ABB Technikerschule

Technik. Informatik. Wirtschaft. Management →

Dominik Meyer
Software Engineering

MVC + Persistenz

Ziel

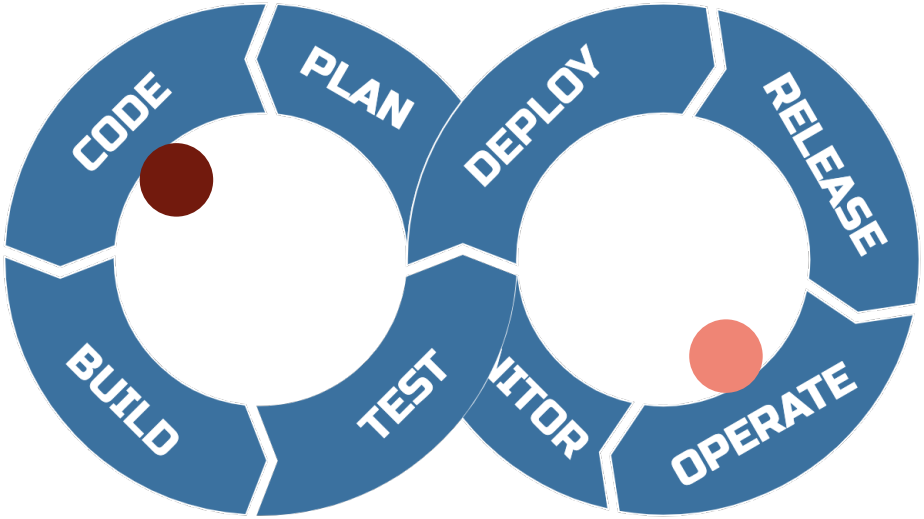
Nach der Lektion begründen die Studierenden den Einsatz des Model-View-Controller Musters (nicht Entwurfs-Muster!).

Nach der Lektion können die Studierenden bestehende Persistenz-Lösungen für Java abfassen.

Persistenz

– MySQL Container
– Datenbank Wissen aus DBM
– Entwurfsmuster
– Architekturen (Hintergrund MVC)

– Applikation mit «echten» Daten
– Applikation, deren Datenmodell sich ändert



Agenda

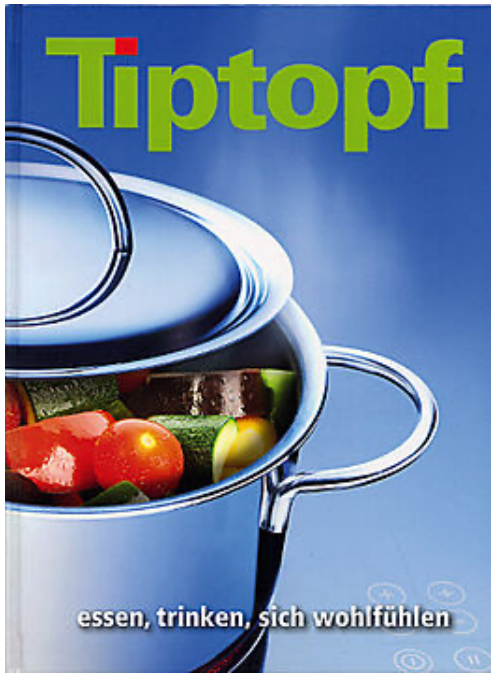
- Ziel
- MVC
- Datenbanken anbinden
  - Nicht Datenbanken selbst, dieses Modul hattet ihr bereits!
- Zielkontrolle

MVC

- Feedback: MVC anschauen
- MVC (Artikel dazu)
  - kein Entwurfsmuster wie die anderen
  - «ein Weg, den Code zu organisieren»
  - Basierend auf Architektur- und Organisations-Gedanken
  - Anwendungsfall
    - Entscheidung des Projektteams aber generell eine gute Wahl
    - Alternativen
  - Beispiele folgen im Verlauf dieser Lektion und in der Transferaufgabe

MVC

Model



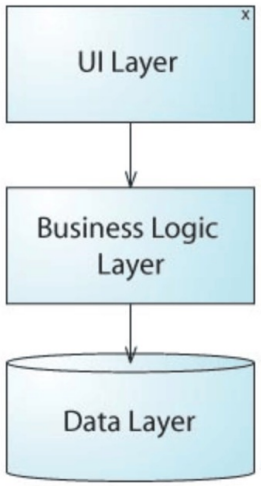
Controller

View

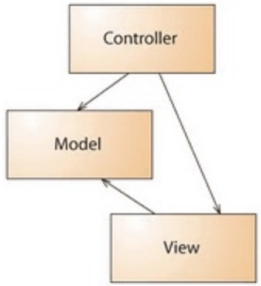
MVC

Difference between MVC and 3-tier architecture:

Here, 3-tier design were like this:



The MVC pattern would be:



- Model
  - Was der Nutzer sieht/nutzt
  - API: REST API
  - Web: UI
- Controller
  - Aller Code, der Logik oder Prozesse in irgendeiner Form handhabt
  - Meist auch der Code, der selber geschrieben wird
- Model
  - Zutaten für die Controller
  - Daten oder Zugriffe auf Daten, welche zum Erfüllen der Logik oder Prozesse benötigt wird

Perfekter Übergang

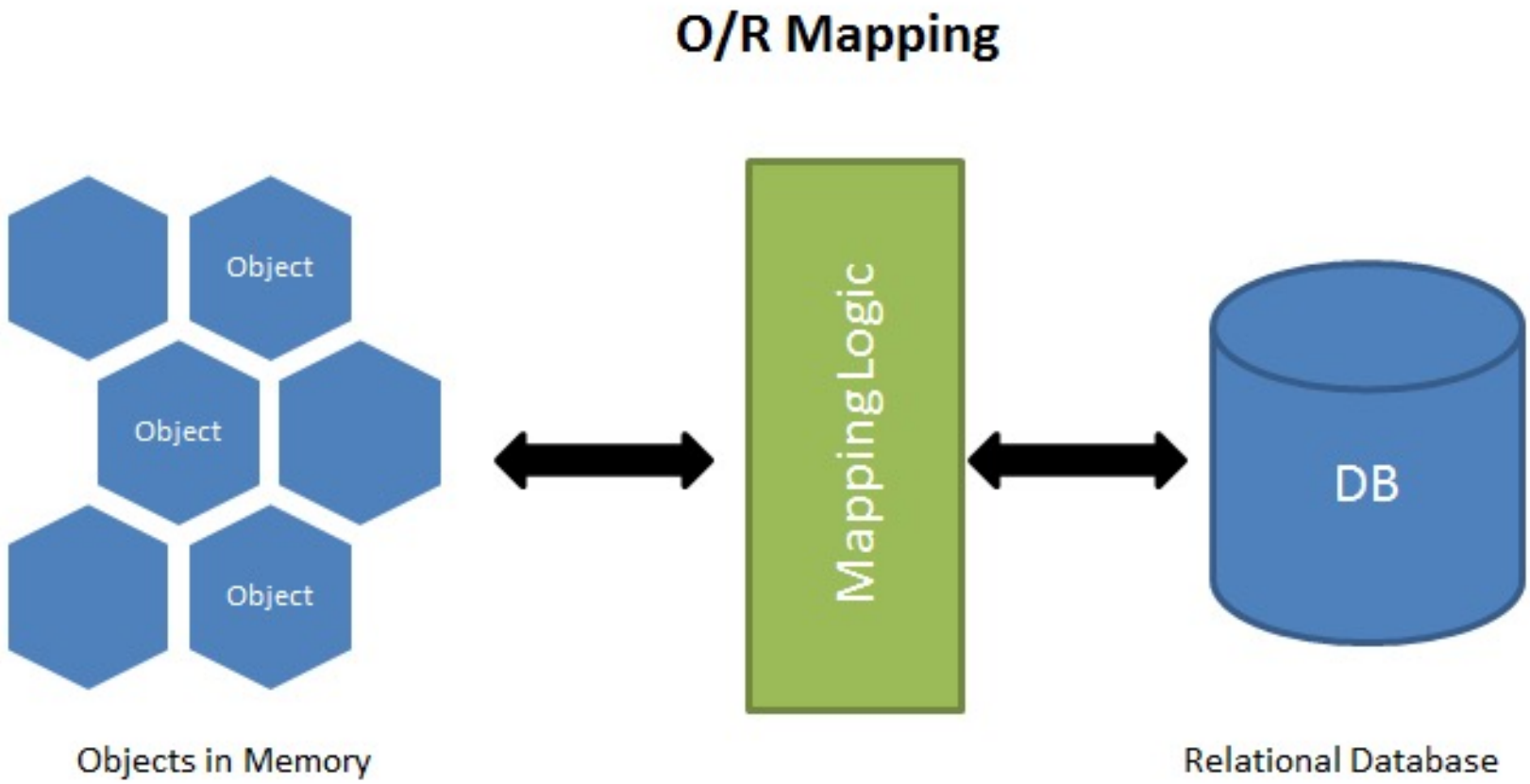
Was wir nicht anschauen

– Alles was in DBM vorkam.

Wir setzen also voraus, dass alle noch wissen:

- Was ist eine Datenbank
- Wozu brauchen wir die
- Was sind Datenbanken, Tabellen und Schemen
- Was sind Datentypen
- Wo können wir die hosten (Hinweis, Docker…)
- Alles was sonst vorkam

Was wir anschauen



Optionen

- ORM nutzen
- Selber alle Abfragen schreiben (in Klassen)
  - Beispiel

SO Frage dazu

Wir nutzen ein ORM, weil es uns viel repetitive Arbeit abnimmt und für 99% der Fälle ausreichend ist.
Ebenso werden Java ORMs seit gefühlt zwanzig Jahren entwickelt, sind also für unsere Fälle erwachsen genug.

ORM

Migrationen (Änderungen am Modell)

– Immer rückwärts-kompatibel! Cloud!

– Eines Tages haben wir bestehende Daten
  – Können also den Container nicht einfach wegwerfen 
– Wir müssen beschreiben, welche Änderungen wir am bestehenden Modell (und den Daten darin) machen wollen
– das sind **Migrationen**
  – SQL Skripte wie in DBM
  – Automatisch ausgeführt vor Applikationsstart
  – Mit Log-Buch, was bereits erfolgreich ausgeführt wurde
  – Transaktional

H2

- H2 ist eine in-memory Datenbank
  - im Arbeitsspeicher
  - flüchtig
- +
  - Wunderbar geeignet für unsere Beispiele!
- -
  - Keine riesige Community, weniger Support
  - Feature-ärmer als alte Bekannte (MySQL, PostgreSQL etc.)

- In der Transferaufgabe ersetzen wir die flüchtige H2 durch eine MySQL Datenbank
  - Nach Wahl eine PostgreSQL Datenbank

Active Record (Anti)-Pattern

- https://en.wikipedia.org/wiki/Active_record_pattern
- Pattern oder Anti-Pattern
- Die Frage: Wann *synchronisieren* wir die Datenbank mit dem Code?
  - Option A: Bei jeder Änderung, egal wer diese auslöst (1 Row = 1 Object)
  - Option B: Explizit, wenn das Programm danach fragt (Model based)

- Active Record ist **Option A**
- JPA aus den vorherigen Slides ist nicht aktiv per se
- ActivJPA würde JPA «*Active Record*» machen, ist aber für uns nicht nötig
- Anwendungsfall: Proof of Concepts & Kurz-lebige Applikationen
  - Artikel
  - Also eher nicht ☺

Auswerten

Zielkontrolle

AUFTRAG
⏰ 5min

https://forms.office.com/r/6JzYeeyTYF

🔐 ABB Login

WEITER WISSEN.

Wir begleiten Sie!