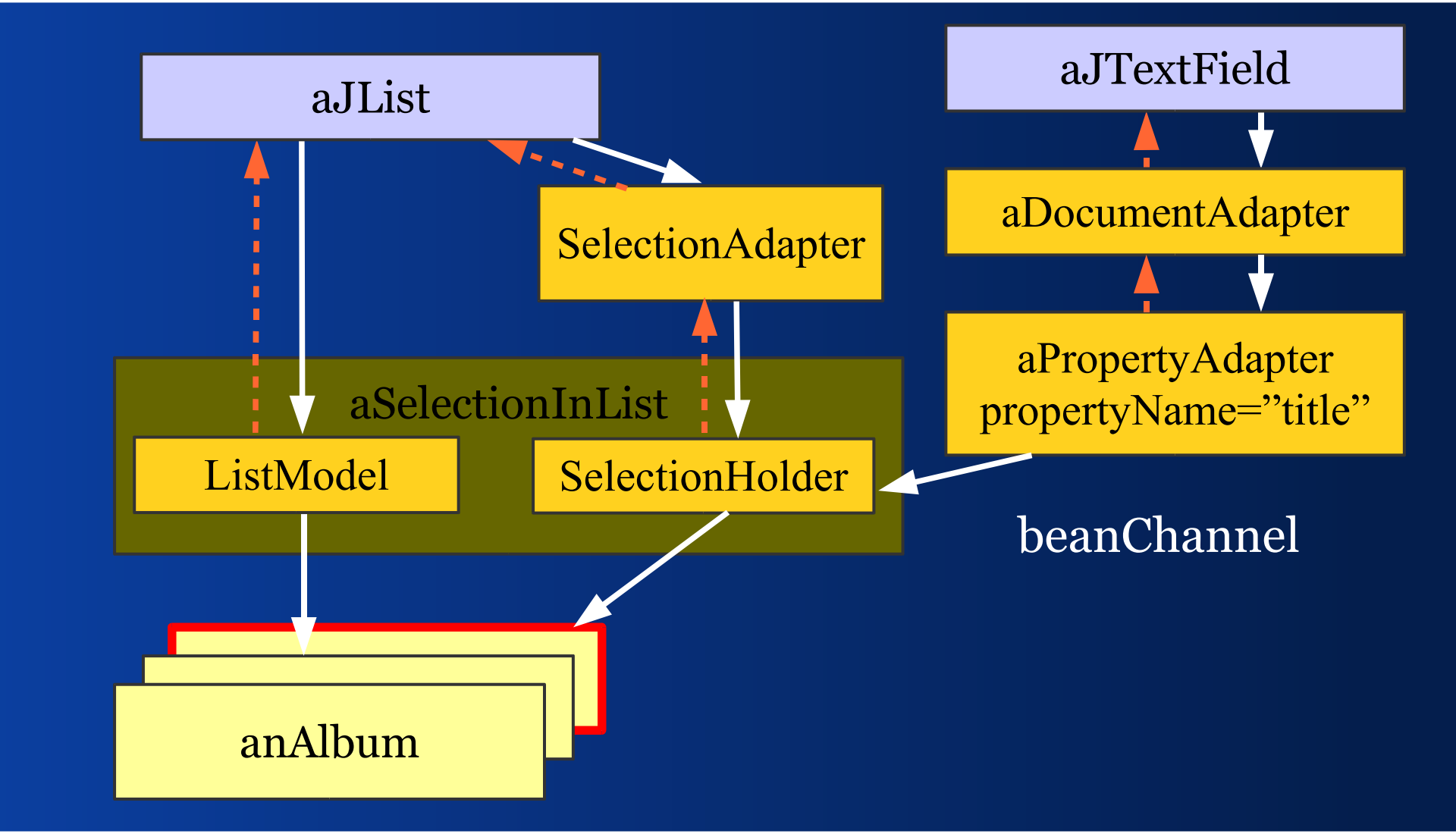
Häufig: Liste mit Einzelselektion

- Dazu bauen wir ein Verbundmodell, das ListModel hält und die Selektion.
- Dieses Modell berichtet Änderungen an:
  - der Selektion
  - dem Selektionsindex
  - den Listeninhalten
  - der Liste

SelectionInList



Überblick / Details



IV - Architektur

Eine 3-Schichten-Swing-Client-Architektur

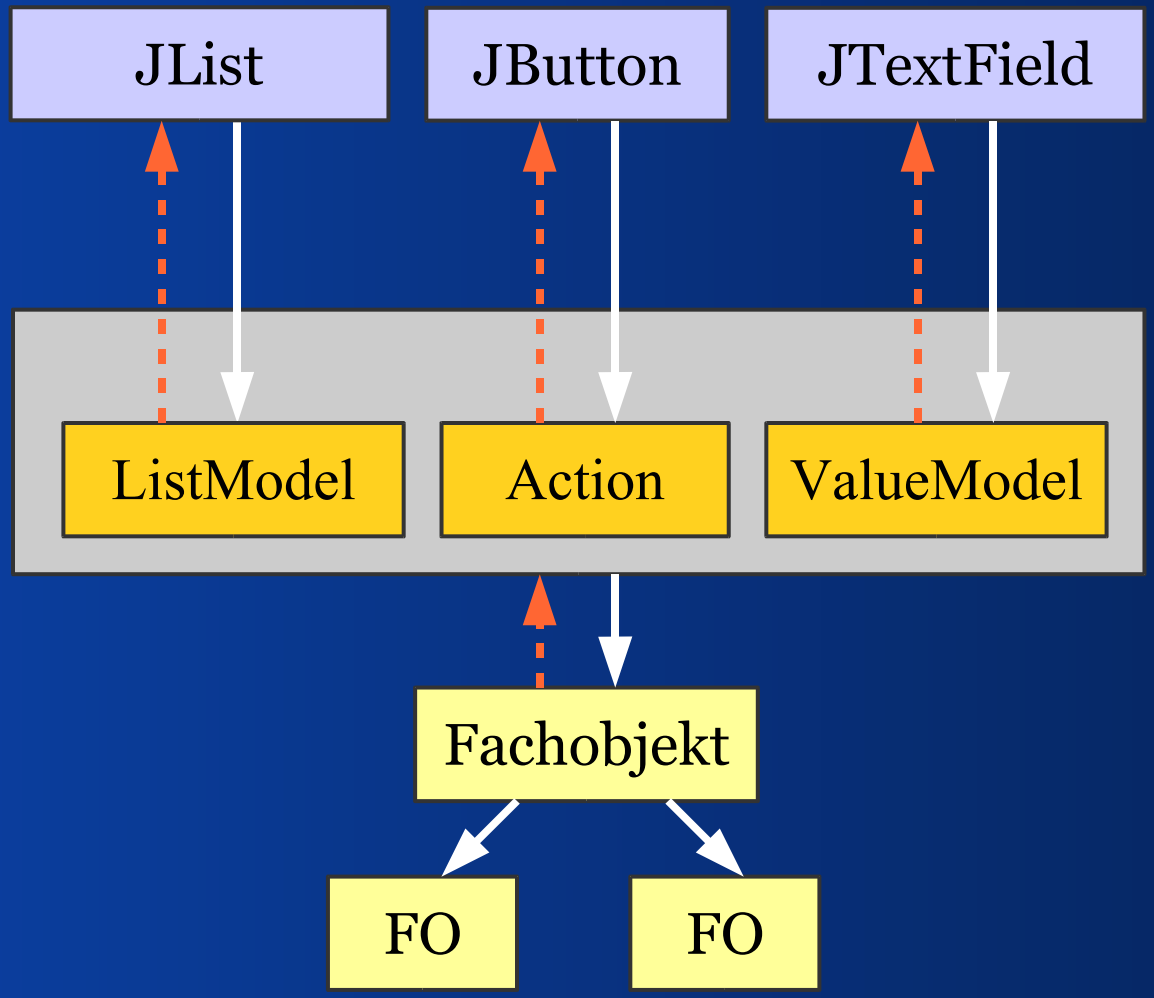
Ziele

- Arbeitet mit Standard-Swing-Komponenten
- Arbeitet mit eigenen Swing-Komponenten

- Braucht keine besonderen Komponenten
- Braucht keine besonderen JPanel

- Passt zusammen mit JGoodies Validation
- Läuft mit unterschiedlichen Prüfstilen

3-Schichten-Client-Architektur



Präsentation

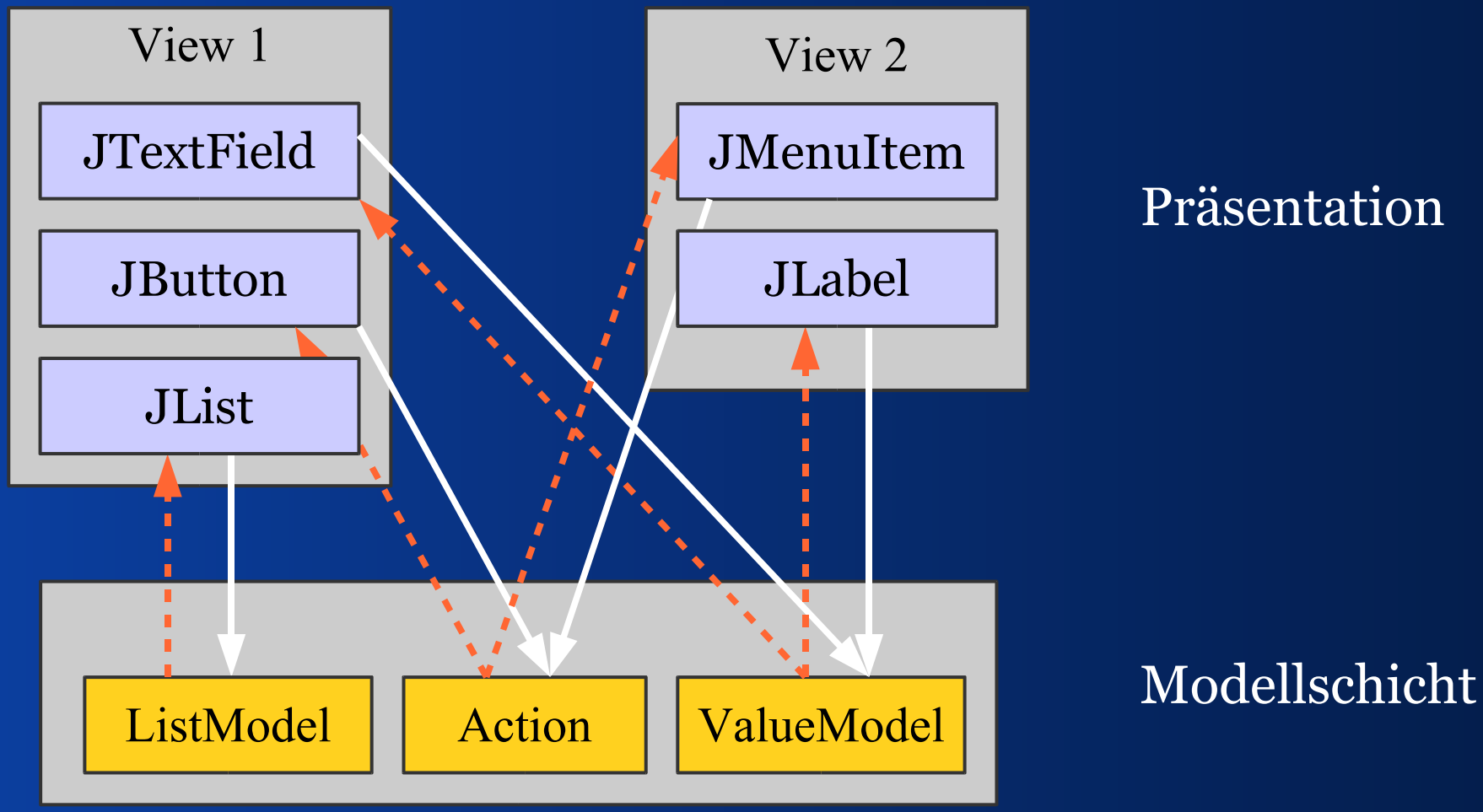
Modellschicht

Fachschicht

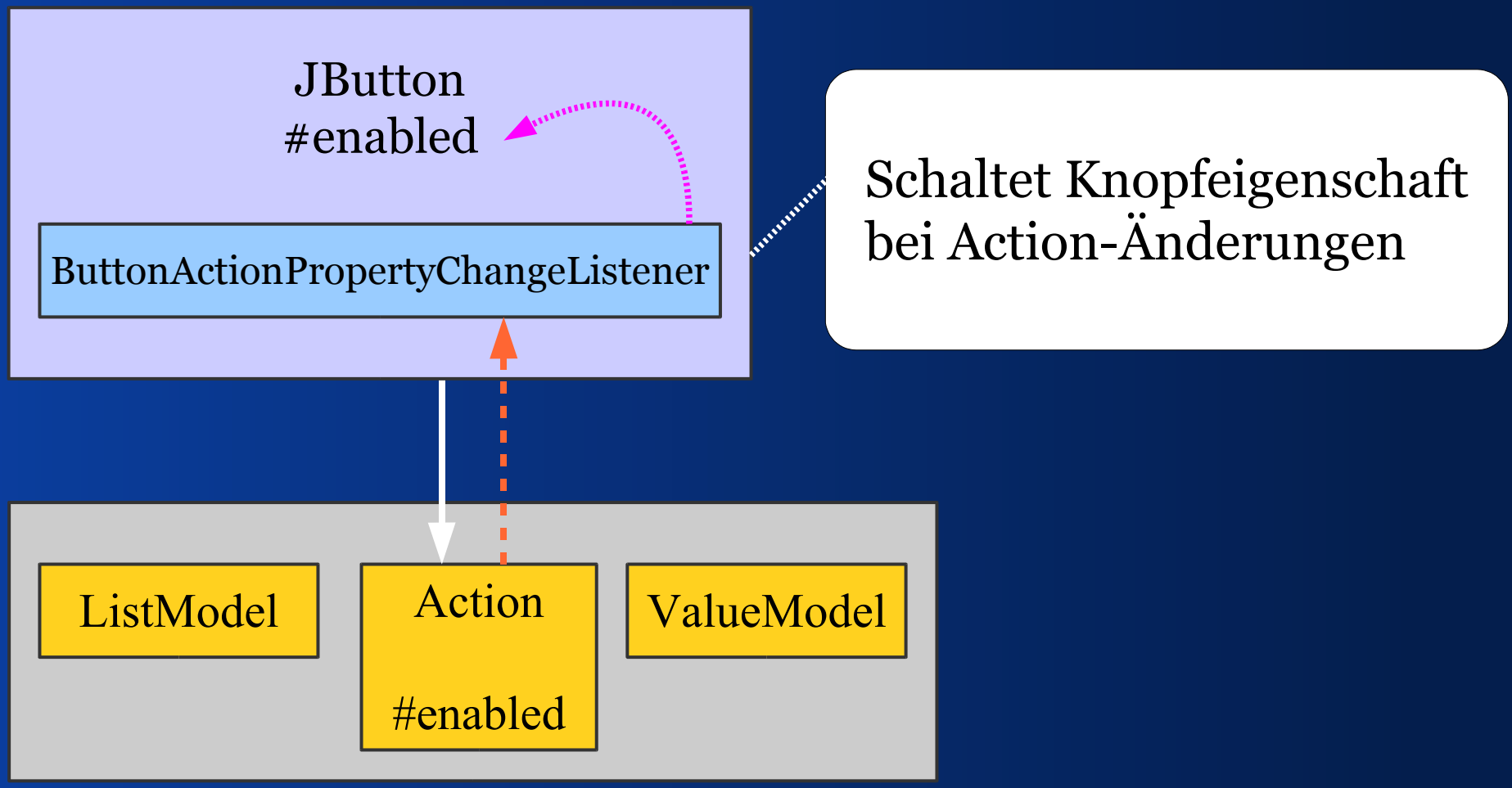
Auswirkungen der 3-Schichten

- Views sind einfach zu bauen (sind dumm)
- Views sind entkoppelt
- Fachschicht ist klar abgetrennt
- Entwickler verstehen, wo was hingehört
- Synchronisation ist einfach
- Komplexität ist gesenkt
- Modelloperationen nur in der Mitte
- “Kuddel-Muddel” ist begrenzt auf die Mitte

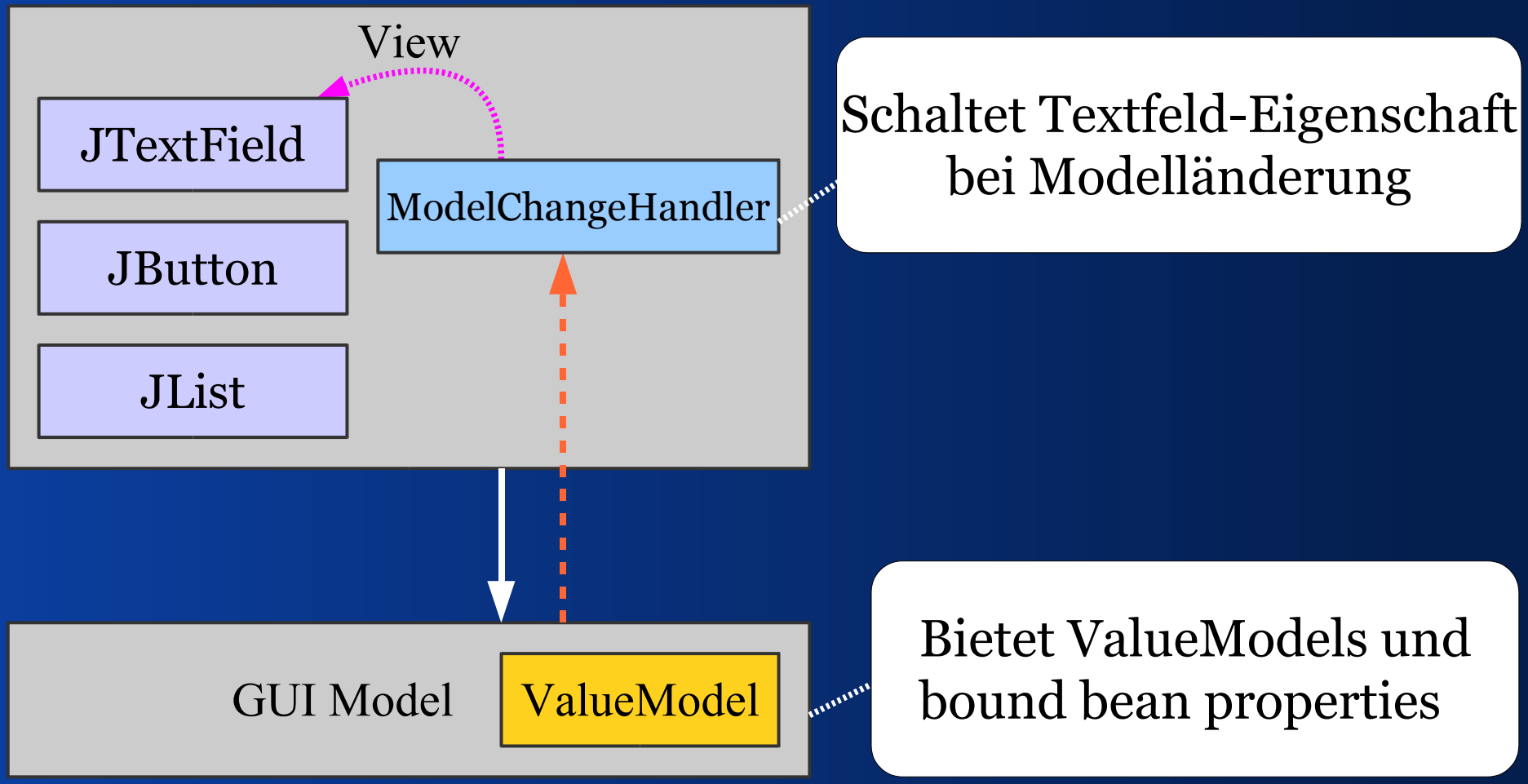
Mehrere Views



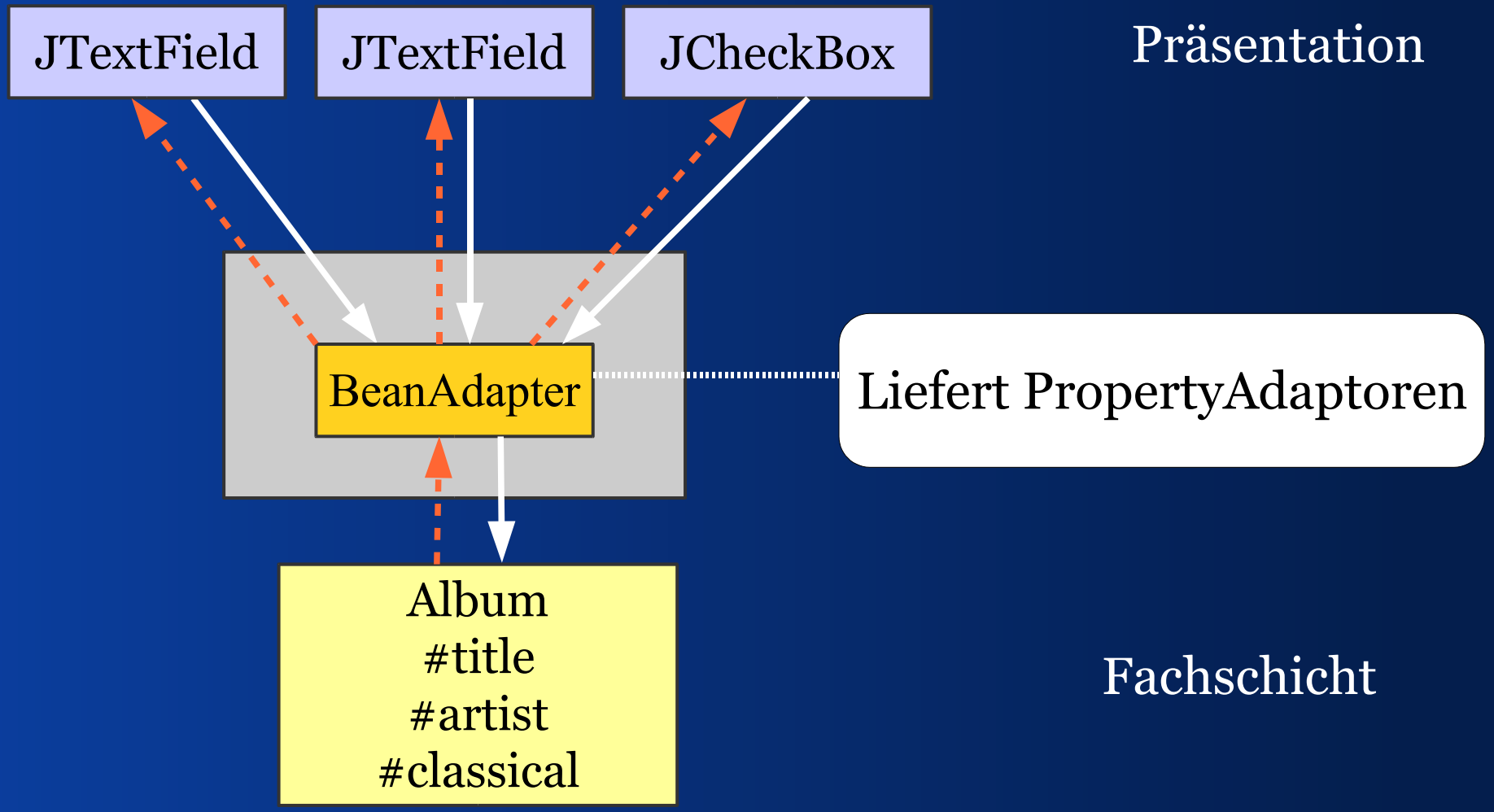
UI-Eigenschaften setzen: Actions



UI-Eigenschaften Setzen



Viele Eigenschaften Adaptieren



View-Quelltextbeispiel

1) Variablen für UI-Komponenten

2) Konstruktoren

3) Erzeuge,binde,konfiguriere UI-Komponenten

4) Registriere GUI-Zustands-Handler im Model

5) Baue und liefere ein Panel

6) Handler, die den GUI-Zustand ändern

Beispiel-View 1/7

```
public final class AlbumView {

    // Referenziert den Modelllieferanten
    private AlbumPresentationModel model;

    // UI-Komponenten
    private JTextField titleField;
    private JCheckBox  classicalBox;
    private JButton    buyNowButton;
    private JList      referencesList;
    ...
```

Beispiel-View 2/7

```
public AlbumView(AlbumPresentationModel m) {

    // Merke den Modelllieferanten
    this.model = m;

    // Konfiguration des Views.
    ...
}
```

Beispiel-View 3/7

```
private void initComponents() {
    titleField = ComponentFactory.createField(
        model.getTitleModel());
    titleField.setEditable(false);

    buyNowButton = new JButton(
        model.getBuyNowAction());

    referenceList = new JList(
        model.getReferenceListModel());
    referenceList.setSelectionModel(
        model.getReferenceSelectionModel());
```

Beispiel-View 4/7

```
private initEventHandling(){
    // Registriere einen GUI-Zustands-Handler
    model.addPropertyChangeListener(
        "composerEnabled",
        new ComposerEnablementHandler());
}
```

Beispiel-View 5/7

```
public JPanel buildPanel() {
    // Erzeuge,binde,konfiguriere Komponenten
    initComponents();

    // Registriere GUI-Zustands-Handler
    initEventHandling();

    FormLayout layout = new FormLayout(
        "right:pref, 3dlu, pref", // 3 columns
        "p, 3dlu, p");            // 3 rows

    ...
```

Beispiel-View 6/7

```
PanelBuilder builder =
    new PanelBuilder(layout);
CellConstraints cc = new CellConstraints();


builder.addLabel("Title",  cc.xy(1, 1));
builder.add(titleField,    cc.xy(3, 1));
builder.add(availableBox,  cc.xy(3, 3));
builder.add(buyNowButton,  cc.xy(3, 5));
builder.add(referenceList, cc.xy(3, 7));

return builder.getPanel();
}
```

Beispiel-View 7/7

```
/* Lauscht auf #composerEnabled,
   schaltet #enabled im composerField.   */
private class ComposerEnablementHandler
    implements PropertyChangeListener {

    public void propertyChange(
        PropertyChangeEvent evt) {

        composerField.setEnabled(
            model.isComposerEnabled());
    }
}
```

Einfacheres Event Handling

```
private initEventHandling(){
     // Synchronisiere Model- mit GUI-Zustand
     PropertyConnector.connect(
        model,          “composerEnabled”,
        composerField, “enabled”);
}
```

V - Erfahrungsbericht

Wie funktioniert dieser Bindestil im Alltag?

Kosten

- Adapterbinding:
  - Erhöht die Lernkosten
  - Senkt die Produktionskosten geringfügig
  - Kann Änderungskosten stark senken

Verbinde mittels Fabrik.

- Kapsel den Aufbau der Verbindung vom ValueModel zur Swing-Komponente.
- Einige Swing-Komponenten haben kein geeignetes Modell z. B. JFormattedTextField
- Liefere z. B. Komponenten zu ValueModels

Puffern

- Nutze BufferedValueModel sparsam
  - behindert Prüfungen auf Modellen
  - verhindert, dass Fachlogik genutzt werden kann
- Die Client-Fachschicht kann puffern wenn:
  - Fachobjekte Kopien sind
  - Fachobjekte ungültige Werte akzeptieren

Performanz

- In den Adapterketten werden viele PropertyChangeEvents gefeuert.
- Das ist nicht als Problem zu merken.

- ListModel spart, Listeninhalte zu kopieren.

Debugging

- Der Kopieransatz ist einfach zu debuggen; man erkennt gut, wann wo was getan wird.
- Was die Bindeklassen an Arbeit abnehmen lasten sie einem beim Debuggen auf.
- Reflection und Introspection erschweren das Verständnis, wer Werte liest und setzt.
- Darum gib allen Listenern einen Namen.

Umbenennen

- Reflection und Introspection erschweren das Umbenennen von Eigenschaften und deren Gettern/Settern.
- Verwende Konstanten für die Eigenschaftsnamen.
- Obfuscator verliert Zusammenhänge.

Für wen passt Adapterbindung?

- Ich schätze, die Adapterbindung passt in etwa 80% aller Fälle.

- Allerdings braucht man einen Experten im Team, der die Bindeklassen beherrscht; es reicht nicht, sie zu kennen.

Vorteile der Adapterbindung

- Kann viel Code sparen.
- Code wird leichter zu lesen.
- Schichten sind leichter zu trennen.
- Komplexität kann wesentlich sinken.

Ganz oder gar nicht

- Die 3-Schichtenarchitektur lohnt, wenn die Anwendung überwiegend danach aufgebaut ist.

- Refaktoriert man Teile zu diesem Muster, sind die Umbaukosten hoch und der Nutzen zweifelhaft.

Wo steht JGoodies Binding?

- Ansatz ist 10 Jahre alt und stabil.
- Architektur des Java-Ports ist stabil.
- Tests decken etwa 90% der Klassen ab.
- Wenig Dokumentation.
- Tutorial ist noch recht klein.

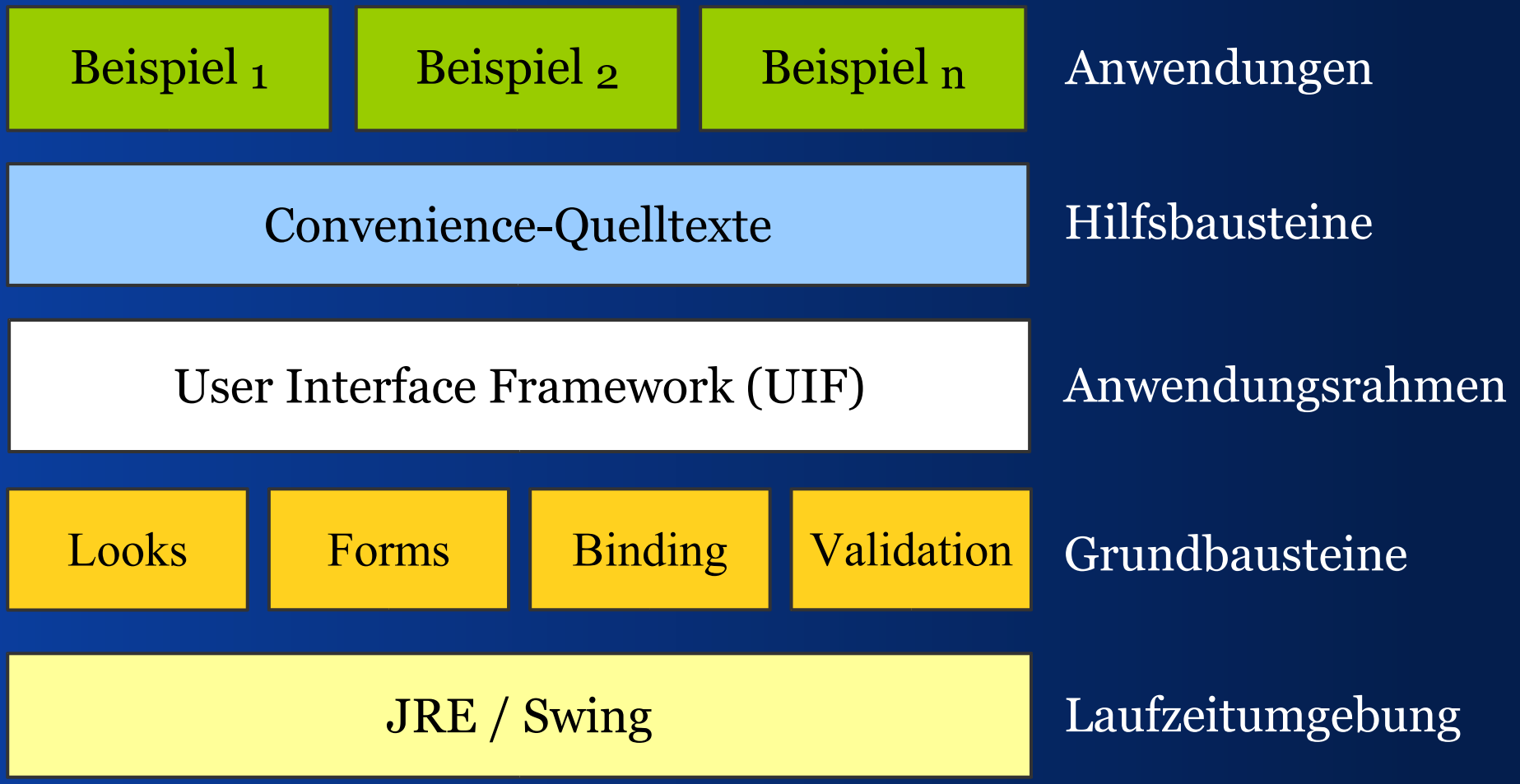
Schluss

Zusammenfassung und Referenzen

Zusammenfassung

Wir haben MVC in Swing gesucht,
Bindeaufgaben identifiziert,
ValueModel motiviert,
gesehen, wie man Werte binden kann,
gelernt, wie man Listen binden kann.

JGoodies Swing Suite



Referenzen I

- JGoodies Binding
  binding.dev.java.net
- JGoodies-Artikel
  www.JGoodies.com/articles/
- JGoodies-Demos
  www.JGoodies.com/freeware/
- Fowlers Enterprise Patterns
  martinfowler.com/eaaDev/

Referenzen II

- Oracles JClient und ADF
  otn.oracle.com/, nach 'JClient' suchen
- Spring Rich Client Project
  www.springframework.org/spring-rcp.html
- Suns JDNC
  jdnc.dev.java.net
- Understanding and Using ValueModels
  c2.com/ppr/vmodels.html

Demo/Tutorial:

JGoodies Binding Tutorial

Aufgaben und Lösungen zum Binden

(in Arbeit)

Ships with the JGoodies Binding

Fragen und Anworten

Ende

Hoffentlich hilft‘s!

Viel Erfolg!