Softwaretechnik

M. Jakob

Gymnasium Pegnitz

15. März 2016

Inhaltsverzeichnis

1. Grundlagen der Projektorganisation

2. Entwurfsmuster
   - Allgemeines
   - Adaptermuster
   - Kompositum
   - Beobachtermuster
   - model-view-controller-Muster

3. Praktische Softwareentwicklung
   - Profi GUI für Graphen
   - Simulation eines Rangierbahnhofes

Gründe der Projektorganisation

Große Projekte müssen perfekt organisiert sein. Gründe:

- Die Anforderungen des Auftraggebers müssen erfüllt werden,
- die Ressourcen (Geld, Zeit, Material . . .) müssen eingeteilt werden,
- es sind viele Personen beteiligt, die nicht ständig zur Verfügung stehen,
- das Projekt dauert Jahre und muss gewartet werden.

Strategien der Projektorganisation

Um komplexe Projekte durchführen zu können, benötigt man eine passende Strategie. Man unterscheidet je nachdem ob die Anforderungen bekannt und stabil sind oder nicht statische und dynamsiche Strategien.

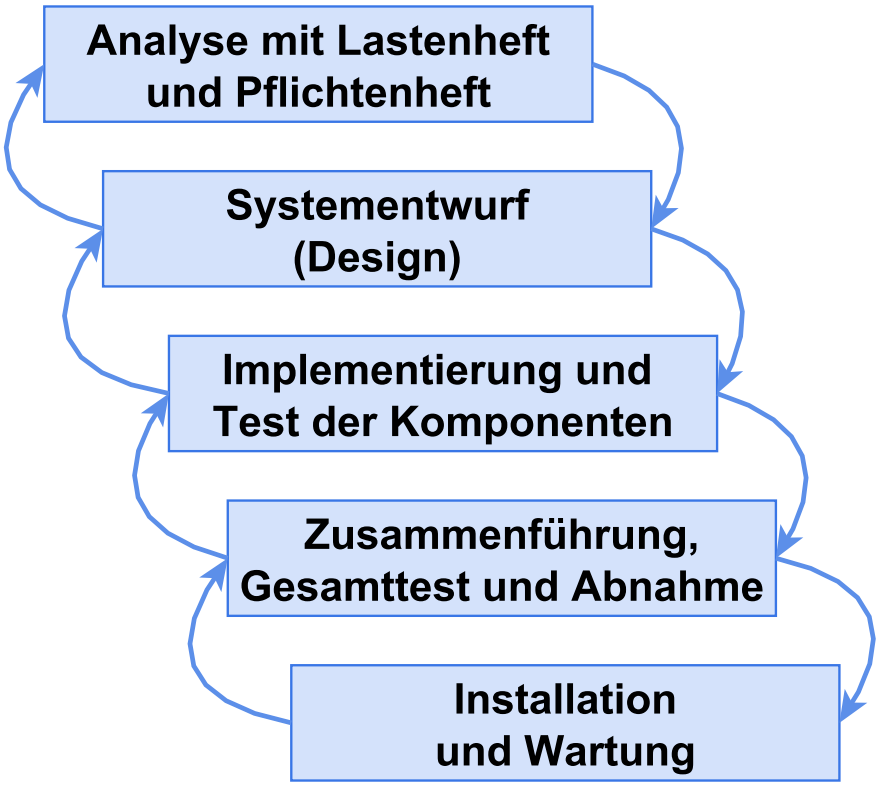
Beispiel: Statische Strategie

- Der Auftraggeber stellt die Anforderungen in einem Lastenheft dar.
- Der Auftragnehmer beschreibt im Pflichtenheft, wie er die Anforderungen erfüllen will.
- Akzeptiert der Auftragnehmer das Pflichtenheft wir das Projekt umgesetzt, und zwar
  - nach einem standardisierten Ablauf (z.B. Wasserfallmodell)
  - durch Zerteilen des Projektes in Teilprojekte mit klarer Schnittstellendefinition,
  - durch Festlegung von Meilensteinen,

Beispiel: Statische Strategie — Das Wasserfallmodell

Das Wasserfallmodell

hilft bei der Umsetzung größerer Projekte. Die Produktentwicklung wird in mehrere Phasen unterteilt. Jede Phase muss abgeschlossen sein, bevor die nächste begonnen wird. Der Projektleiter koordiniert den gesamten Arbeitsablauf. Teamsprecher organisieren die Arbeit in den einzelnen Gruppen.



Zentrale Nachteile statischer Strategien

- Das Ergebnis entspricht möglicherweise nicht den tatsächlichen Erfordernissen weil im Lastenheft Anforderungen übersehen wurden, oder sich die Anforderungen geändert haben.
- Möglicherweise unnötiger und sehr großer Planungsaufwand.

Dynamische Strategien

Sie begegnen dem Nachteil der statischen Strategien, indem sie versuchen, die reine Entwurfsphase auf ein Mindestmaß zu reduzieren und im Entwicklungsprozess so früh wie möglich zu ausführbarer Software zu gelangen, die dann in regelmäßigen, kurzen Abständen dem Kunden zur gemeinsamen Abstimmung vorgelegt werden kann. Auf diese Weise soll es jederzeit möglich sein, flexibel auf Kundenwünsche einzugehen, um so die Kundenzufriedenheit insgesamt zu erhöhen.

Zentraler Nachteil dynamsicher Strategien

Ein planvolles Handeln findet kaumt statt, es werden nur halbwegs taugliche Lösungen geliefert (Gefahr der „Flickschusterei“)

Übungen

Ü 1.1: Projektstrategien im Alltag

Nachfolgende sind mehrere Projekte des Alltags angegeben. Strukturiere jedes der Projekte mit einer statischen und eine dynamischen Strategie, erläutere die Vor- und Nachteile und beurteile, welche Strategie geeigneter ist.

1. Herstellen einer Sitzgelegenheit für 10 Personen
2. Bestehen des Schuljahres
3. Sicherung der familiären Ernährungslage für einen Monat

In diesem Abschnitt

2. Entwurfsmuster
   - Allgemeines
   - Adaptermuster
   - Kompositum
   - Beobachtermuster
   - model-view-controller-Muster

Entwurfsmuster

Alltagsbeispiele

- Essens-Schema: Vorspeise - Hauptspeise - Nachtisch
- Witze-Schema: „zuerst-dann-als letztes“-Aufbau
- Werbe-Schema: Aufmerksamkeit erwecken - Bedürfnis schaffen - Lösung anbieten
- Kompositions-Schema: Tonika - Subdominante - Dominante
- Schema-Schema: 1-2-3-Schema

Entwurfsmuster

helfen bei der Strukturierung der Klassen in der Entwurfsphase, weil sie gut durchdachte Lösungsschemata für immer wieder vorkommende Standardsituationen anbieten. Dadurch wird die Übersicht verbessert und Entwicklungsfehler vermieden.

In diesem Abschnitt

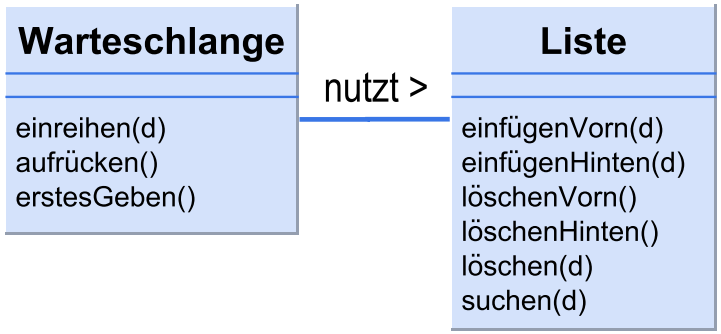
2. Entwurfsmuster
   - Allgemeines
   - Adaptermuster
   - Kompositum
   - Beobachtermuster
   - model-view-controller-Muster

Adaptermuster

Alltagsbeispiele: Adapterstecker, Geld

Adaptermuster

. . . nutzen und ergänzen vorhandene Klassen und bieten der darüber liegenden Ebene eine einheitliche Schnittstelle. So kann flexibel auf spätere Erweiterungen und Korrekturen reagiert werden.



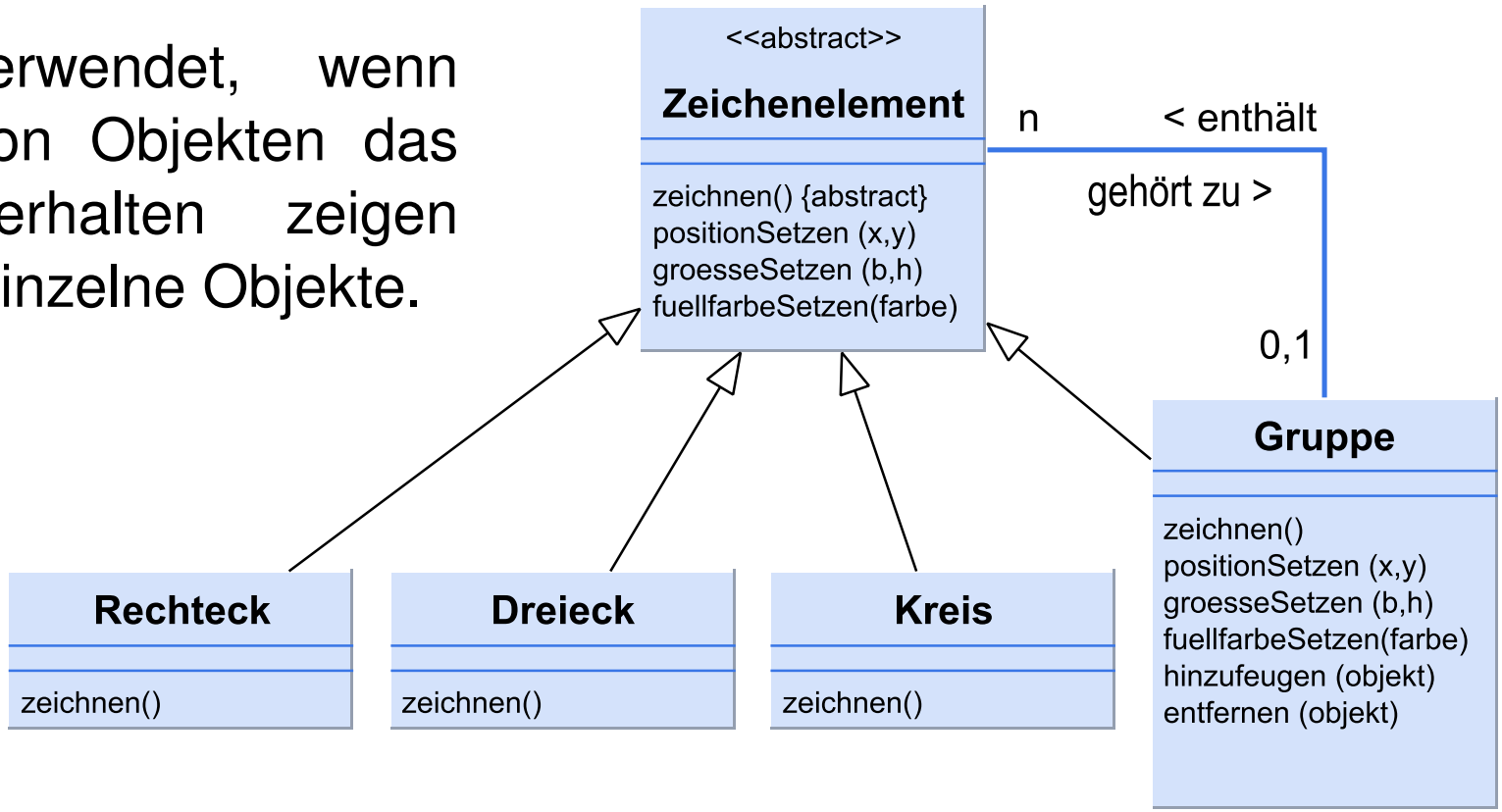
In diesem Abschnitt

2 Entwurfsmuster
- Allgemeines
- Adaptermuster
- Kompositum
- Beobachtermuster
- model-view-controller-Muster

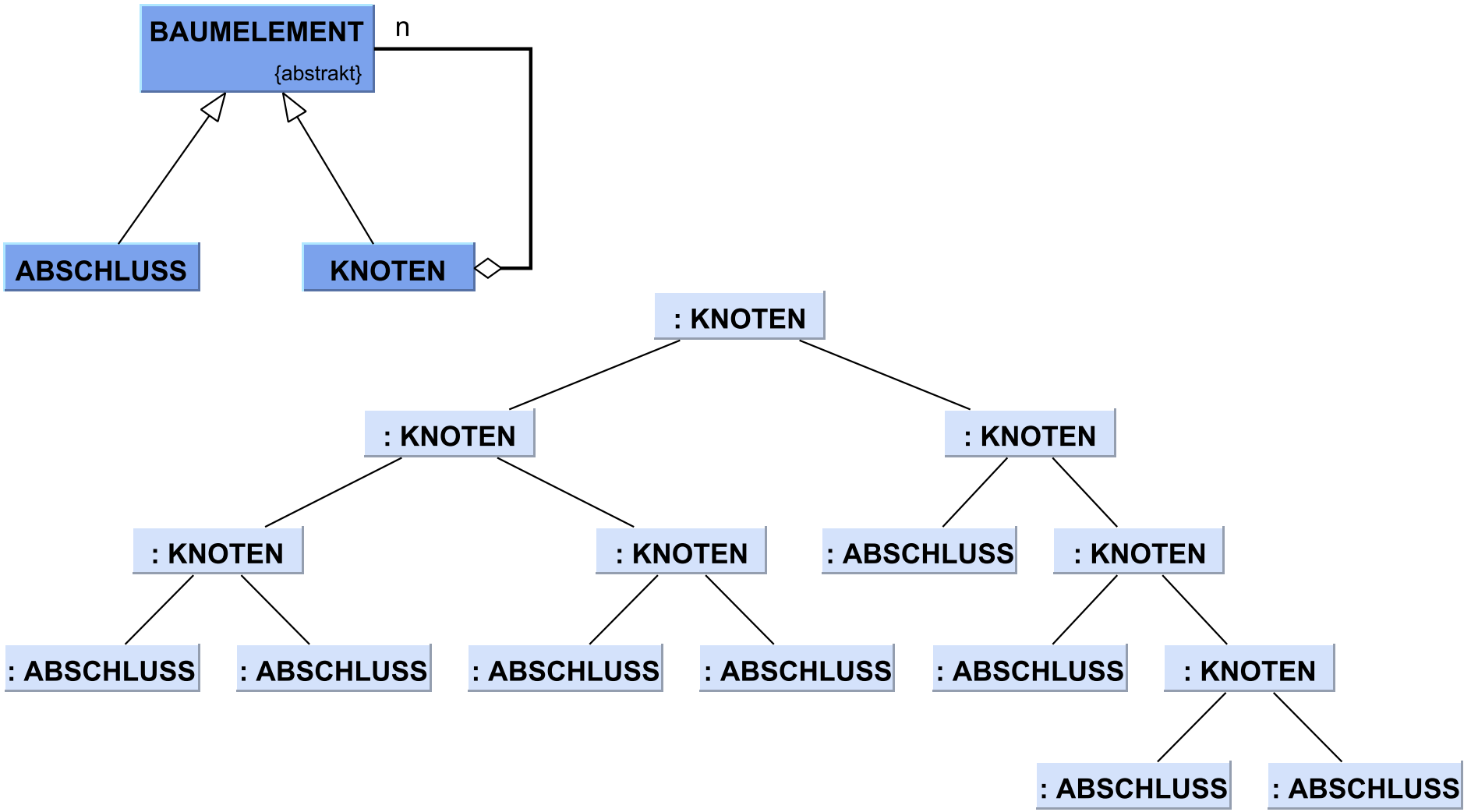
Kompositum

Das Kompositum

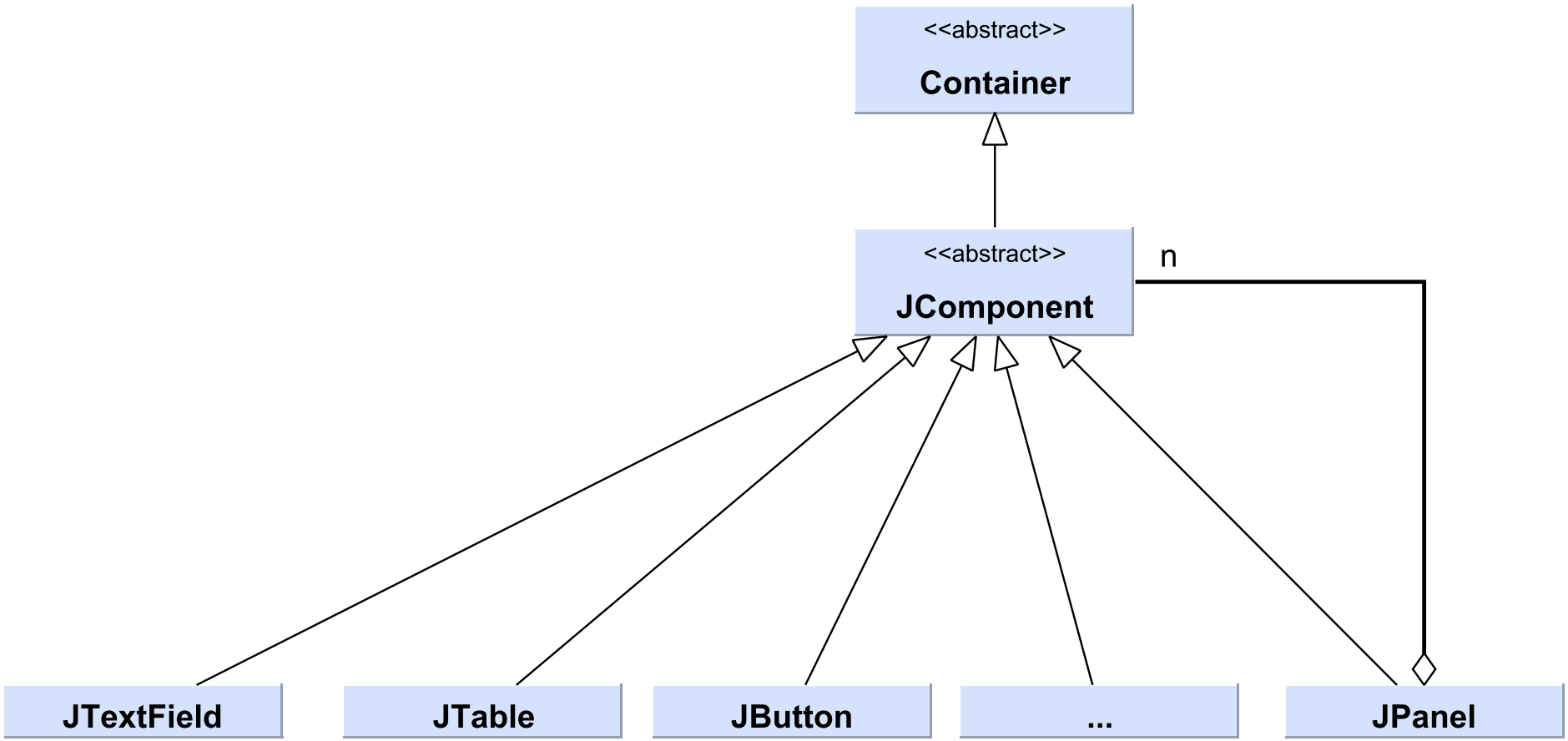
. . . wird verwendet, wenn Gruppen von Objekten das gleiche Verhalten zeigen sollen wie einzelne Objekte.



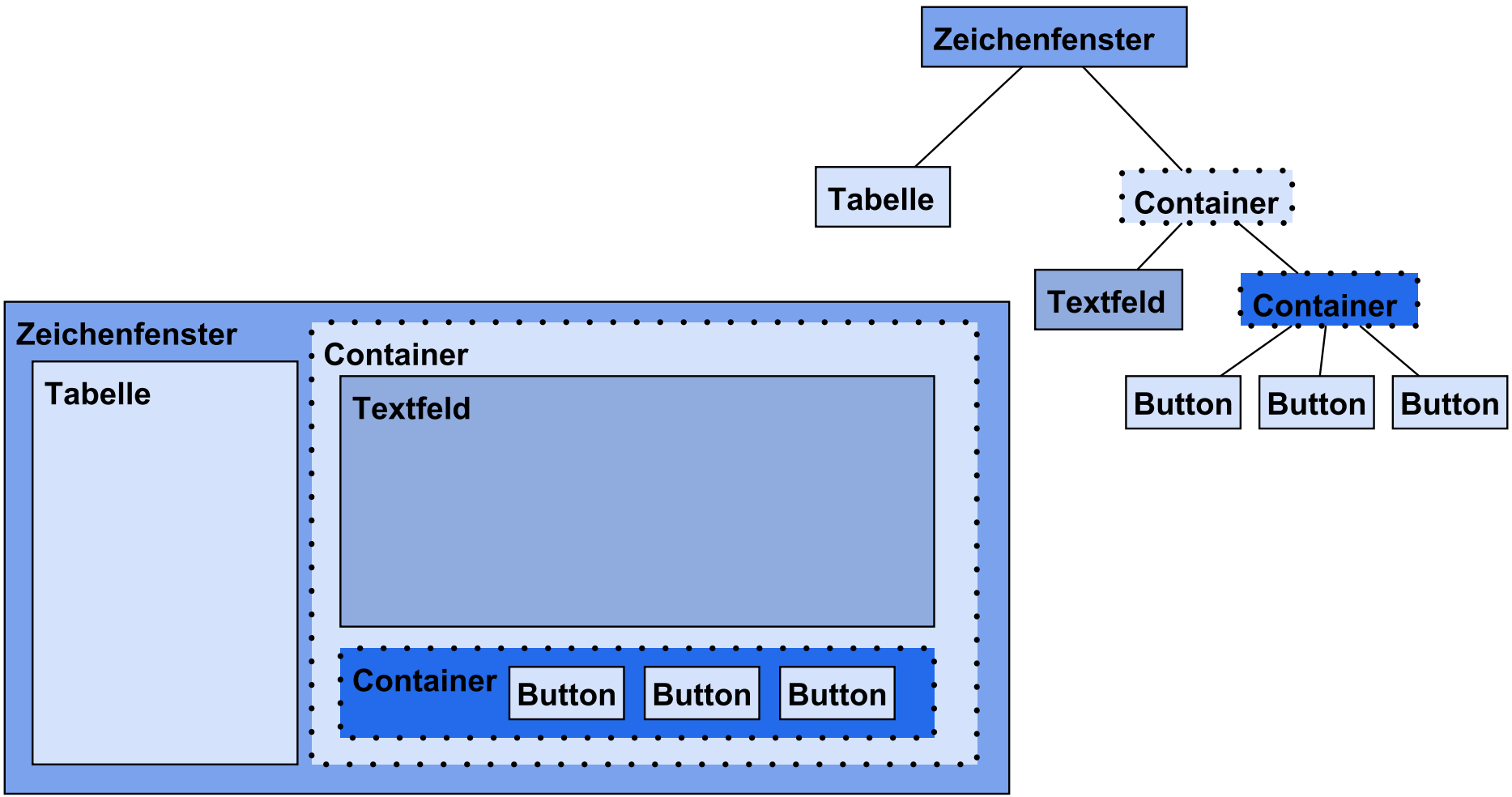
Beispiel 1 — Baumstruktur als Kompositum



Beispiel 2 — Grafische Oberfläche bei Java Swing



Beispiel 2 — Grafische Oberfläche bei Java Swing



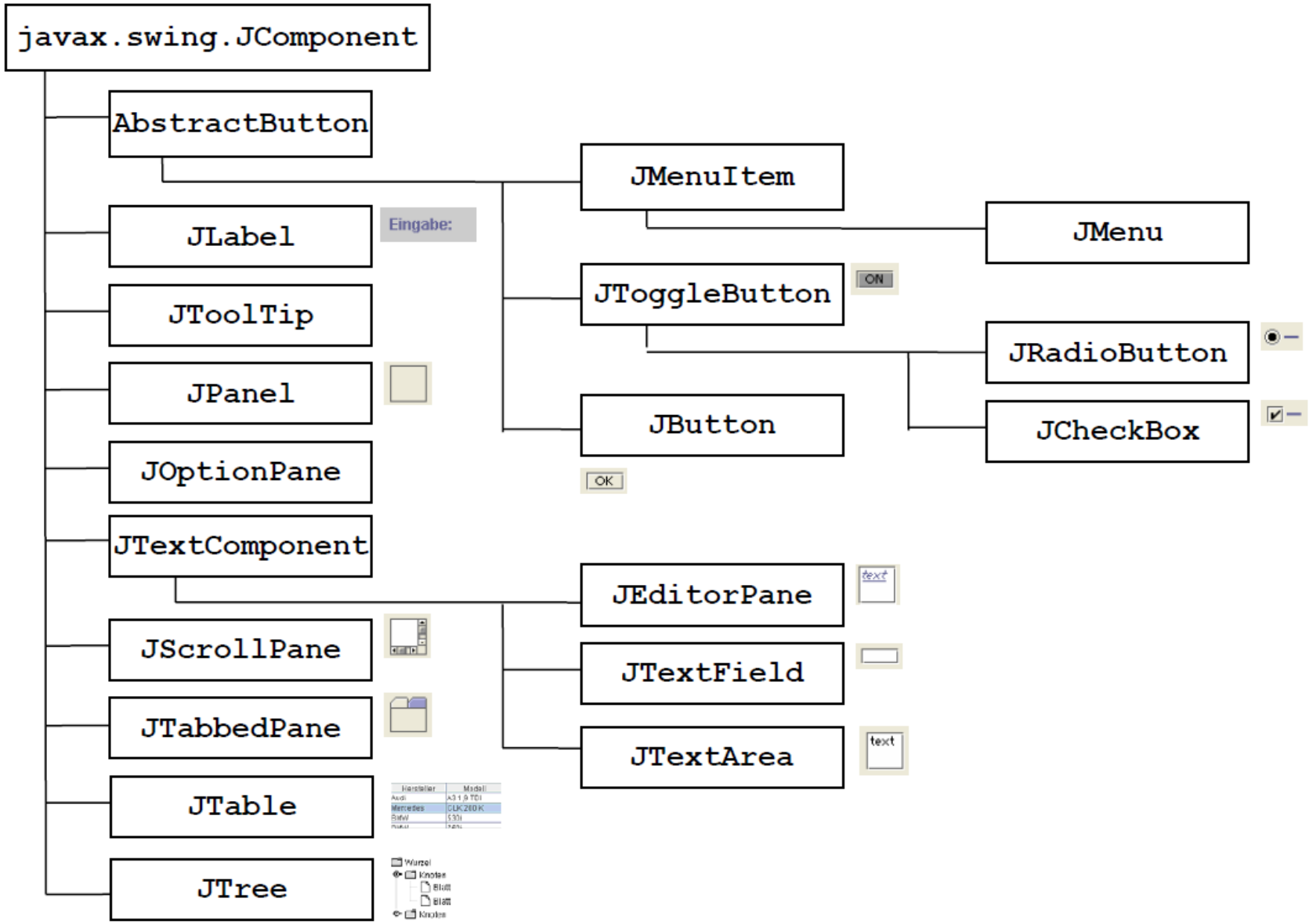
Beispiel 2 — Implementation

```
1  // Blatt-Objekte erzeugen
2  JTable tabelle = new JTable();
3  JTextField textFeld = new JTextField("Textfeld");
4  JButton button1 = new JButton("Knopf 1");
5  JButton button2 = new JButton("Knopf 2");
6  JButton button3 = new JButton("Knopf 3");
7
8  // Kompositum-Objekte erzeugen
9  JPanel containerRechts = new JPanel();
10 JPanel containerUnten = new JPanel();
```

Beispiel 2 — Implementation

```
1  // Struktur aufbauen
2  containerRechts.add ( textFeld );
3  containerRechts.add ( containerUnten );
4  conatinerUnten.add ( button1 );
5  conatinerUnten.add ( button2 );
6  conatinerUnten.add ( button3 );
7
8  // in das Fenster einhängen
9  JFrame zeichenfenster = new JFrame ( "
       Zeichenfenster" );
10 frame.add ( tabelle );
11 frame.add ( containerRechts );
```

Übersicht Swing-Komponenten



Übung

- Ü 2.1: Java.swing inspizieren
- Ü 2.2: ProfiGUI_1
- Ü 2.3: ProfiGUI_2

In diesem Abschnitt

2. Entwurfsmuster
   - Allgemeines
   - Adaptermuster
   - Kompositum
   - Beobachtermuster
   - model-view-controller-Muster

Beobachtermuster

Grundprinzip

Einem Informanten stellt registrierten Nutzern einheitliche Informationen zur Verfügung. Die Nutzer werden benachrichtigt, wenn neue Informationen vorhanden sind. Im pull-Verfahren holen die Nutzer die Informationen selbstständig beim Informanten ab, im push-Verfahren werden alle Informationen vom Informanten an alle Nutzer weitergegeben.

Alltagsbeispiele

Zeitungsabonnement push-Verfahren

Unterrichtsstunde push-Verfahren (bei Lehrer J.)

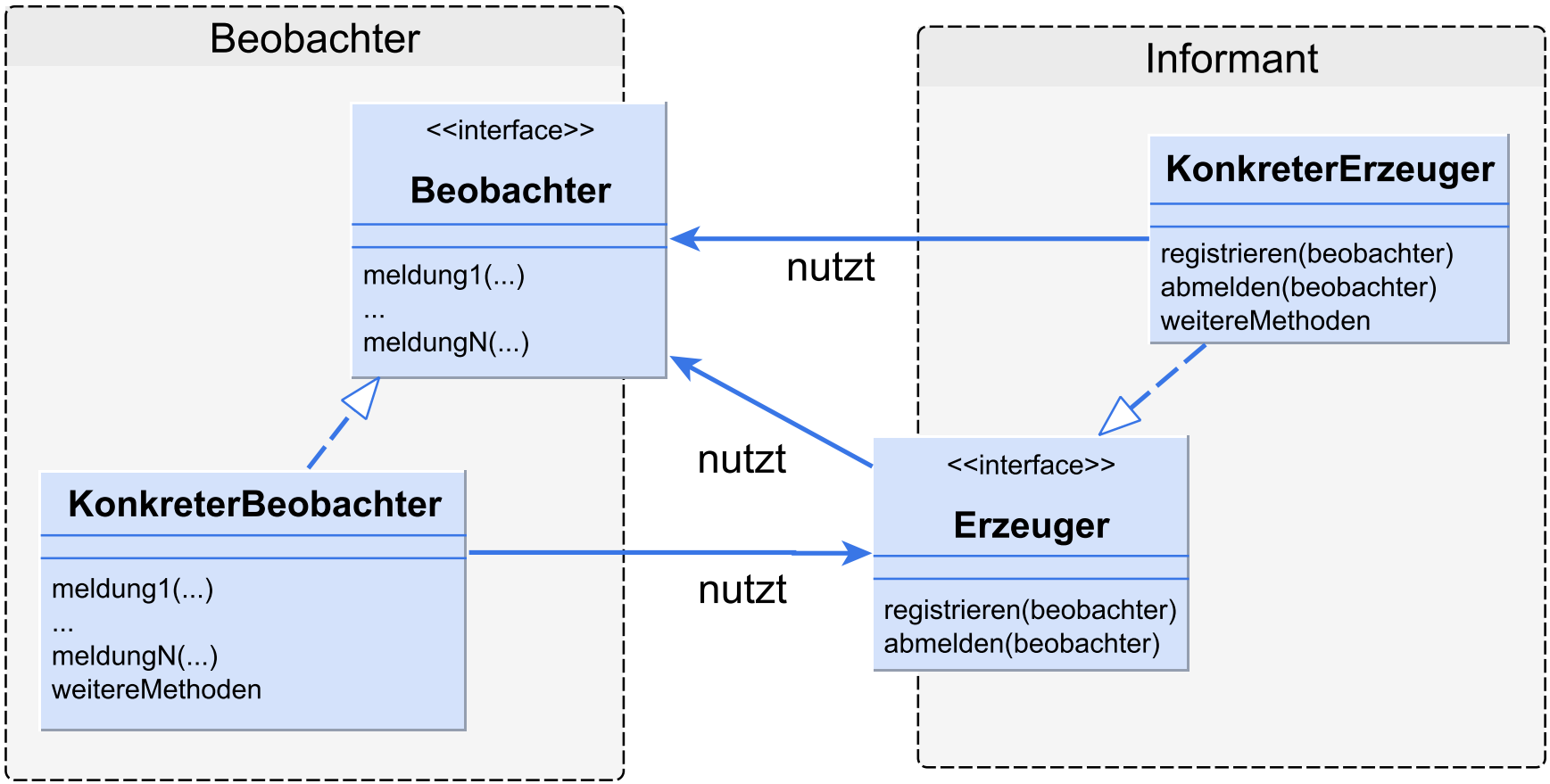
Newsletter pull- und push-Verfahren

Beobachtermuster (Grundprinzip)

Beobachtungsmuster . . .

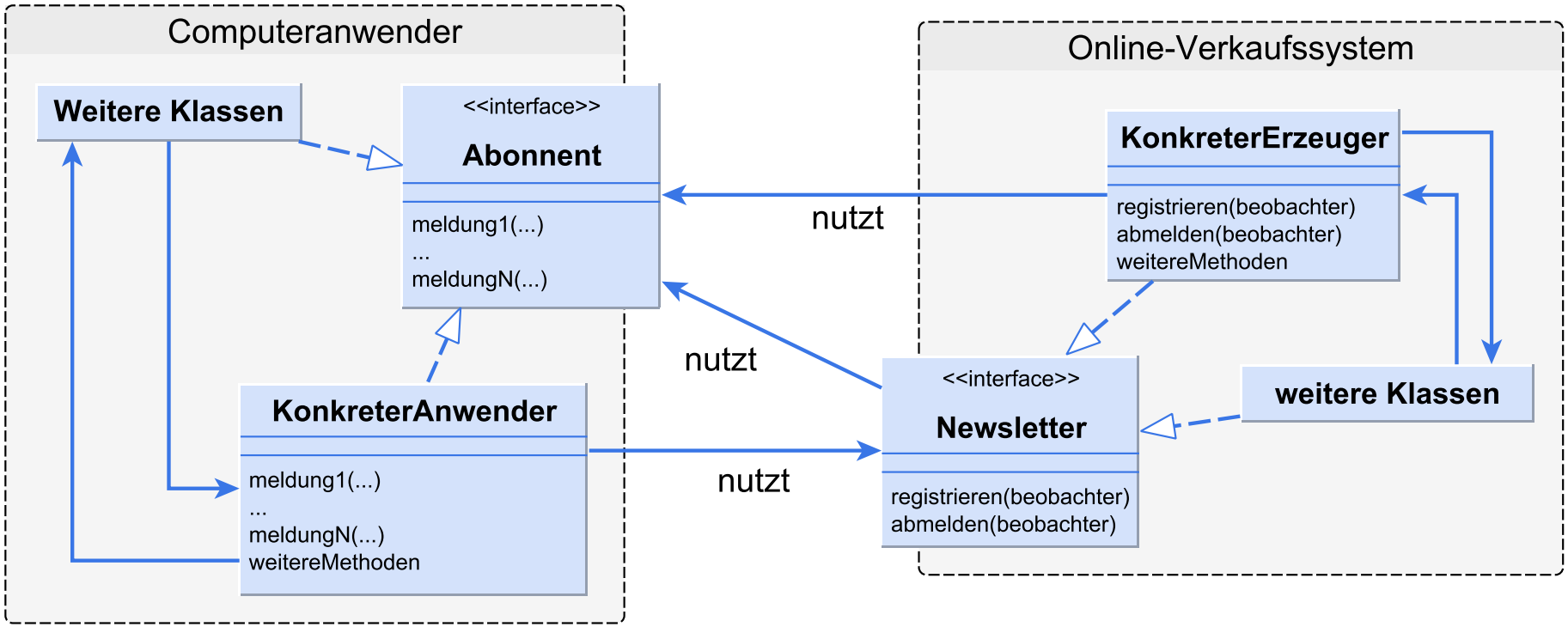
. . . halten, die Kommunikation zwischen einem Erzeuger von Information (Informaten) und den Interessenten an dieser Information (Beobachter) offen und flexibel.
Der Informant bietet ein Interface an, das Interessenten nutzen können. Dazu müssen sich die Beobachter beim Informanten registrieren lassen.
Die Beobachter müssen ihrerseits garantieren, dass sie alle Botschaften des Informanten verstehen.

Beobachtermuster



Beispiel Unterricht

Beispiel 1: Newsletter



Vorteile von Beobachtermustern

- Neue Beobachter können jederzeit hinzugefügt werden, ohne dass der Informant geändert werden muss.
- Beobachter und Informant können unabhängig voneinander wiederverwendet werden.
- Informant und Beobachter können beliebig geändert werden, so lange ihr Interface gleich bleibt.

Nachteile von Beobachtermustern

- Gibt es viele Beobachter, kann die Benachrichtigung durch den Informanten sehr zeitaufwändig werden.
- Jeder Beobachter wird mit der gleichen Information versorgt auch wenn er gar nichts oder nur Teile davon benötigt. (Beispiele: Newsletter, Werbung)

Beispiel 2: Eventhandling in Java

- Informant: Swing-Komponenten (Schaltflächen, Buttons, etc.)
- Beobachter: Sie werden Listener genannt, können sich bei den Swing-Kompnenten registrieren (Jeder Swing-Button erbt von `AbstractButton` Methoden zum An- und Abmelden von Listenern).
- Findet ein Event an einer Swing-Komponente statt (z.B. ein Mausklick), so werden alle registrierten Listener von dieser Swing-Komponenten über das Ereignis benachrichtigt.
- Die Listener müssen die Methode `actionPerformed(ActionEvent)` implementieren, in der festgelegt wird, wie auf das Ereignis reagiert werden soll.

Beispiel 2: Eventhandling in Java

```
1  ...
2  // Button erzeugen
3  JButton knopf = new JButton("Klick mich");
4  fenster.add(knopf);
5
6  //Listener registrieren
7  knopf.addActionListener(
8     new ActionListener() {
9        public void actionPerformed ( ActionEvent e
            ){
10          System.out.println("geklickt: " + e.
               getActionCommand());
11       }
12    }
13    );
```

Beispiel 2: Eventhandling in Java

Beachte

Die runde Klammer in Zeile 6 wird erst in Zeile 12 geschlossen. D.h. der Methode `addActionListerer` wird eine Klasse übergeben, die hier erst definiert wird.
Alternativ könnte für die Zeile 7 – 11 eine eigene Klasse erstellt werden. Aber wer will schon für 50 oder 100 Events eigene Klassen erstellen.

Ü 2.4: ProfiGUI_3

In diesem Abschnitt

2. Entwurfsmuster
   - Allgemeines
   - Adaptermuster
   - Kompositum
   - Beobachtermuster
   - model-view-controller-Muster

model-view-controller-Muster (MVC)

Das model-view-controller-Muster (MVC)

basiert auf der klaren Trennung von

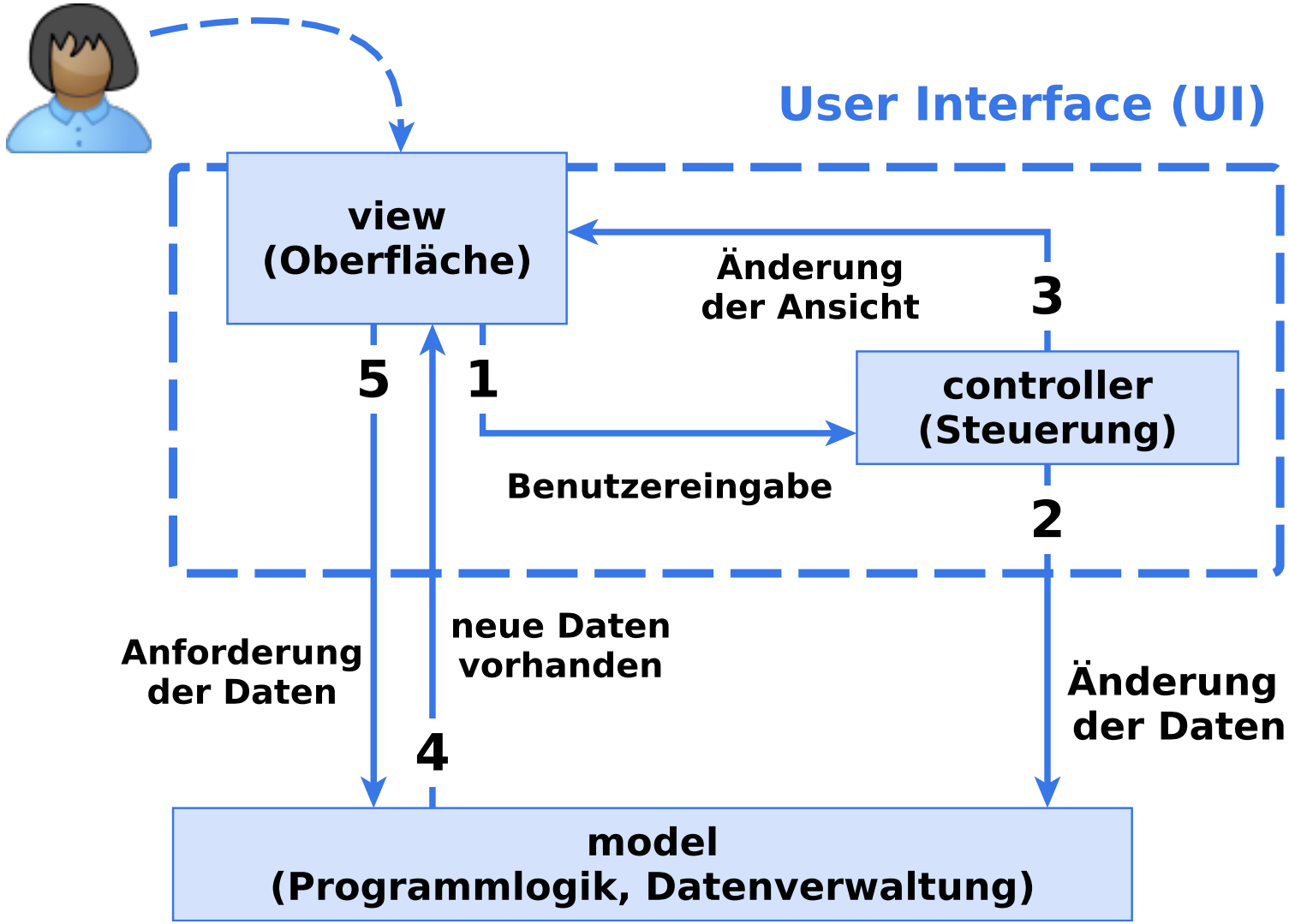
Modell funktionalem Kern des Systems

View Darstellung der Daten einschließlich Ein- und Ausgabefelder sowie Buttons

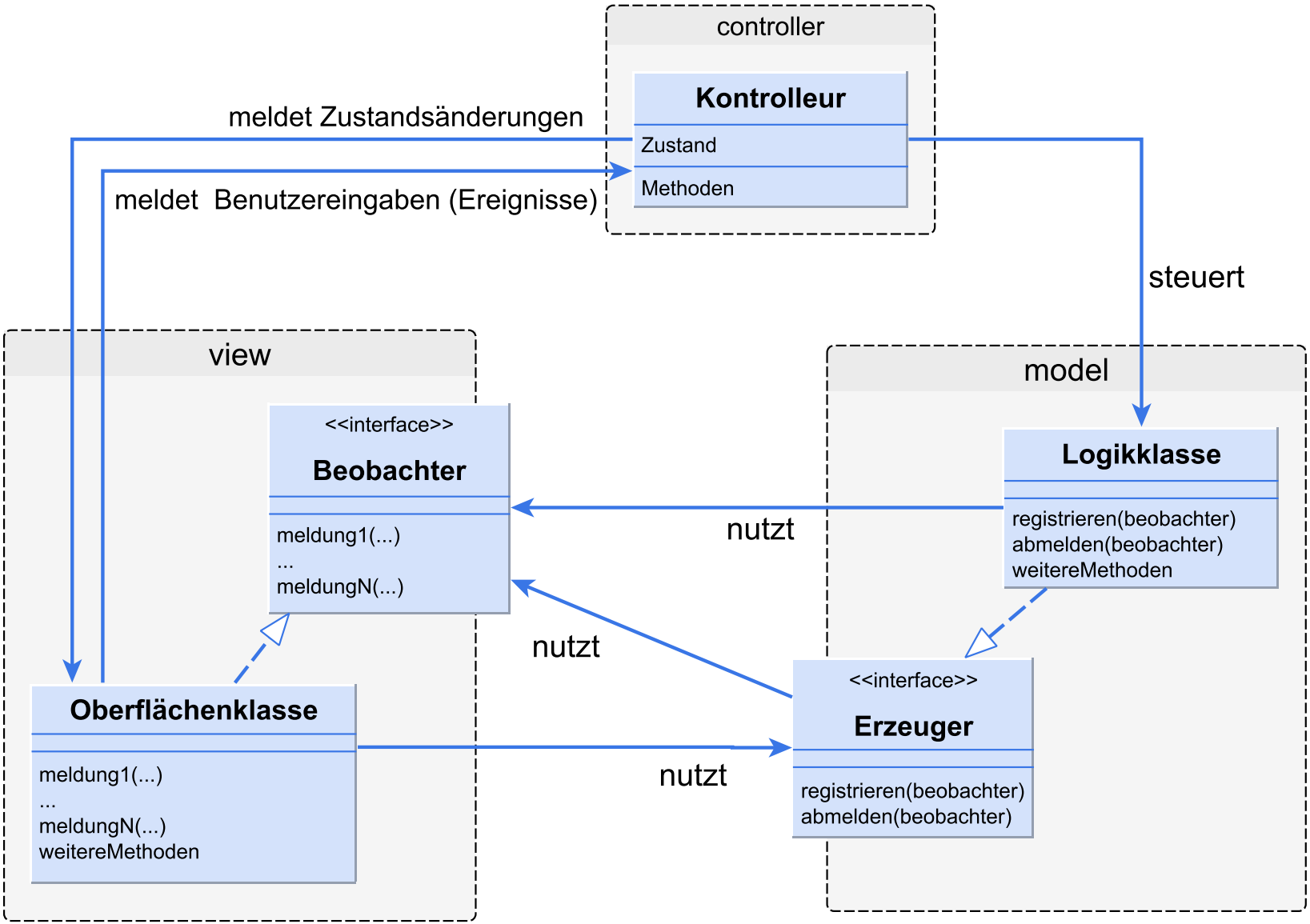
Controller Entgegennahme und Verarbeitung der Benutzereingaben

- View und Controller bilden zusammen das Benutzerinterface (UserInterface) und sind wechselseitig voneinander abhängig.
- View und Modell werden oft nach dem Beobachtungsmuster entwickelt wobei der View als Beobachter agiert.

Zusammenspiel der Komponenten



MVC-Klassendiagramm



Anwendungen

- Graphical-User-Interface (GUI)-Programmierung (Verwendung von MVC-Muster war der entscheidende Schritt)
- Verschiedene Endgeräte (Computer, Handy, ipad) nutzen gleiches Programm (E-Mail)
- Nutzung verschiedener Web-Browser

Vorteil des MVC-Musters

Vorteil des MVC-Musters

- spätere Änderungen oder Erweiterungen werden erleichtert,
- die Wiederverwendbarkeit der einzelnen Komponenten wird ermöglicht.

In diesem Abschnitt

3. Praktische Softwareentwicklung
   - Profi GUI für Graphen
   - Simulation eines Rangierbahnhofes

In diesem Abschnitt

3. Praktische Softwareentwicklung
   - Profi GUI für Graphen
   - Simulation eines Rangierbahnhofes