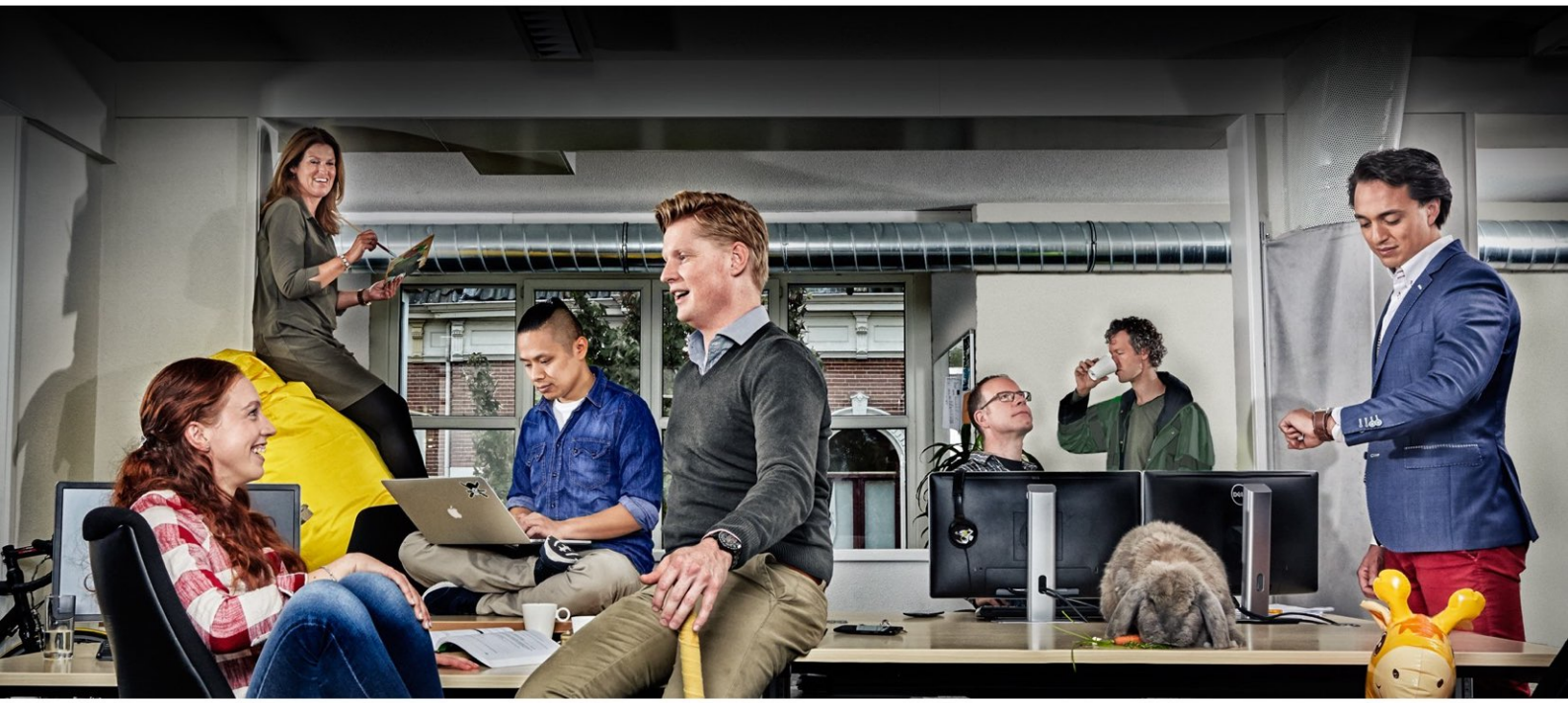
topicus
ACADEMY



TA-210

Object-Relationele Mapping met JPA en Hibernate

Docentenhandleiding

TA210 Object-Relationele Mapping met JPA en Hibernate

Topicus Academy

Inhoudsopgave

Inleiding

In het opleidingsprogramma voor software ontwikkelaars is behoefte aan een inleiding tot Object Relationele Mapping (ORM) gebaseerd op de Java Persistence API (JPA). Dit is de studiegids voor deze inleiding.

Met deze inleiding wordt beoogd dat de ontwikkelaar weet waarvoor ORM gebruikt wordt, waar in de architectuur zich dit bevindt en de betreffende ORM mappings kan lezen.

De volgende onderwerpen komen aan bod:

- ▢ wat is ORM, wat is JPA
- ▢ waar zit dit in de architectuur
- ▢ hoe doe je ORM
- ▢ wat zijn de JPA annotaties
- ▢ wat zijn overervingstrategieën
- ▢ wat zijn modelleringstrategieën voor overerving en relaties
- ▢ wat zijn performance overwegingen bij ORM

Deze course is met name bedoeld voor ontwikkelaars die in hun opleiding geen vak gevolgd hebben over ORM.

Voorkennis

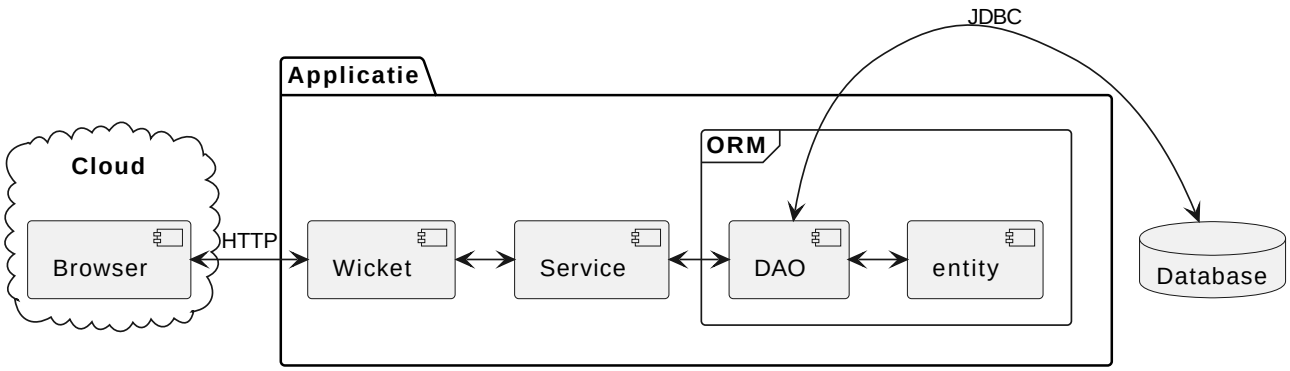
Vereiste voorkennis is dat de ontwikkelaar kennis heeft van de volgende onderwerpen:

- Java classes, fields, overerving
- Databases: tabellen, kolommen, kolomtypes, foreign keys, constraints
- UML: class diagram

De course

Object-Relationele Mapping wordt gebruikt om de gegevens die in de database staan (automatisch) te vertalen van en naar Java objecten. Hiermee wordt een behoorlijke versnelling van het ontwikkelproces bereikt.

Object-Relationele Mapping wordt in de Java wereld geïmplementeerd door de *Java Persistence API* (JPA), en *Hibernate* is daar een specifieke vorm van. Binnen Topicus Onderwijs wordt Hibernate gebruikt voor het mappen tussen de database en de Java code. In deze cursus staat Hibernate dan ook centraal.



Competenties

Je leert in deze cursus de volgende zaken:

- waar in de applicatie architectuur moet ik ORM plaatsen
- wat is ORM, JPA en Hibernate
- wat doet ORM
- hoe map je van Java naar SQL en vice versa met JPA annotaties
- hoe persisteer en verwijder je objecten uit de database
- hoe schrijf je in Java queries met JPA
- waarom en hoe beperk je het aantal resultaten
- hoe kan je class hiearchieën vertalen naar tabellen
- hoe kan je relaties tussen objecten vertalen naar tabellen
- caching van resultaten voor snellere applicaties

Materialen

Er is geen recent goed boek beschikbaar voor JPA 2.1 zodat je het moet hebben van online cursussen en stackoverflow antwoorden. [Oracle] publiceert wel een online "introductie" die goed kan dienen als referentiemateriaal. Er is wel een goed beoordeelde JPA 2.1 cursus op PluralSight beschikbaar via [agonc].

- [] Antonio Goncalves. Java Persistence API (JPA) 2.1. 2014.
- [] Oracle. Introduction to the Java Persistence API. 2014.

Chapter 1. Wat is ORM / JPA

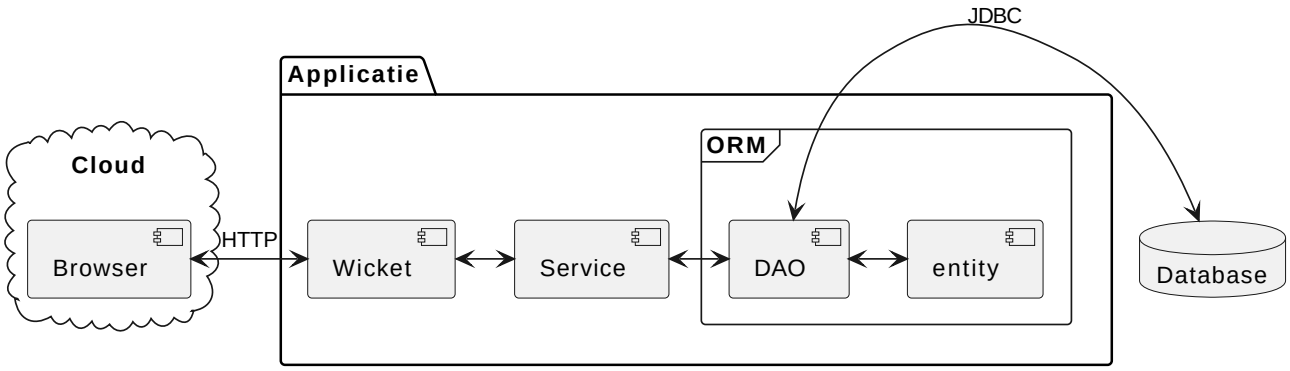
Lesplan voor de les "Wat is ORM" uit de cursus "Introductie ORM". Dit zijn aantekeningen voor de cursusleider om alle belangrijke begrippen de revue te laten passeren.

Het doel van deze les is dat de ontwikkelaar weet waarvoor ORM gebruikt wordt, welke begrippen daarbij horen en hoe zich dat verhoudt tot Java EE. Daarnaast leert de ontwikkelaar hoe de verhouding tussen de database en Java code ligt, hoe de mapping geschiedt met annotaties en een paar simpele annotaties.

Na deze sessie kan de ontwikkelaar zich enigszins redden in een entity class.

1.1. Wat is ORM en waar zit dat in de architectuur?

Begin met tekenen van architectuurplaat:



Merk op:

- verschillende lagen met elk hun verantwoordelijkheid: UI, service (business logica) en data (database opslag)
- we hebben database technologie en Java technologie
- ORM is de technologie die het mogelijk maakt om van Java naar database tabellen te gaan en weer terug
- ORM leeft in de Java wereld in het blokje ORM:
- het blokje ORM omvat:
  - DAOs voor de queries en database acties
  - Entities voor de mapping tussen database tabellen en Java objecten.
- ORM heet in Java de *Java Persistence API* (JPA)
- JPA is een standaard waarvan verschillende implementaties (varianten) van zijn
- JPA is onderdeel van de Java Enterprise Edition standaard (Java EE)
- Java EE is een combinatie van standaarden zoals web technologie, persistentie, caching, computer-computer communicatie (messaging) enz, en JPA is er één van.
- Hibernate is een JPA implementatie (variant)

- JPA, ORM en Hibernate wordt door elkaar heen gebruikt als terminologie (voor ons is Hibernate hetzelfde als JPA)

1.2. Hoe werkt ORM precies?

Houd er rekening mee dat je 3 kolommen nodig hebt op je bord:

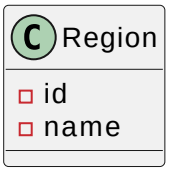
1. tabel
2. UML class
3. Java code voor class

Het liefst houd je 50% over voor de Java code.

Teken een tabel `regions` (uit de Introductie SQL), gebaseerd op:

```
CREATE TABLE regions
    ( region_id       NUMBER
      CONSTRAINT  region_id_nn NOT NULL
    , region_name     VARCHAR2(25)
    );
```

En teken een UML class `Region` met daarin de velden `id` en `name`:



merk op dat je zo'n Java class ook 'Entity' of 'Entiteit' noemt.

Schrijf daarnaast de Java code uit voor de Java class:

```
①
public class Region {
    ①
    private Long id;

    ①
    private String name;

    public Long getId() {}
    public void setId(Long id) {}
    public String getName() {}
    public String setName(String name) {}
}
```

① Houd genoeg ruimte over voor het toevoegen van annotaties

Vraag naar de verschillen tussen de Java code en de database. Dit zijn dus met name de naamgeving van de class versus tabel en de kolommen versus de velden.

Merk op dat ORM hier juist voor bedoeld is en dat de manier zoals wij ontwikkelen (Java first) niet de enige manier van software bouwen is (andere bedrijven doen schema first, of je erft een bestaand schema).

Werk uit hoe dit in traditioneel Java koppelt met de database:

```
public Region getRegionById(Long id) throws SQLException{
    try (
        Connection connection = datasource.getConnection();
        PreparedStatement ps = connection.prepareStatement(
            "SELECT region_id, region_name "
          + "FROM regions "
          + "WHERE region_id = ?");
    ) {
        ps.setParameter(1, id);
        try (ResultSet rs = ps.executeQuery();) {
            while(rs.next()) {
                Long id = rs.getLong(1);
                String name = rs.getString(2);
                Region r = new Region();
                r.setId(id);
                r.setName(name);
                return r;
            }
        }
    }
}
```

Laat de cursisten beantwoorden welke onderdelen erg onderhoudgevoelig zijn en welke onderdelen redelijk dom werk zijn.

Merk op dat je dit voor elke entiteit moet doen, voor elke query en voor elke actie (creates, updates en deletes). Merk ook op dat je hier geen controle hebt of er slechts één record gevonden is bij de query.

De beste programmeur is een luie programmeur dus deze vlieger gaat niet op. Daarom JPA.

1.3. ORM annotaties toevoegen

Vraag wat een annotatie is.

Leg nog een keer kort uit dat een annotatie een uitbreiding is op een Java class, veld, parameter of methode om extra informatie daarbij op te slaan die later weer uitgelezen kan worden door

programmacode (zoals Hibernate).

Begin met het maken van een entiteit van de `Region` class door de `@Entity` annotatie toe te voegen. JPA weet nu dat deze class onder zijn regie valt en zal dus zijn magie er op los laten.

Merk op dat JPA standaard alle velden automatisch gebruikt voor de mapping tussen Java en SQL.

Vraag wat er nu mis gaat

JPA weet niet welke tabel hierbij hoort. Dus de `@Table` annotatie toevoegen.

Vraag wat er nu nog mis gaat

JPA weet niet welke kolom bij `id` en `name` hoort. Standaard wordt de veldnaam gebruikt. Voeg de `@Column` annotatie toe.

Wat mist er nog?

De kolom "region_id" mag niet `null` zijn. Voeg `nullable = false` toe aan de column annotatie.

Er is nog iets niet goed: het `id` veld is niet zomaar een kolom, het is een `@Id` kolom. Wanneer je dit probeert te starten in Hibernate zie je dat Hibernate een foutmelding geeft dat de entiteit `Region` geen ID value heeft.

Regel: een entiteit is een object met een ID.

Chapter 2. Code voor uitlezen

De code voor het uitlezen van een `Region` met JPA is vrij eenvoudig:

```
public Region getRegionById(Long id) {
    EntityManagerFactory factory = Persistence
            .createEntityManagerFactory("persistenceUnitName");
    entityManager = factory.createEntityManager();
    return entityManager.find(Region.class, Id);
}
```

Vertel dat normaal gesproken de `entityManager` door *Wildfly* wordt geleverd en dat je die niet hoeft op te zoeken. De code wordt dus nog 2 regels korter.

De code voor het opslaan is ook simpel:

```
@PersistenceContext ①
private EntityManager entityManager;

public void save(Region region) {
    entityManager.persist(region);
}
```

① Zorgt ervoor dat *Wildfly* de entity manager *injecteert* in jouw object.

Merk op dat dit natuurlijk alleen werkt wanneer de applicatie binnen _Wildfly draait. Daarbuiten moet je er zelf voor zorgen.

2.1. Afsluiting

Vertel dat er nog veel meer annotaties zijn en dat je met die annotaties relaties tussen de classes kan vastleggen. Er is ook nog een heleboel te leren over het schrijven van queries met de JPA-API, en dat komt in de volgende lessen aan bod.

- Stel controlevragen over wat is JPA, Hibernate en ORM.
- Waarom is `@Entity` belangrijk?
- Moet een database schema overeenkomen met je Java code?

2.2. Huiswerk

Laat de ontwikkelaars het opleidingsprogramma project van Github (https://github.com/topicusonderwijs/opleidingsprogramma) clonen naar hun lokale workspace. En daarna het "Intro ORM" project importeren.

- Run de `Main` class uit de package `orm`.
- Als het goed is werkt dit out-of-the-box en zie je onder andere een lijst van regio's in de console

voorbij gaan, naast een hoop Hibernate spam.

```
{"id":1,"name":"Europe"}
{"id":2,"name":"Americas"}
{"id":3,"name":"Asia"}
{"id":4,"name":"Middle East and Africa"}
```

Chapter 3. Mapping basics

In deze les komen de verschillende mogelijkheden aan bod om de kolommen en relaties tussen tabellen te modelleren met JPA annotaties.

De volgende onderwerpen zullen de revue passeren:

- `@Entity`: geeft aan dat de class een JPA entiteit is
- `@Table`: geeft aan welke tabel gebruikt moet worden voor de entiteit
- `@ManyToOne`: een adres ligt maar in één land
- `@OneToMany`: een land heeft meerdere provincies
- `@ManyToMany`: een land kan veel rivieren door zich hebben stromen, en een rivier kan door veel landen stromen
- `@OneToOne`: een account heeft één persoon en een persoon heeft slechts één account
- `@Temporal`: geeft aan of je alleen geïnteresseerd bent in de datum, tijd of de datum en de tijd
- `@Column`: bevat alle opties voor die voor de gemarkeerde kolom gelden
- `@JoinColumn`: bevat alle opties voor die voor de gemarkeerde `@ManyToOne` kolom gelden
- `@Id`: markeert dit veld (en de bijbehorende kolom) als identifier voor JPA. Elke entiteit moet er één hebben.
- `@Basic`: geeft aan dat deze kolom opgeslagen moet worden als een basistype (gebruiken we weinig)
- `@Transient`: geeft aan dat dit veld niet gebruikt moet worden door JPA
- `@Enumerated`: geeft aan hoe je een `enum` opslaat in de database
- `FetchType`: geeft aan of JPA de kolom of de relatie meteen moet ophalen, of alleen wanneer die nodig is

3.1. Vorm

De vorm is een sessie bij een whiteboard en een beeldscherm. Je dient na afloop van de sessie het huiswerk te maken en dat zal dan besproken worden in een volgende sessie.

3.2. Huiswerk

Maak de opgaven in de class `HuiswerkLes2`:

- [x] opdracht_1_voegConstraintsToeAanRegion(EntityManager)
- [x] opdracht_2_voegConstraintsToeAanRegion(EntityManager)
- [ ] opdracht_3_voegConstraintsToeAanCountry(EntityManager)
- [ ] opdracht_4_voegConstraintsToeAanCountry(EntityManager)
- [ ] opdracht_5_voegConstraintsToeAanLocation(EntityManager)

- ▢ opdracht_6_voegConstraintsToeAanLocation(EntityManager)
- ▢ opdracht_7_voegConstraintsToeAanLocation(EntityManager)
- ▢ opdracht_8_voegConstraintsToeAanDepartment(EntityManager)
- ▢ opdracht_9_voegConstraintsToeAanDepartment(EntityManager)
- ▢ opdracht_10_voegConstraintsToeAanEmployee(EntityManager)
- ▢ opdracht_11_voegConstraintsToeAanEmployee(EntityManager)
- ▢ opdracht_12_voegConstraintsToeAanEmployee(EntityManager)
- ▢ opdracht_13_voegConstraintsToeAanEmployee(EntityManager)
- ▢ opdracht_14_voegConstraintsToeAanJob(EntityManager)
- ▢ opdracht_15_voegConstraintsToeAanJob(EntityManager)
- ▢ opdracht_20_voegRelatiesToeAanCountry(EntityManager)
- ▢ opdracht_21_voegRelatiesToeAanLocation(EntityManager)
- ▢ opdracht_22_voegLocationRelatieToeAanDepartment(EntityManager)
- ▢ opdracht_23_voegEmployeesRelatieToeAanDepartment(EntityManager)
- ▢ opdracht_24_voegManagerRelatieToeAanDepartment(EntityManager)
- ▢ opdracht_25_voegDepartmentRelatieToeAanEmployee(EntityManager)
- ▢ opdracht_26_voegManagerRelatieToeAanEmployee(EntityManager)
- ▢ opdracht_27_voegJobRelatieToeAanEmployee(EntityManager)

Chapter 4. JPA Queries schrijven

- JPA metamodel
- queries schrijven
- aggregatie

Chapter 5. Levenscyclus van een entiteit

- ID generatie
- versionering
- attached/detached
- cache

Chapter 6. Overervings- en relatiestrategieën

- single table
- multi table
- many to many

Chapter 7. Advanced JPA

- custom data types
- query atoms (uit cobra)
- de gevaren zone