Einführung in die Programmierung für NF MI

Übung 11

Inhalt

- Design Patterns (Entwurfsmuster)
- MVC und Observer Pattern

Design Patterns

A Design Pattern is a solution to a problem in a context

- Context
  - Situation, in der das Pattern anwendbar ist
- Problem
  - Ziele und Einschränkungen durch den Kontext
- Solution
  - Das Ziel mit den Einschränkungen erreichen

Bestandteile eines Design Pattern

- Name
- Motivation
- Applicability
- Participants
- Structure
- Collaboration
- Consequences
- Related Patterns

Design Pattern

- Pattern legen meist nur grobe Richtlinien fest, die sich bewährt haben
- Die konkrete Umsetzung im konkreten Fall ist immer abweichend und individuell und kann von Fall zu Fall variieren
- Daher auch im Deutschen: Entwurfsmuster

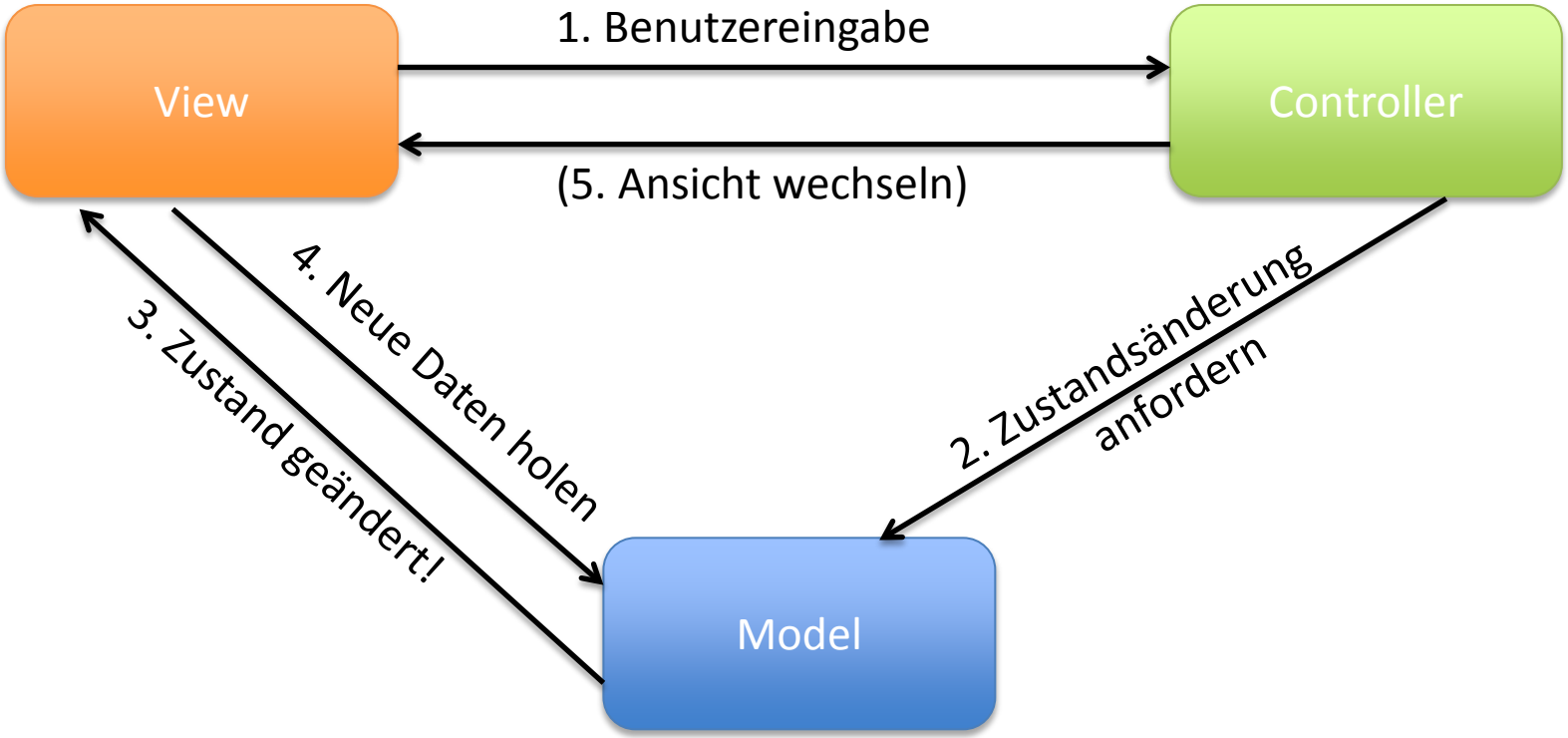
Das MVC Pattern

- MVC = Model, View, Controller
- Problem: Benutzerschnittstellen
- Viele unterschiedliche Anforderungen:
  - Layout, Benutzerführung
  - Benutzerinteraktion, Eingabevalidierung
  - Datenvisualisierung, Datenhaltung
  - Anwendungslogik
  - u.v.m.

Das MVC Pattern

- Aufteilung dieser Anforderungen in
  - Model
    - Anwendungslogik, Datenhaltung
  - View
    - Layout, Datenvisualisierung, Benutzerinteraktion
  - Controller
    - Vermittlung, Eingabevalidierung
- Ziel: Eine skalierbare und übersichtliche Programmstruktur

Das MVC Pattern - Nutzereingabe



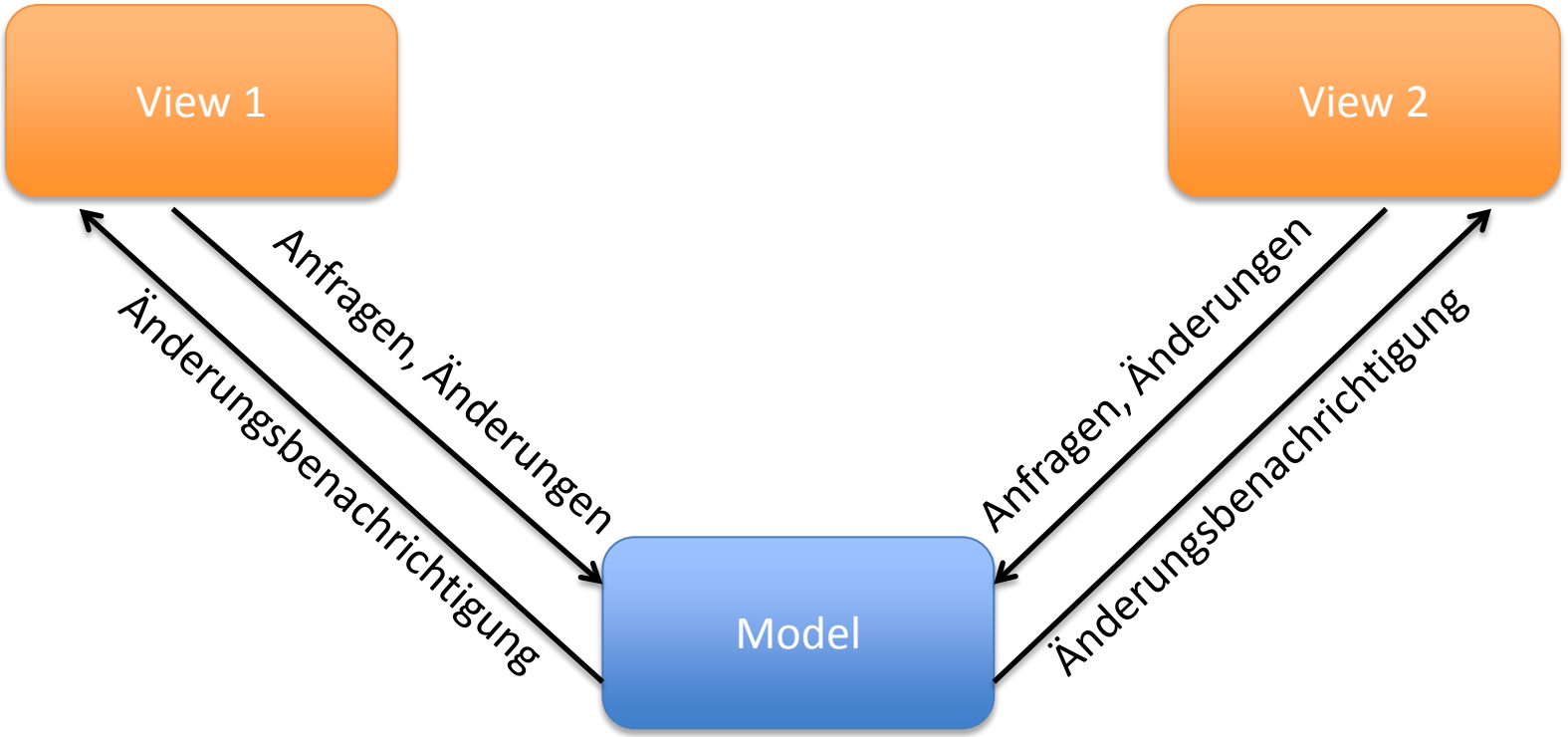
Das MVC Pattern

- In kleinen Programmen wird oft auf den Controller verzichtet
- Dieser schafft hauptsächlich eine Austauschbarkeit mehrere Models oder Views
- Es bleibt die Problematik, wie die einzelnen Klassen aufeinander zugreifen

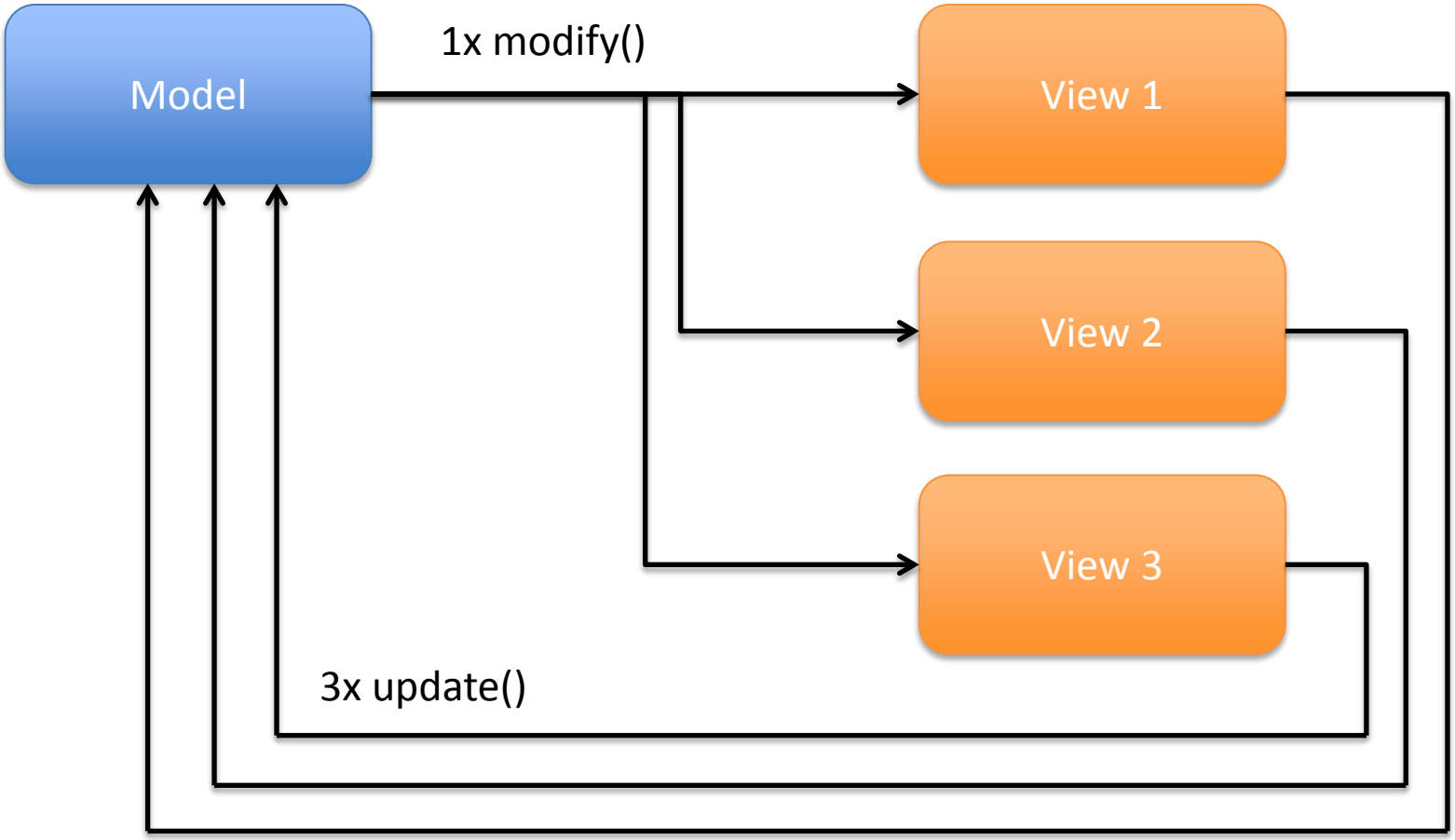
Das Observer Pattern

- Das Observer Pattern behandelt die Kommunikation zwischen zwei Objekten, wenn sich eines aufgrund des anderen ändern / anpassen muss
- Beispiel: Model und View
- Wenn eine View die Daten eines Models anzeigt, muss sie sich ändern, wenn sich die Daten ändern

Problem


View 1
View 2
Anfragen, Änderungen
Änderungsbenachrichtigung
Anfragen, Änderungen
Änderungsbenachrichtigung
Model

Starre Lösung



Starre Lösung

- Probleme der starren Lösung:
  - Das Model muss jede View kennen
  - Hinzufügen oder Entfernen von Views erfordert jedes Mal eine Änderung am Model
  - Jede View muss ihr Model kennen
  - Die update()-Methode kommt dreimal vor und muss in jeder View quasi identisch programmiert werden

Flexible Lösung

- Einführung von Observer und Observable
- Observer = „Beobachter“
- Observable = „Beobachteter“

- Das Model soll ein Beobachteter sein
- Jede View ist ein Beobachter

Flexible Lösung

- Realisierung in Java durch die Bibliotheksklasse `Observable` und das Interface `Observer`
- Model erbt von Observable

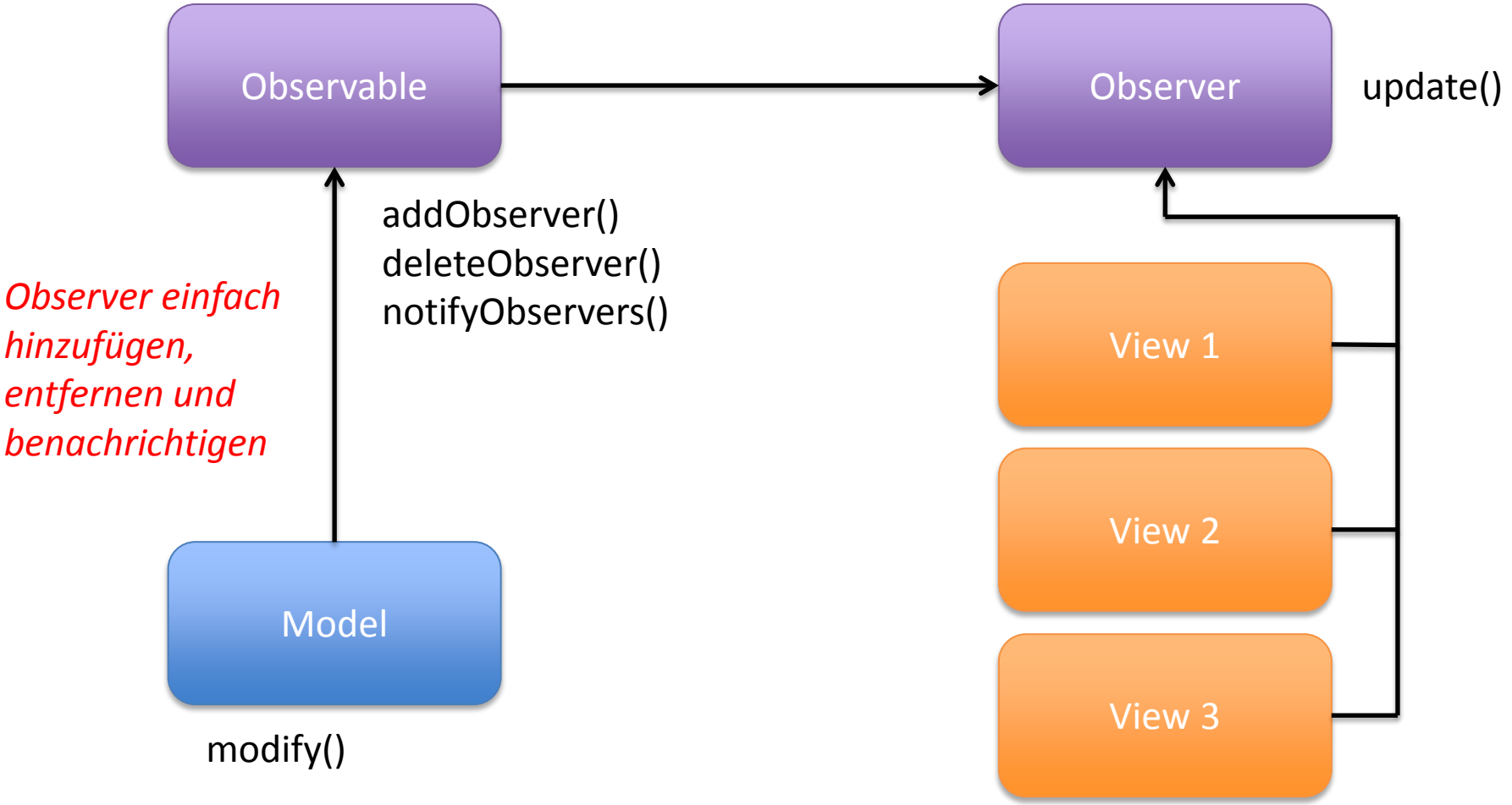
  ```
  class Model extends Observable
  ```

- View implementiert Observer

  ```
  class View implements Observer
  ```

Flexible Lösung

Eine gemeinsame Methode



Observer benachrichtigen

- Wenn sich im Observable etwas ändert werden folgende Methoden aufgerufen:

```
setChanged();
notifyObservers();
```

- Wenn `notfiyObservers()` aufgerufen wird, wird automatisch die `update()`-Methode im Observer ausgeführt

Observer benachrichtigen

- Bei der Benachrichtigung können auch weitere Informationen an die update-Methode „übergeben“ werden
- Beispiele gibt es zum Download auf der Vorlesungshomepage und unter

http://openbook.galileocomputing.de/javainsel/javainsel_10_002.html

Kombination MVC und Observer

- Wenn das MVC Pattern vollständig umgesetzt wird, werden die Observer im Controller dem Observable hinzugefügt
- Der Controller erstellt also ein neues Model, eine neue View und fügt dem Model die View als Observer hinzu
- Wenn nun im Model die Observer benachrichtigt werden, wird in der View automatisch die update()-Methode aufgerufen

Fragen zum Übungsblatt?