Spring Workshop II

Persistenz und ORM einfach gemacht / Spring Data

https://github.com/Michaeli71/AMTC_Spring_Workshop

Michael Inden

Freiberuflicher Consultant und Trainer

Speaker Intro

- Michael Inden, Jahrgang 1971
- Diplom-Informatiker, C.v.O. Uni Oldenburg

- ~8 ¼ Jahre **SSE** bei Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Jahre **TPL, SA** bei IVU Traffic Technologies AG in Aachen
- ~4 ¼ Jahre **LSA / Trainer** bei Zühlke Engineering AG in Zürich
- ~3 Jahre **TL / CTO** bei Direct Mail Informatics / ASMIQ in Zürich
- **Freiberuflicher Consultant, Trainer und Konferenz-Speaker**
- Autor und Gutachter beim dpunkt.verlag

E-Mail: michael_inden@hotmail.com
Blog: https://jaxenter.de/author/minden

Kurse: Bitte sprecht mich an!

https://github.com/Michaeli71/AMTC_Spring_Workshop

Agenda

Workshop Contents

- **PART 1: Einführung**
  - Überblick: JDBC / JPA
  - ORM
  - DAO Pattern
  - In Memory DB

- **PART 2: JdbcTemplate**

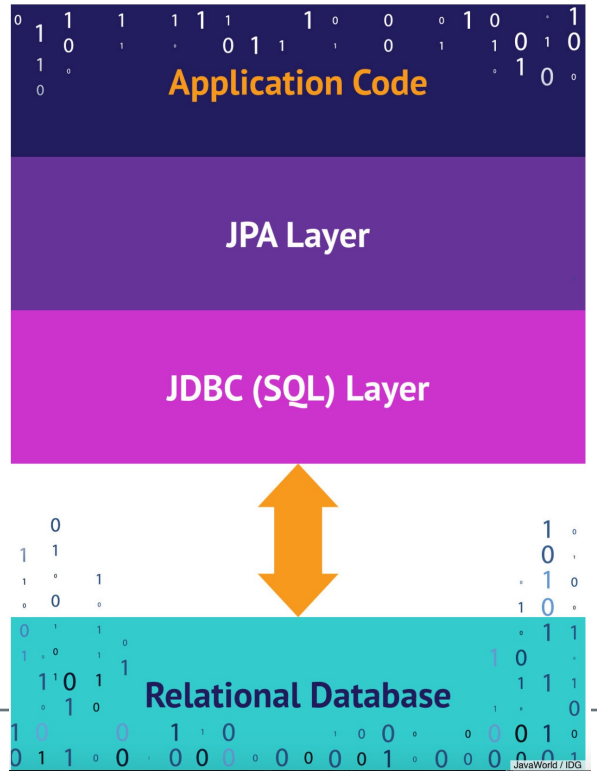
Workshop Contents

- **PART 3: Spring Data**
  - Einführung
  - Spring Data JPA Grundlagen
  - Spring Data Repositories
  - Spring Data Mongo DB
- **PART 4: Validierung**
- **PART 5: MapStruct**

PART 1:
Einführung

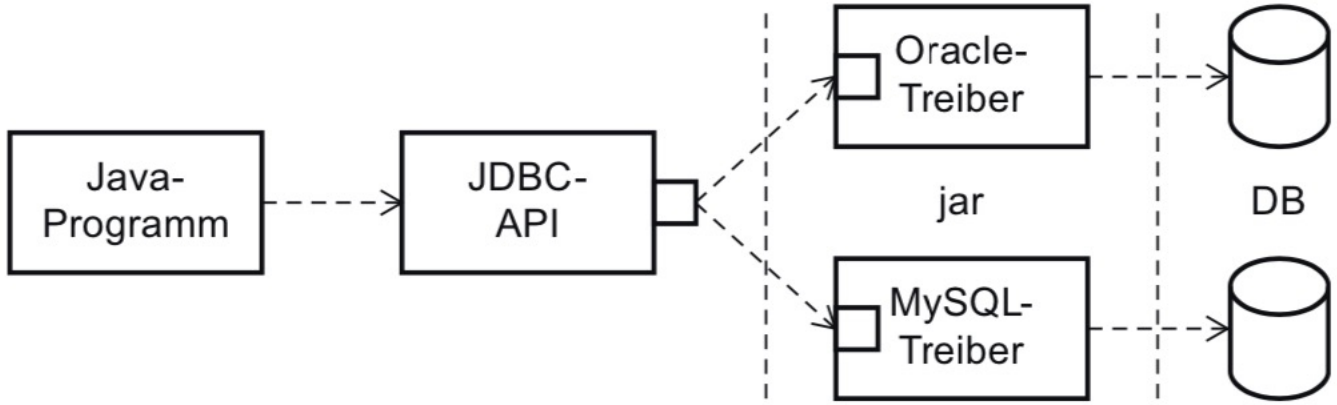
JPA Grundlagen

- Java Persistence API (JPA) ist eine Sammlung von Klassen und Interfaces zur Verwaltung von Daten in einer Datenbank
- JPA "vermittelt" zwischen Application / Domain Model und Datenbanken (RDBMS)



JDBC

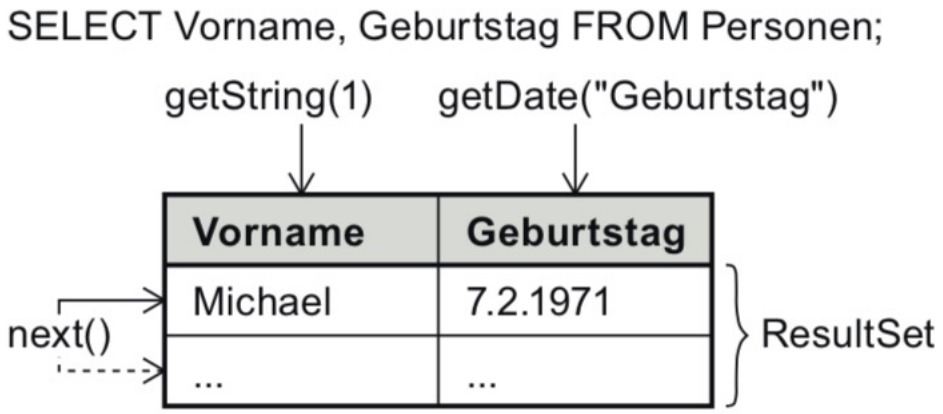
- JDBC = Java Database Connectivity
- Definiert einen Standard zum Zugriff auf relationale Datenbanken



- `javax.sql.DataSource` repräsentiert eine abstrakte Fabrik für Verbindungen zu Datenbanken

JDBC

- `java.sql.Connection` repräsentiert eine Verbindung zu Datenbank und erzeugt u.a. Statements und steuert Transaktionen usw.
- `java.sql.Statement` / `PreparedStatement` repräsentiert eine Anweisung, die ausgeführt werden soll
- `java.sql.ResultSet` repräsentiert Ergebnisse von Abfragen



JPA vs JDBC

- **JPA deutlich mehr high-level als JDBC**
  - Keine JDBC/SQL-Anweisungen durch Entwickler mehr nötig
  - Viele JDBC/SQL-Kommandos werden automatisch durch JPA generiert
  - JPA ist datenbankunabhängig, passt das SQL dann datenbankspezifisch an
  - Für Abfragen SQL ähnliche Sprache (JPQL)
  - Datenbank scheint zwar durch, aber viel mehr Abstraktion

- **JPA dient zum Object/Relational Mapping (ORM)**
  - Applikation arbeitet mit Objekten und dem Konzept der Entities
  - Entities werden auf Datenbanktabellen abgebildet
  - Kommandos und Aktionen als Methoden und nicht SQL-Befehle
  - Assoziationen und Vererbung lassen sich ohne Tricks realisieren

JPA Features

- **JPA gleicht mögliche Änderungen am Objekt-/Entity-Zustand automatisch mit der Datenbank ab (am Transaktionsende)**
- **Benutzer ändert also im Objektmodell und dies wird gleich in der Datenbank abgebildet, keine zusätzlichen SQL-Befehle nötig**
- **Selbst anspruchsvolle Dinge wie Assoziationen und Vererbung werden problemlos unterstützt:**
  - Automatischen Laden von referenzierten Objekten möglich
  - Automatisches Löschen von referenzierten Objekten möglich
- **JPA verwaltet die Entities und speichert diese zwischen (1st Level Caching)**

JPA Wissenswertes

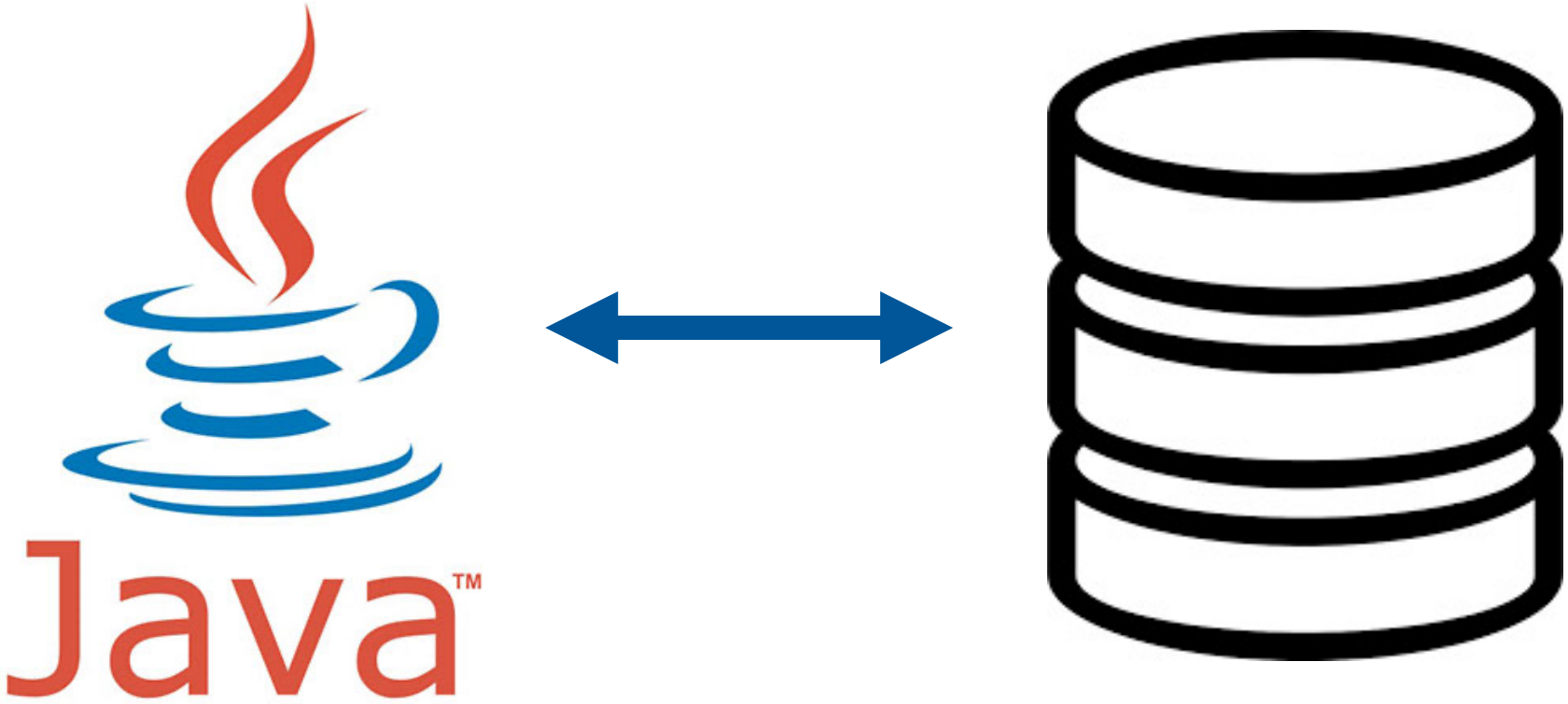
- JPA ist (nur) eine Spezifikation (keine Implementation)

- JPA Funktionalität wird durch verschiedene Provider implementiert
  - Hibernate
  - EclipseLink / TopLink
  - ObjectDB

ORM Intro

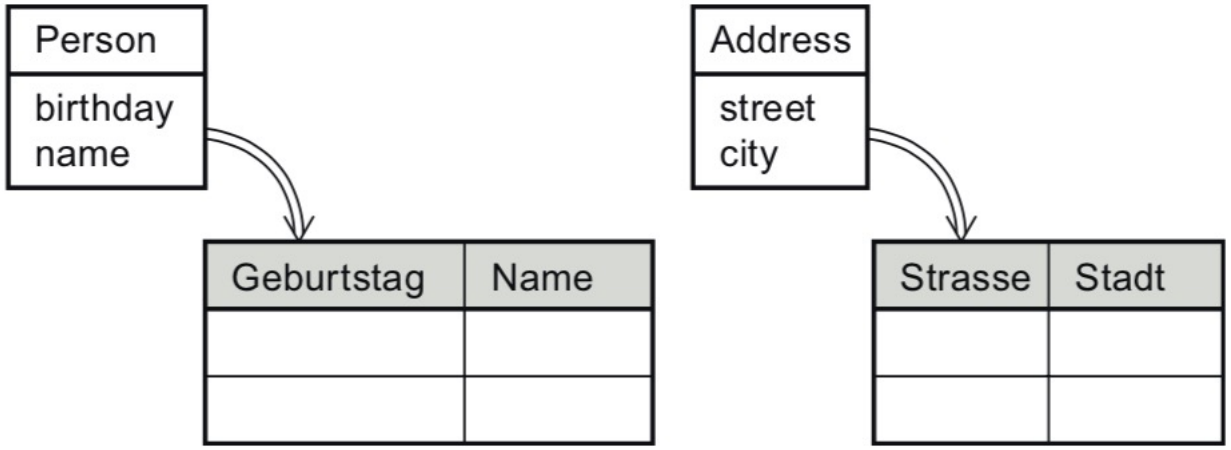
ORM = Object-Relational Mapping

- ORM = Object-Relational Mapping
- Abbildung vom objektorientierten Modell auf das Datenbankmodel



ORM = Object-Relational Mapping

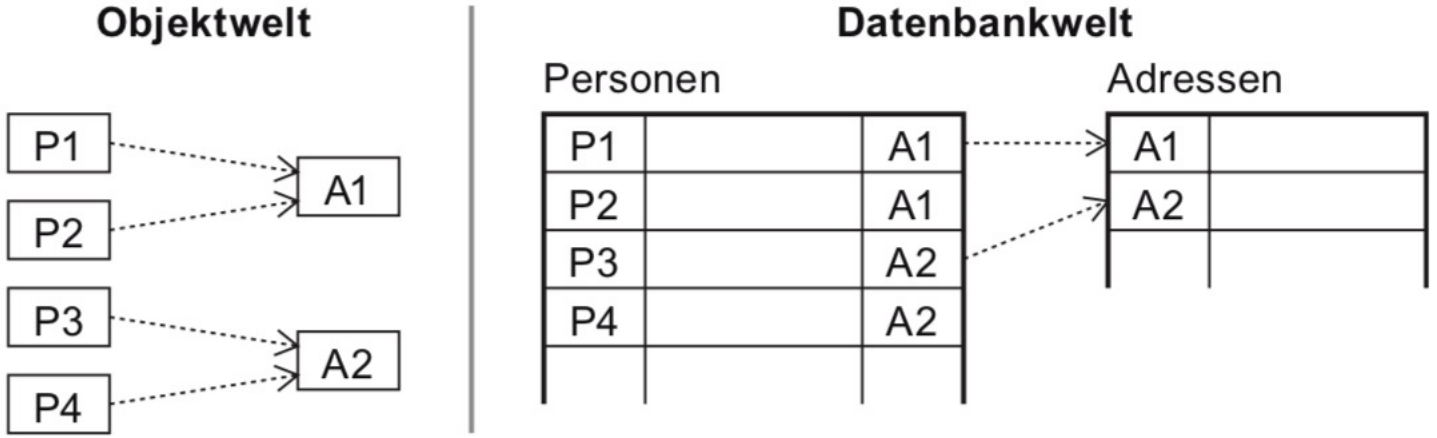
- **Objekte auf Tabellen einer Datenbank abbilden:**
  - Objekte in DB speichern



  - Objekte aus DB rekonstruieren

ORM = Object-Relational Mapping

- **Objekte auf Tabellen einer Datenbank abbilden:**
  - Beziehungen



  - Vererbung

- **ORM lässt sich mit etwas Mühe selbst programmieren.**
- **Wird aber doch mitunter komplex und herausfordernd (vor allem Assoziationen und Vererbung)**

ORM = Object-Relational Mapping

- **ORM mit «purem» JDBC ist recht aufwendig**
  - umso komplexer, aufwendiger und fehleranfälliger, je verzweigter der Objektgraph und je mehr Objekte und Assoziationen sowie Vererbungsbeziehungen existieren.
- **JPA erleichtert dies ungemein**
  - SQL-Anweisungen werden durch JPA automatisch generiert
  - Manchmal sind die generierten SQL-Kommandos suboptimal
  - Performance leidet, wenn viele referenzierte Daten verwaltet werden
- **Convention over Configuration**
  - Entitäten sind Java-Klassen mit Attributen und Annotations
  - In der Datenbank gibt es korrespondierende Tabellen und Spalten
  - **Implizite Abbildung erfolgt über Namen**
  - Explizite Abbildung erfolgt über Annotationen möglich (oder XML)

Beispiel Entity Variante Annotations an den Methoden

```
@Entity
@Table(name = "PersonenJPA")
public class Person implements Serializable
{
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDate birthday;

    @Id
    @GeneratedValue
    public Long getId()  // read only
    {
        return id;
    }

    @Column(name = "Vorname")
    public String getFirstName()
    {
        return firstName;
    }

    ...
}
```

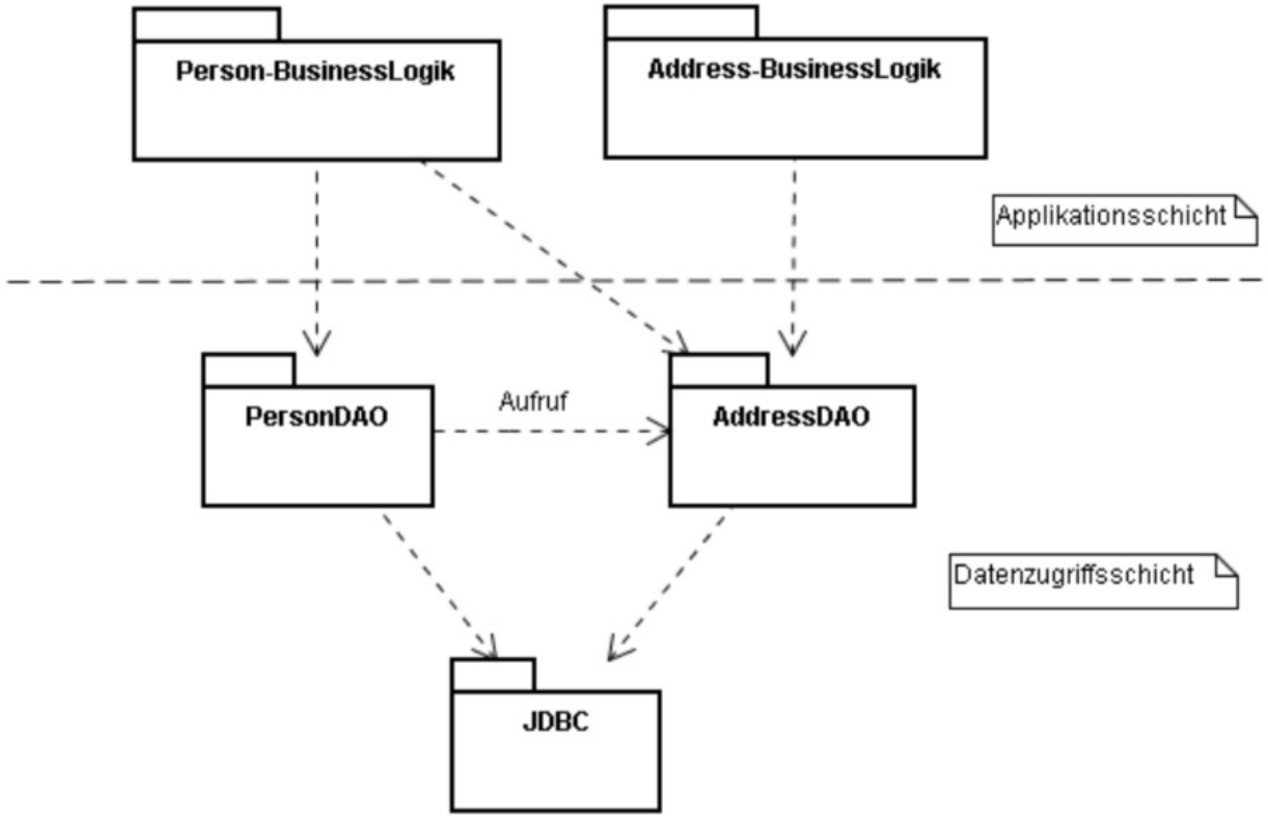
Beispiel Entity Annotations an den Attributen

```java
@Entity
@Table(name = "PersonenJPA")
public class Person implements Serializable
{
    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "Vorname")
    private String firstName;

    @Column(name = "Name")
    private String lastName;

    @Column(name = "Geburtstag")
    private LocalDate birthday;

    ...
}
```

DAOs und Repositories

DAO Pattern

- Das Data Access Object (DAO)-Muster ist ein Strukturmuster
- Es ermöglicht, die Anwendungs-/Geschäftsschicht von der Persistenzschicht mithilfe einer abstrakten API zu isolieren.



DAO Pattern

- DAO verbirgt alle Komplexitäten, die mit der Durchführung von CRUD-Operationen verbunden sind, vor der Anwendung.
- DAO führt zu loser Kopplung: Bestandteile der Applikation können sich leichter getrennt voneinander weiterentwickeln
- Eng verwandt mit dem Repository Pattern (DAO wird manchmal auch als Repository bezeichnet)
- Idealerweise erfolgen alle Datenbankzugriffe im System über ein DAO, um eine gute Kapselung zu erreichen.

DAO Pattern

- **Jede DAO-Instanz ist für eine Entität zuständig, insbesondere die CRUD-Operationen: Anlegen, Lesen (nach Primärschlüssel), Aktualisieren und Löschen (CRUD) des Domänenobjekts verantwortlich**

```
public interface IPersonDAO
{
    // CRUD-Funktionalität
    List<Person> getAllPersons() throws DataAccessException;
    long addPerson(Person person) throws DataAccessException;
    void updateFromOther(long personId, Person otherPerson) throws DataAccessException;
    void removePerson(long personId) throws DataAccessException;
}
```

- **DAO kann weitere Aktionen ermöglichen, etwa spezielle Abfragen**
- **DAO ist nicht für die Handhabung von Transaktionen, Session oder Connections verantwortlich - diese werden außerhalb der DAO behandelt**

```java
private static void executeStatements(final EntityManager entityManager)
{
    // DAO erzeugen
    final IPersonDAO dao = new PersonDAO(entityManager);

    // Einfügeoperationen ausführen und das Resultat prüfen
    final Person michael = new Person("Micha-DAO", "Inden", new Date(71, 1, 7));
    final Person michael2 = new Person("Micha-DAO", "Inden", new Date(71, 1, 7));
    final Person werner = new Person("Werner-DAO", "Inden", new Date(40, 0, 31));

    final long michaelId = dao.createPerson(michael);
    final long michaelId2 = dao.createPerson(michael2);
    final long wernerId = dao.createPerson(werner);

    final List<Person> persons2 = dao.findAllPersons();
    persons2.forEach(System.out::println);

    // Änderungen ausführen und das Resultat prüfen
    dao.deletePersonById(michaelId);
    werner.setFirstName("Dr. h.c. Werner");

    final List<Person> persons = dao.findAllPersons();
    persons.forEach(System.out::println);
}
```

Allgemeineres DAO Pattern

- **Rudimentäres Interface für einen generischen DAO**
  - **`save()` ist für «create» und «update» zuständig bzw. «update» geschieht durch Attributänderungen**

```
import java.util.List;
import java.util.Optional;

public interface Dao<T>
{
    T save(T t);

    T get(long id);
    List<T> getAll();

    void delete(T t);
}
```

```
import java.util.List;
import java.util.Optional;

public interface Dao<T>
{
    T save(T t);

    Optional<T> get(long id);
    List<T> getAll();

    void delete(T t);
}
```

Allgemeineres DAO Pattern mit JPA

- Verschiedene Typen für Keys
- Implementierung und Anbindung an `EntityManager`

```
public final class GenericDAO<T, K>
{
    private final EntityManager entityManager;
    private final Class<T> clazz;

    GenericDAO(final EntityManager entityManager, final Class<T> clazz)
    {
        this.entityManager = entityManager;
        this.clazz = clazz;
    }

    // C -- CREATE
    public T save(final T newObject)
    {
        entityManager.persist(newObject);
        return newObject;
    }
```

Allgemeineres DAO Pattern

```
// R -- READ
public T findById(final K id)
{
    return entityManager.find(clazz, id);
}

// R -- READ
public List<T> findAll()
{
   final TypedQuery<T> query = entityManager.createQuery("FROM " + clazz.getSimpleName(),
                                                          clazz);
   return query.getResultList();
}

// D -- DELETE
public void deleteById(final K id)
{
    final T objectInDb = findById(id);
    if (objectInDb != null)
    {
        entityManager.remove(objectInDb);
    }
}
```

Das ist schon gut. Wäre es nicht super, wenn man das noch kürzer und mächtiger machen könnte?

Spring Data Repositories

- Data Access Object (DAO) vereinfachen den Datenzugriff in der Regel enorm
- Besser noch sind die Spring Data Repositories, die von Hause aus bereits eine Sammlung sehr nützlicher Methoden mit bringen
- Es können ganz leicht eigene Abfragen definiert werden

```
public interface IFooDAO extends JpaRepository<Foo, Long> {

    Foo findByName(String name);

}
```

- Später mehr …

Experimente mit In-Memory-DB

H2 In-Memory-DB

- H2 ist eine der beliebtesten In-Memory-Datenbanken. Spring Boot bietet eine sehr gute Integration für H2.
- H2 ist ein in Java geschriebenes relationales Datenbankmanagementsystem. Es kann in Java-Anwendungen eingebettet oder im Client-Server-Modus ausgeführt werden.
- H2 unterstützt eine (wesentliche) Teilmenge des SQL-Standards.
- H2 bietet auch eine Web-Konsole für die Verwaltung der Datenbank.
- http://www.h2database.com/html/features.html

Additional H2 Dependencies

Maven

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
</dependency>
```

Gradle

```
implementation group: 'com.h2database', name: 'h2', version: '1.4.200'
```

H2 Login

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:test

User Name: sa

Password:

Connect Test Connection

H2 Screen

Auto commit | Max rows: 1000 | Auto complete Off | Auto select On

- jdbc:h2:mem:test
  - PEOPLE
  - INFORMATION_SCHEMA
  - Users
- H2 1.4.200 (2019-10-14)

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM PEOPLE
```

Important Commands

		Displays this Help Page
		Shows the Command History
	Ctrl+Enter	Executes the current SQL statement
	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

H2 Abfrage

Auto commit | Max rows: 1000 | Auto complete Off | Auto select On

jdbc:h2:mem:test
- PEOPLE
- INFORMATION_SCHEMA
- Users

H2 1.4.200 (2019-10-14)

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM PEOPLE
```

SELECT * FROM PEOPLE;

ID	FIRST_NAME	LAST_NAME	AGE
1	Michael	Inden	50
2	Tim	Boetzmeyer	50
3	Heinz	Mustermann	32
4	James	Bond	44

(4 rows, 43 ms)

Edit

H2 InMemoryDB Interessante Links

- https://howtodoinjava.com/spring-boot2/h2-database-example/
- https://www.linkedin.com/pulse/unit-testing-using-h2-in-memory-db-raghunandan-gupta/
- https://phauer.com/2017/dont-use-in-memory-databases-tests-h2/

PART 2:
JdbcTemplate

Einführung JdbcTemplate

- JdbcTemplate kümmert sich um die Erstellung und Freigabe von Ressourcen wie das Erstellen und Schließen von Verbindungsobjekten usw.
- Also: Kein lästiges „Daraufachten“, die die Verbindung in jedem Fall wieder zu schließen
- JdbcTemplate kümmert sich auch um Exceptions und liefert informative Fehlermeldungen
- Mit Hilfe der Klasse JdbcTemplate können wir alle Datenbankoperationen wie Einfügen, Aktualisieren, Löschen und Abrufen von Daten aus der Datenbank durchführen.

```
jdbcTemplate.execute("DROP TABLE customers IF EXISTS");
jdbcTemplate.execute("CREATE TABLE customers(" + "id SERIAL, first_name VARCHAR(255),
                     last_name VARCHAR(255))");

jdbcTemplate.query("SELECT id, first_name, last_name FROM customers WHERE first_name = ?",
                   new Object[] { "Josh" },
```

- https://spring.io/guides/gs/relational-data-access/

Einführung JdbcTemplate

1. **`int update(String query)`** ermöglicht es, Datensätze einzufügen, zu aktualisieren und zu löschen.

2. **`int update(String query,Object... args)`** dient zum Einfügen, Aktualisieren und Löschen von Datensätzen mit PreparedStatement unter Verwendung der angegebenen Argumente.

3. **`void execute(String query)`** wird verwendet, um eine DDL-Abfrage auszuführen.

4. **`List query(String sql, RowMapper rm)`** wird verwendet, um Datensätze mit RowMapper aus den Daten zu extrahieren.

5. **`T query(String sql, ResultSetExtractor rse)`** dient dazu, Datensätze mit Hilfe von ResultSetExtractor abzurufen.

Beispiel: Verwaltung von Employee-Daten

```
create table EMPLOYEE (
    id integer not null primary key,
    name varchar(20) not null,
    salary float not null
);
```

```
public class Employee {
    private int id;
    private String name;
    private float salary;

    public Employee() {
    }

    public Employee(int id, String name, float salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
```

…

DAO mit JdbcTemplate

```
public class EmployeeDao {
    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    public int saveEmployee(Employee e) {
        String sql = "insert into employee values('" + e.getId() + "','" +
                     e.getName() + "','" + e.getSalary() + "')";
        return jdbcTemplate.update(sql);
    }

    public int getEmployeeCount() {
         String sql = "select count(*) from employee";
         return jdbcTemplate.queryForObject(sql, Integer.class, new Object[] {});
    }

    …
```

DAO mit JdbcTemplate

```
    …

    public int deleteEmployee(Employee e) {
        String sql = "delete from employee where id='" + e.getId() + "' ";
        return jdbcTemplate.update(sql);
    }

    // Erweiterungen
    public Employee getEmployeeById(int id) {
        // Fieser Fehler: String sql = "select * from employee where id = " + id;
        String sql = "select * from employee where id = ?";
        return jdbcTemplate.queryForObject(sql, new EmployeeRowMapper(),
                                                new Object[] { id });
    }

    public List<Employee> getAllEmployees() {
        String sql = "select * from employee";
        return jdbcTemplate.query(sql, new EmployeeRowMapper());
    }
}
```

Datenextraktion mit RowMapper

```
public class EmployeeRowMapper implements RowMapper<Employee> {
    @Override
    public Employee mapRow(ResultSet resultSet, int i) throws SQLException
    {
        Employee person = new Employee();

        person.setId(resultSet.getInt(1));
        person.setName(resultSet.getString(2));
        person.setSalary(resultSet.getFloat(3));

        return person;
        // return new Employee(rs.getInt(1), rs.getString(2), rs.getFloat(3));
    }
}
```

DAO im EInsatz

```
var ctx = new ClassPathXmlApplicationContext("applicationContext.xml");

EmployeeDao dao = (EmployeeDao) ctx.getBean("employeeDao");
System.out.println("employee count: " + dao.getEmployeeCount());

int status = dao.saveEmployee(new Employee(4711, "Michael", 12345.0f));
System.out.println(status);
System.out.println("employee count: " + dao.getEmployeeCount());

System.out.println(dao.getAllEmployees());
System.out.println(dao.getEmployeeById(1));
System.out.println(dao.getEmployeeById(4711));
```

Konfiguration in Plain Spring für JdbcTemplate

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …

<jdbc:initialize-database data-source="dataSource">
<jdbc:script location="classpath:scripts/schema.sql" />
<jdbc:script location="classpath:scripts/data-h2.sql" />
</jdbc:initialize-database>

    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="org.h2.Driver" />
        <property name="url" value="jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-1" />
        <property name="username" value="sa" />
        <property name="password" value="" />
    </bean>

    <bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"></property>
    </bean>

    <bean id="employeeDao" class="springjdbc.dao.EmployeeDao">
        <property name="jdbcTemplate" ref="jdbcTemplate"></property>
    </bean>
</beans>
```

DEMO

»spring-jdbc-intro»
» spring-jdbc-with-spring-boot-1»
»spring-jdbc-with-spring-boot-2-web»

Das ist schon eine gute Hilfe, aber: Was ist daran unschön?

Part 3:
Spring Data

- Einführung
- Spring Data JPA Grundlagen
- Spring Data Repositories
- Spring Data Mongo DB

Einführung

Spring Data

- provide a **familiar** and **consistent**, Spring-based programming model for **data access**
- makes it **easy** to use **relational** and **non-relational databases**, and **cloud-based data** services.
- **umbrella project** which contains **many subprojects** that are specific to a given database.

http://projects.spring.io/spring-data/

Spring Data Main Modules

- **Spring Data Commons** - Core Spring concepts underpinning every Spring Data project.
- **Spring Data JPA** - Makes it easy to implement JPA-based repositories.
- **Spring Data MongoDB** - Spring based, object-document support and repositories for MongoDB.
- …

Gleich und doch verschieden

JPA

```
@Entity
@Table(name="TUSR")
public class User {

  @Id
  private String id;

  @Column(name="fn")
  private String name;

  private Date lastLogin;

...
}
```

MongoDB

```
@Document(
collection="usr")
public class User {

  @Id
  private String id;

  @Field("fn")
  private String name;

  private Date lastLogin;

 ...
}
```

Neo4j

```
@NodeEntity
public class User {

  @GraphId
  Long id;


  private String name;

  private Date lastLogin;

...
}
```

Spring Data Grundlagen
JPA Example

Getting Started — Maven Dependencies

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.6</version>
</parent>


<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
...
```

Getting Started — Gradle Dependencies

```
plugins {
  id "org.springframework.boot" version "2.5.6"
}

apply plugin: 'java'
apply plugin: 'eclipse'


repositories {
    mavenCentral()
}

sourceCompatibility = 11
targetCompatibility = 11

dependencies {

    implementation 'org.springframework.boot:spring-boot-starter-data-jpa:2.5.6'
    testImplementation 'org.springframework.boot:spring-boot-starter-test:2.5.6'
```

First Spring Boot Application Example

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApp {

   public static void main(String[] args) {
      SpringApplication.run(MyApp.class, args);
   }
}
```

First Entity Example

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class SimpleEmployee
{
    @Id
    @GeneratedValue
    private Long id;
    private String firstName, lastName, description;

    private SimpleEmployee()
    {
    }

    public SimpleEmployee(String firstName, String lastName, String description)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.description = description;
    }
```

...

First Repository Example

- Datenbankabfragen folgen dem **DAO Pattern**
- Diese sind durch sogenannte **Repositories** beschrieben

- In Spring sind das einfache **Interfaces (POJI)** => **Deklarative Programmierung**

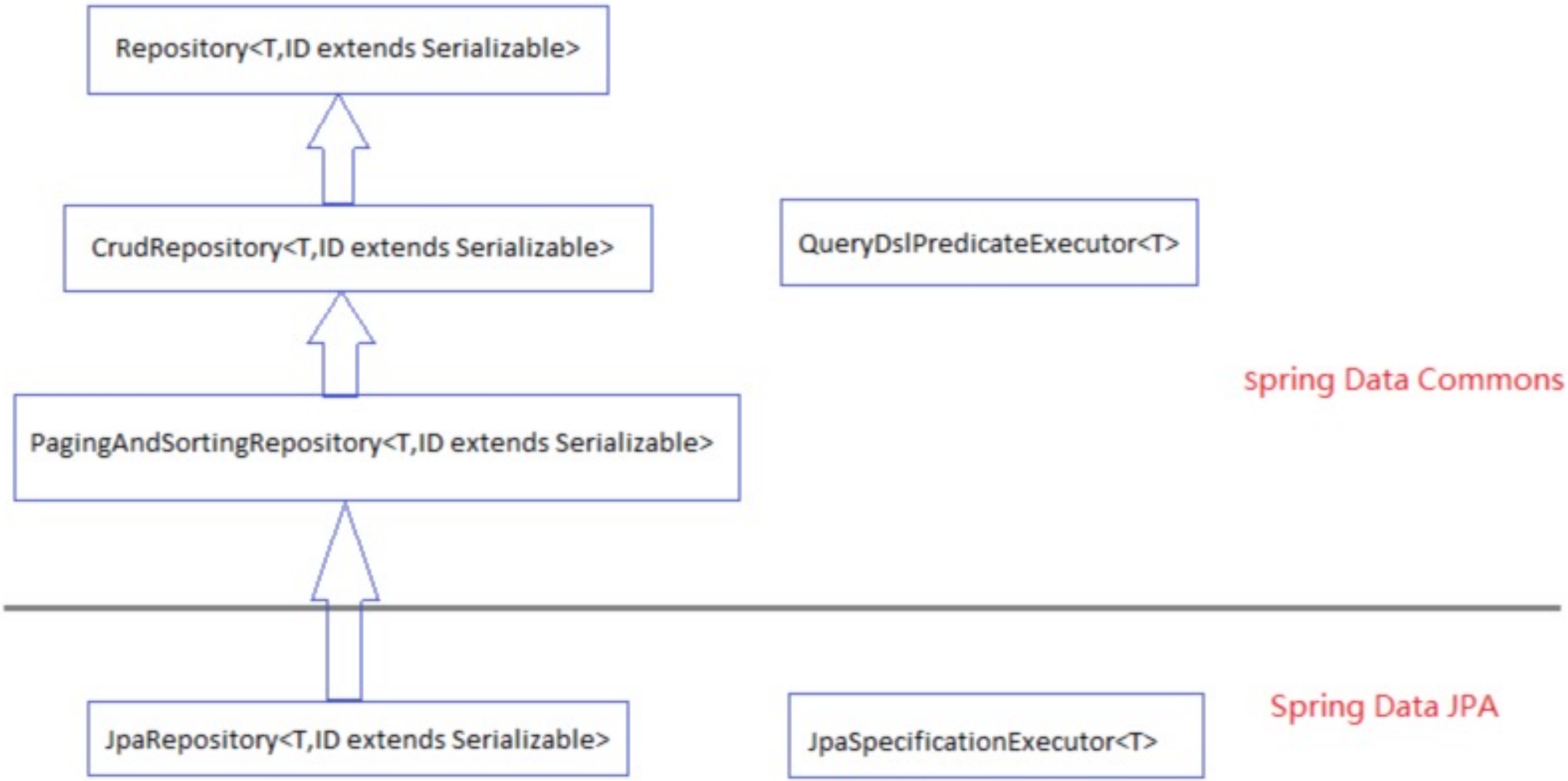
```
import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface SimpleEmployeeRepository extends CrudRepository<SimpleEmployee, Long>
{
    SimpleEmployee findByFirstName(String firstName);

    List<SimpleEmployee> findByLastName(String lastName);
}
```

- **Datenbankzugriffe werden basierend auf `findXyz()` automatisch generiert**

Basis Spring CRUD Repository — Standardfunktionalität

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // …
}
```

Basis Spring Repositories



First Example

Starten wir einfach mal …

```
@SpringBootApplication
public class MyApp {

   public static void main(String[] args) {
      SpringApplication.run(MyApp.class, args);
   }
}
```

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Cannot determine embedded database driver class for database type NONE
```

Wie binden wir eine DB ein?

H2 und Spring Boot

- Die Konfiguration der H2-Datenbank mit Spring Boot ist sehr einfach: **Einfach die H2-Abhängigkeit zur POM hinzufügen:**

- Spring Boot erstellt automatisch die Datenbank, richtet alle JDBC-Objekte der Datenbank ein und konfiguriert Hibernate standardmäßig in einem Create-Drop-Modus.

- Wenn Hibernate startet, scannt es die JPA-kommentierten Klassen und generiert automatisch den SQL-Code, der für die Erstellung der Datenbanktabellen erforderlich ist, und führt ihn aus.

RECAP: Additional H2 Dependencies

Maven

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.200</version>
</dependency>
```

Gradle

```
implementation group: 'com.h2database', name: 'h2', version: '1.4.200'
```

Applikation starten

Maven

```
mvn clean package
mvn spring-boot:run
```

Gradle

```
gradle clean assemble
gradle bootRun
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.6)
```

Wie arbeiten wir mit der DB?

Datenbank befüllen — CommandLineRunner (oder Skripte)

```
@SpringBootApplication
public class Application implements CommandLineRunner
{
   @Autowired
   private SimpleEmployeeRepository repository;

   public static void main(String[] args)
   {
      SpringApplication.run(Application.class, args);
   }

   public void run(String... args) throws Exception
   {
      …
   }
}
```

Datenbank befüllen — CommandLineRunner

```
public void run(String... args) throws Exception
{
   Employee emp1 = new Employee("Michael", "Inden", "Team Lead");
   Employee emp2 = new Employee("Karthi", "Bollu Ganesh", "Lead Engineer");
   Employee emp3 = new Employee("Marcello", "Fluri", "Senior SW Engineer");

   System.out.println("Employees: " + repository.count());
   repository.save(emp1);
   repository.save(emp2);
   repository.save(emp3);
   System.out.println("Employees: " + repository.count());
   System.out.println("Employees: " + repository.findAll());

   // Find + Delete
   repository.delete(repository.findByFirstName("Marcello"));
   System.out.println("Employees: " + repository.count());
   System.out.println("Employees: " + repository.findAll());
}
```

Datenbank befüllen — CommandLineRunner

```
Employees: 0
Employees: 3
Employees: [SimpleEmployee [id=6, firstName=Michael, lastName=Inden, description=Team Lead],
SimpleEmployee [id=7, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer],
SimpleEmployee [id=8, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer]]
Employees: 2
Employees: [SimpleEmployee [id=6, firstName=Michael, lastName=Inden, description=Team Lead],
SimpleEmployee [id=7, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer]]
```

Spring Data Repositories

Spring Data Repositories: Abfrage-Varianten über Methodennamen

```
public interface MovieRepository extends JpaRepository<Movie, Long>
{
    List<Movie> findByTitleIgnoringCase(String title);
}
```

- findBy, readBy, getBy, countBy, queryBy
- GreaterThan, LessThan, Between
- Like, In

- Sortierung: OrderBy…Asc / Desc
- Eindeutigkeit: Distinct
- Einschränkungen / Paging: Top / First, etwa Top10

Mögliche Varianten

- **And, Or**
  - findByLastnameAndFirstname() / findByLastnameOrFirstname()
  - … where x.lastname = ?1 and (or) x.firstname = ?2
- **Is, Equals**

  findByFirstnameIs() / findByFirstnameEquals() / findByFirstname()

  … where x.firstname = ?1
- **Between**
  - findByStartDateBetween()

    … where x.startDate between ?1 and ?2
- **LessThan, GreaterThan**
  - findByAgeLessThan() / findByAgeGreaterThan()

    … where x.age < ?1 / … where x.age > ?1

Mögliche Varianten

- After, Before
- IsNull, IsNotNull, NotNull
- Like / NotLike
- Containing
- OrderBy
- True / False
- In / NotIn
- Not
- IgnoreCase
- Asc / Desc

- **Begrenzung der Ergebnisgröße einer Abfrage**
  - findFirst10ByLastnameAsc

Schlüsselwörter

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1

Repository Example

```
public interface EmployeeRepository extends CrudRepository<Employee, Long>
{
   Employee findByFirstName(String firstName);
   List<Employee> findByLastName(String lastName);

   List<Employee> findByAgeGreaterThan(int age);
   List<Employee> findByAgeBetween(int lower, int upper);
   int countByAgeBetween(int lower, int upper);

   List<Employee> findTop3ByAgeLessThan(int maxAge);
   List<Employee> findByAgeLessThanOrderByFirstNameAsc(int maxAge);

   @Query("SELECT emp FROM Employee emp WHERE emp.firstName LIKE %?1%")
   List<Employee> getFirstNameLike(String firstName);

   List<Employee> findByFirstNameOrLastName(String firstName, String lastName);

   List<Employee> findByLastNameInAndAgeBetween(Collection<String> names,
                                                int lower, int upper);
}
```

Repository Example

```
Employee emp1 = new Employee("Michael", "Inden", "Team Lead", 47);
Employee emp2 = new Employee("Karthi", "Bollu Ganesh", "Lead Engineer", 33);
Employee emp3 = new Employee("Marcello", "Fluri", "Senior SW Engineer", 52);
Employee emp4 = new Employee("Marco", "Sonderegger", "SW Engineer", 30);
Employee emp5 = new Employee("Numa", "Trezzini", "SW Engineer", 30);
Employee emp6 = new Employee("Martin", "Dorta", "Senior SW Engineer", 50);

employeeRepository.save(emp1);
employeeRepository.save(emp2);
employeeRepository.save(emp3);
employeeRepository.save(emp4);
employeeRepository.save(emp5);
employeeRepository.save(emp6);
```

Repository Example

```
System.out.println("Employees 40-50: " + employeeRepository.findByAgeBetween(40, 50));
System.out.println("#Employees 40-50: " + employeeRepository.countByAgeBetween(40, 50));

System.out.println("Employees > 40: " + employeeRepository.findByAgeGreaterThan(40));
System.out.println("Employees < 50 Top 3: " + employeeRepository.findTop3ByAgeLessThan(50));
System.out.println("Employess: " + employeeRepository.findByAgeLessThanOrderByFirstNameAsc(35));
System.out.println("Employees: " + employeeRepository.getFirstNameLike("Ma"));
System.out.println("Employees: " + employeeRepository.findByFirstNameOrLastName("Michael",
                                                                                "Fluri"));
```

Repository Example

```
System.out.println("Employees 40-50: " + repository.findByAgeBetween(40, 50));
System.out.println("#Employees 40-50: " + repository.countByAgeBetween(40, 50));

System.out.println("Employees > 40: " + repository.findByAgeGreaterThan(40));
System.out.println("Employees < 50 Top 3: " + repository.findTop3ByAgeLessThan(50));
System.out.println("Employees: " + repository.findByAgeLessThanOrderByFirstNameAsc(35));

System.out.println("Employees: " + repository.getFirstNameLike("Ma"));
```

```
Employees 40-50: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                  Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
#Employees 40-50: 2
Employees > 40: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                 Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
                 Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                       Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
                       Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employess: [Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=5, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees: [Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
```

DEMO

«spring-data-slides-examples»

Spring Data Repositories

- **Mit Spring Data Repositories können nicht nur eigene Abfragen basierend auf Methodenname, sondern auch spezielle SQL-artige definiert werden:**

```
@Query("SELECT u FROM User u WHERE u.status = 1")
Collection<User> findAllActiveUsers();
```

```
@Query("SELECT u FROM User u WHERE u.status = ?1 and u.name = ?2")
User findUserByStatusAndName(Integer status, String name);
```

```
@Query(
  value = "SELECT * FROM USERS u WHERE u.status = 1",
  nativeQuery = true)
Collection<User> findAllActiveUsersNative();
```

Spring Data Repositories

- **Weitere Möglichkeiten**

```
@Query(value = “SELECT u FROM User u WHERE u.name IN :names”)
List<User> findUserByNameList(@Param(“names”) Collection<String> names);
```

```
@Modifying
@Query(“update User u set u.status = :status where u.name = :name”)
int updateUserSetStatusForName(@Param(“status”) Integer status,
  @Param(“name”) String name);
```

Exercises 21 – 22

https://github.com/Michaeli71/AMTC_Spring_Workshop

Spring Data MongoDB Example

Repository Example

```
import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Employee
{
   @Id
   private String id;

      …
}
```

Repository Example

```
public interface EmployeeRepository extends MongoRepository<Employee, String>
{
   Employee findByFirstName(String firstName);
   List<Employee> findByLastName(String lastName);

   List<Employee> findByAgeGreaterThan(int age);
   List<Employee> findByAgeBetween(int lower, int upper);
   int countByAgeBetween(int lower, int upper);

   List<Employee> findTop3ByAgeLessThan(int maxAge);
   List<Employee> findByAgeLessThanOrderByFirstNameAsc(int maxAge);

   List<Employee> getByFirstNameLike(String firstName);

   List<Employee> findByFirstNameOrLastName(String firstName, String lastName);

   List<Employee> findByLastNameInAndAgeBetween(Collection<String> names,
                                                int lower, int upper);
}
```

Repository Example

```
System.out.println("Employees 40-50: " + repository.findByAgeBetween(40, 50));
System.out.println("#Employees 40-50: " + repository.countByAgeBetween(40, 50));

System.out.println("Employees > 40: " + repository.findByAgeGreaterThan(40));
System.out.println("Employees < 50 Top 3: " + repository.findTop3ByAgeLessThan(50));
System.out.println("Employees: " + repository.findByAgeLessThanOrderByFirstNameAsc(35));

System.out.println("Employees: " + repository.getFirstNameLike("Ma"));
```

```
Employees 40-50: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47]]
#Employees < 50: 1
Employees > 40: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
          Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
          Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3:
         [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
          Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
          Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d0, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
            Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52]]
```

Repository Example

MongoDB: BETWEEN: lower < x < upper

```
Employees 40-50: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47]]
#Employees 40-50: 1
Employees > 40: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
         Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
         Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3:
        [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
         Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
         Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d0, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
```

JPA: BETWEEN: lower <= x <= upper

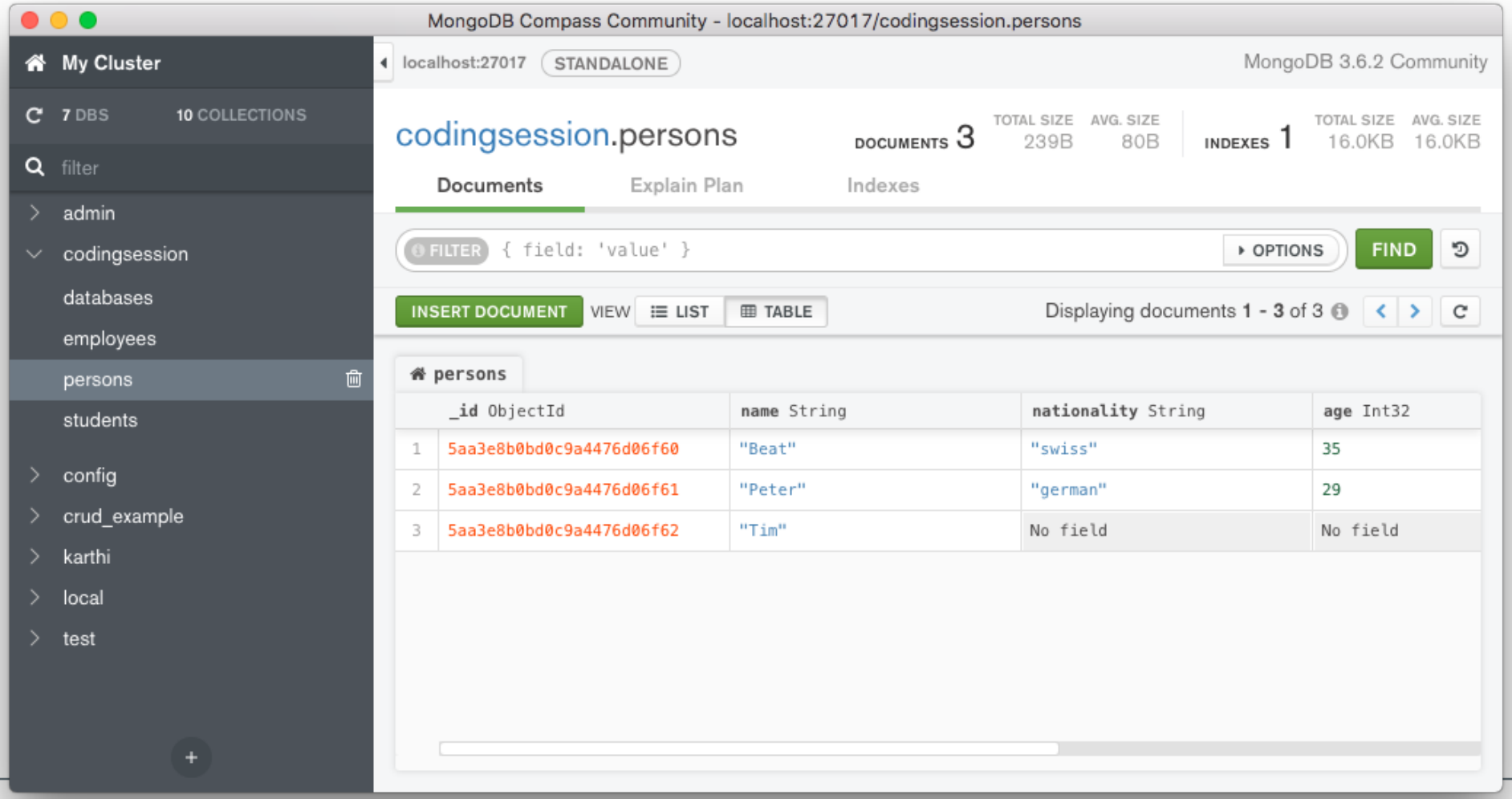
```
Employees 40-50: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                  Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
#Employees 40-50: 2
Employees > 40: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                 Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
                 Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                       Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
                       Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employees: [Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=5, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees: [Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
```

Repository Example

Names-Postfix	Operation als JSON
GreaterThan	`{ "age" : { "$gt" : <value> } }`
LessThan	`{ "age" : { "$lt" : <value> } }`
Between	`{ "age" : { "$gt" : from, "$lt" : to } }`
IsNotNull, NotNull	`{ "age" : { "$ne" : null } }`
IsNull, Null	`{ "age" : null }`
-/-	`{ "age" : <value> }`
Not	`{ "age" : { "$ne" : <value> } }`

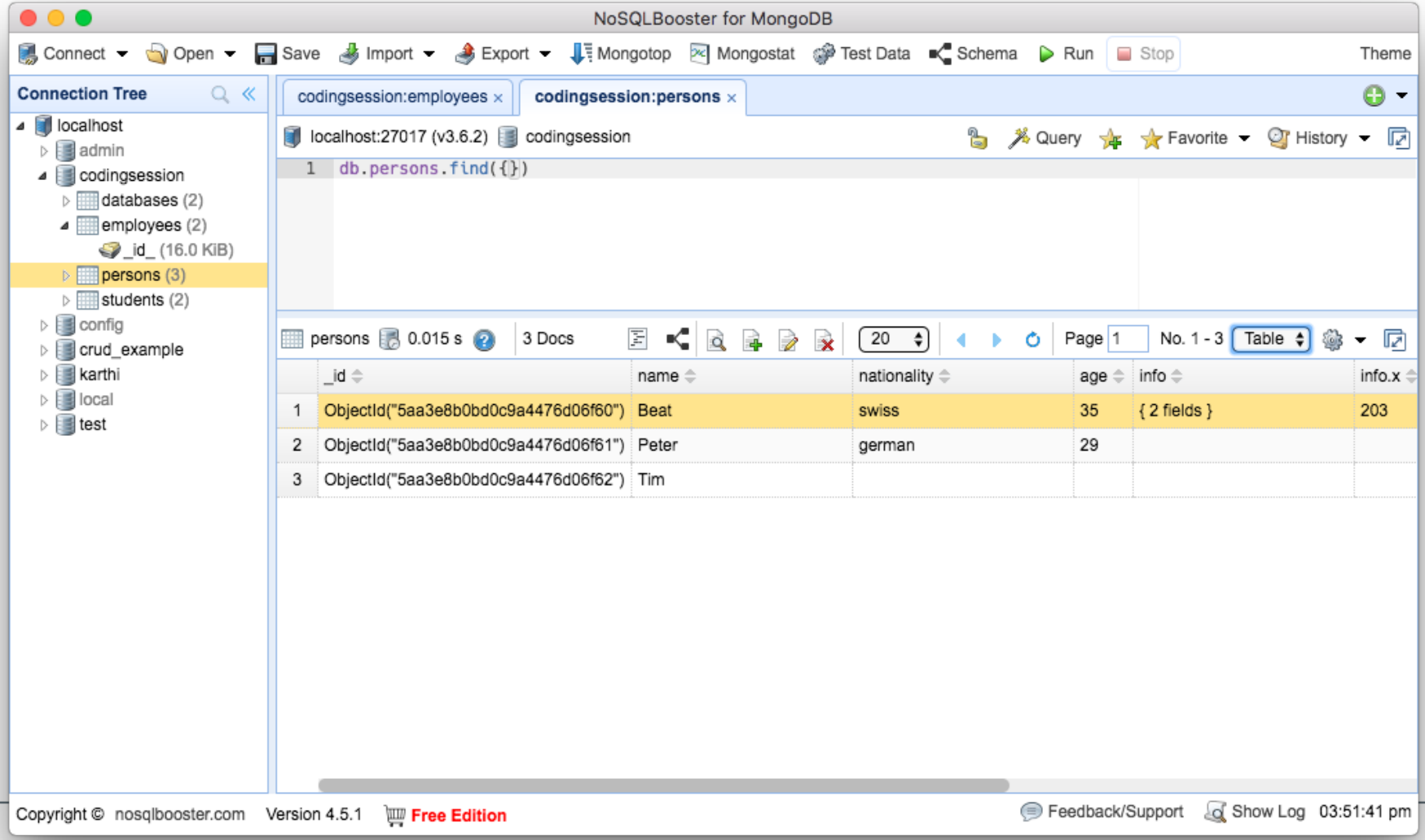
MongoDB Compass

https://docs.mongodb.com/compass/master/install/



NoSQL Booster Query Tool

https://nosqlbooster.com/downloads



Exercises 23 – 25

https://github.com/Michaeli71/AMTC_Spring_Workshop

Part 4: Validation

Dependencies

```
<dependency>
<groupId>org.hibernate.validator</groupId>
<artifactId>hibernate-validator</artifactId>
<version>6.0.22.Final</version>
</dependency>
```

Validation

```
import javax.validation.constraints.AssertTrue;
import javax.validation.constraints.Email;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class User
{
    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue
    private boolean working;

    // ...
}
```

Validation „pur“

```
public class User
{
    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue
    private boolean working;

    @Size(min = 10, max = 200,
          message = "About Me must be between 10 and 200 characters")
    private String aboutMe;

    @Min(value = 18, message = "Age should not be less than 18")
    @Max(value = 150, message = "Age should not be greater than 150")
    private int age;

    @Email(message = "Email should be valid")
    private String email;

    // standard setters and getters
}
```

Validation „pur“

```
public class ProgramaticValidationExample
{
    public static void main(final String[] args)
    {
        UserWithValidation user = new UserWithValidation();
        user.setWorking(false);
        user.setAboutMe("No info about me!");
        user.setAge(11);

        try (ValidatorFactory factory = Validation.buildDefaultValidatorFactory())
        {
            Validator validator = factory.getValidator();
            Set<ConstraintViolation<UserWithValidation>> violations = validator.validate(user);
            for (ConstraintViolation<UserWithValidation> violation : violations)
            {
                System.err.println(violation.getMessage());
            }
        }
    }
}
```

```
Name cannot be null
muss wahr sein
Age should not be less than 18
```

Validation in JPA

```
@Entity
public class UserWithValidation
{
    @Id @GeneratedValue
    private Long id;

    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue
    private boolean working;

    @Size(min = 10, max = 200,
          message = "About Me must be between 10 and 200 characters")
    private String aboutMe;


    @Min(value = 18, message = "Age should not be less than 18")
    @Max(value = 150, message = "Age should not be greater than 150")
    private int age;

    // ...
    // standard setters and getters
}
```

Validation in JPA

```
private static void executeStatements(final EntityManager entityManager)
{
    UserWithValidation user = new UserWithValidation();
    user.setWorking(false);
    user.setAboutMe("No info about me!");
    user.setAge(11);

    entityManager.persist(user);
    System.out.println(user);
}
```

```
ConstraintViolationImpl{interpolatedMessage='Age should not be less than 18',
propertyPath=age, rootBeanClass=class t_validation.UserWithValidation,
messageTemplate='Age should not be less than 18'}
ConstraintViolationImpl{interpolatedMessage='muss wahr sein', propertyPath=working,
rootBeanClass=class t_validation.UserWithValidation,
messageTemplate='{javax.validation.constraints.AssertTrue.message}'}
ConstraintViolationImpl{interpolatedMessage='Name cannot be null', propertyPath=name,
rootBeanClass=class t_validation.UserWithValidation, messageTemplate='Name cannot be
null'}
```

DEMO

ProgramaticValidationExample.java

Validation -- Annotations

• `@NotBlank`	kann nur auf Textwerte angewendet werden und validiert, dass die Eigenschaft nicht Null oder Leerzeichen ist.
• `@Positive &`	
• `@PositiveOrZero`	werden auf numerische Werte angewendet und prüfen, ob diese positiv oder positiv einschließlich 0 sind.
• `@Negative &`	
• `@NegativeOrZero`	gelten für numerische Werte und bestätigen, dass sie negativ oder negativ, einschließlich 0, sind.
• `@Past &`	
• `@PastOrPresent`	prüfen, ob ein Datumswert in der Vergangenheit oder in der Vergangenheit einschließlich der Gegenwart liegt; für alle Datumstypen einschließlich der in Java 8
`@Future &`	
• `@FutureOrPresent`	fordern, dass ein Datumswert in der Zukunft oder in der Zukunft einschließlich der Gegenwart liegt.

VALIDATION in Persistence Unit

* Anpassungen in Persistence Unit

```
<validation-mode>AUTO</validation-mode>

<validation-mode>CALLBACK</validation-mode>
```

- Fallstricke Versionen Hibernate & Hibernate-Validator sowie `javax.validation / jakarta.validation`
  - Hibernate-Validator 7.x ⇔ / `jakarta.validation`
  - Hibernate-Validator 6.x ⇔ / `javax.validation`
- Nur ältere Variante läuft in Persistence Unit sauber, ansonsten Versions- und Initialisierungsprobleme
- Standalone geht beides problemlos

Wie kann man eigene Validatoren bauen?

Eigene Validatoren in JPA

```
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = CheckEnumValidator.class)
public @interface CheckEnum
{
    String message() default "Please enter a valid enum value for this field.";
    Class<?> type();

    // für Constraint
    Class<?>[] groups() default {};
    Class<?extends Payload>[] payload() default {};
}
```

Eigene Validatoren in JPA

```
public class CheckEnumValidator implements ConstraintValidator<CheckEnum, String>
{
    Class<?> type;

    @Override
    public void initialize(CheckEnum constraintAnnotation)
    {
        type = constraintAnnotation.type();
        if (!type.isEnum())
            throw new IllegalArgumentException("type is not an enum");
    }

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context)
    {
        ...
    }
}
```

Eigene Validatoren in JPA

```
public class CheckEnumValidator implements ConstraintValidator<CheckEnum, String>
{
    ...

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context)
    {
        if (value == null)
            return true;

        Enum<?>[] enumValues = (Enum<?>[])type.getEnumConstants();
        for (Enum<?> enumValue : enumValues)
        {
            if (enumValue.name().equals(value.trim()))
                return true;
        }
        return false;
    }
}
```

Eigene Validatoren in JPA

```
public class ValidatedDomainClass
{
    @NotBlank(message = "Deposit Date is required.")
    @CheckLocalDate(dateFormat = { "yyyy-MM-dd" })
    String depositDate;

    @CheckLocalDate(dateFormat = "dd.MM.yyyy")
    String publicationDate;
    @CheckLocalDate(dateFormat = { "dd.MM.yyyy", "dd.MM.yy" })
    String collectionDate;

    // Enum-Validator für Legacy
    @CheckEnum(type = Seasons.class)
    String season;
    @CheckEnum(type = SpecialColors.class)
    String color;
    @CheckListOfValues(allowedValues = { "Anne", "Will", "Peter", "Lustig" })
    String value;
}
```

DEMO

`CustomValidatorsExample.java`

Validation in Spring

Dependencies

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

https://reflectoring.io/bean-validation-with-spring-boot/

Validation

```java
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Pattern;

public class Input {

  @Min(1)
  @Max(10)
  private int numberBetweenOneAndTen;

  @Pattern(regexp = "^[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}$")
  private String ipAddress;

  // ...
}
```

https://reflectoring.io/bean-validation-with-spring-boot/

Validation

```
@RestController
class ValidateRequestBodyController {

    @PostMapping("/validateBody")
    ResponseEntity<String> validateBody(@Valid @RequestBody Input input) {
        return ResponseEntity.ok("valid");
    }
}
```

https://reflectoring.io/bean-validation-with-spring-boot/

Validation

```
@RestController
@Validated
class ValidateParametersController {

    @GetMapping("/validatePathVariable/{id}")
    ResponseEntity<String> validatePathVariable(@PathVariable("id") @Min(5) int id) {
        return ResponseEntity.ok("valid");
    }

    @GetMapping("/validateRequestParameter")
    ResponseEntity<String> validateRequestParameter(@RequestParam("param") @Min(5) int param)
    {
        return ResponseEntity.ok("valid");
    }

    @ExceptionHandler(ConstraintViolationException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    ResponseEntity<String> handleConstraintViolationException(ConstraintViolationException e)
{
        return new ResponseEntity<>("not valid due to validation error: " + e.getMessage(),
                HttpStatus.BAD_REQUEST);
    }
}
```

https://reflectoring.io/bean-validation-with-spring-boot/

DEMO

«spring-validation-slides-examples»

DEMO / Hands On

https://spring.io/guides/gs/validating-form-input/

Exercise 26

https://github.com/Michaeli71/AMTC_Spring_Workshop

Part 5:
Mappings mit MapStruct

Mapping

```
public class Car {

    private String make;
    private int numberOfSeats;
    private CarType type;

    public Car(String make, int numberOfSeats,
               CarType type) {
        this.make = make;
        this.numberOfSeats = numberOfSeats;
        this.type = type;
    }
…
}
```

```
public class CarDto {

    private String make;
    private int seatCount;
    private String type;
```

```
public enum CarType {
    PLAIN, PICKUP, SUV, TRUCK
}
```

Sollen wir etwa jedes einzelne Attribut von Hand übertragen?

Maven Dependencies & more

Zum Einbinden und Aktivieren von MapStruct muss die POM wie folgt ergänzt werden:

```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>1.4.2.Final</version>
</dependency>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
        <release>11</release>
        <annotationProcessorPaths>
            <path>
                <groupId>org.mapstruct</groupId>
                <artifactId>mapstruct-processor</artifactId>
                <version>1.4.2.Final</version>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
```

Mapping mit MapStruct

```java
public class SimpleSource {
    private String name;
    private String description;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    …
}
```

```java
public class SimpleDestination {
    private String name;
    private String description;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    …
}
```

```java
@Mapper
public interface SimpleSourceDestinationMapper {
    SimpleDestination sourceToDestination(SimpleSource source);
    SimpleSource destinationToSource(SimpleDestination destination);
}
```

Mapping mit MapStruct

```
public class SimpleSourceDestinationMapperTest {
private SimpleSourceDestinationMapper mapper =
Mappers.getMapper(SimpleSourceDestinationMapper.class);

    @Test
    public void sourceToDestinationMapsCorrect() {
        SimpleSource simpleSource = new SimpleSource();
        simpleSource.setName("SourceName");
        simpleSource.setDescription("SourceDescription");

        SimpleDestination destination = mapper.sourceToDestination(simpleSource);

        assertEquals(simpleSource.getName(), destination.getName());
        assertEquals(simpleSource.getDescription(), destination.getDescription());
    }

    …
```

Mapping mit MapStruct

…

```
    @Test
    public void destinationToSourceMapsCorrect() {
        SimpleDestination destination = new SimpleDestination();
        destination.setName("DestinationName");
        destination.setDescription("DestinationDescription");

        SimpleSource source = mapper.destinationToSource(destination);

        assertEquals(destination.getName(), source.getName());
        assertEquals(destination.getDescription(), source.getDescription());
    }
}
```

Mapping

```
public class Car {

    private String make;
    private int numberOfSeats;
    private CarType type;

    public Car(String make, int numberOfSeats,
               CarType type) {
        this.make = make;
        this.numberOfSeats = numberOfSeats;
        this.type = type;
    }
…
}
```

```
public class CarDto {

    private String make;
    private int seatCount;
    private String type;
```

```
public enum CarType {
    PLAIN, PICKUP, SUV, TRUCK
}
```

Wie können wir steuernd eingreifen?

Mapping mit MapStruct

```
@Mapper
public interface CarMapper {

    CarMapper INSTANCE = Mappers.getMapper( CarMapper.class );

    @Mapping(source = "numberOfSeats", target = "seatCount")
    CarDto toDto(Car car);

    @Mapping(source = "seatCount", target = "numberOfSeats")
    Car toCar(CarDto dto);
}
```

Mapping mit MapStruct

```
public class CarMapperTest {

    @Test
    public void shouldMapCarToDto() {
        // given
        Car car = new Car("FORD", 5, CarType.PICKUP);

        // when
        CarDto carDto = CarMapper.INSTANCE.toDto(car);

        // then
        assertNotNull(carDto);
        assertEquals("FORD", carDto.getMake());
        assertEquals(5, carDto.getSeatCount());
        assertEquals("PICKUP", carDto.getType());
    }

    …
```

Wie nutzt man das im Kontext von Spring?

Mapping mit MapStruct

```java
@Mapper(componentModel = "spring")
public interface ProductMapper {
    ProductDTO toProductDTO(Product product);
    List<ProductDTO> toProductDTOs(List<Product> products);

    Product toProduct(ProductDTO productDTO);
}
```

```java
@Entity
public class Product {
    @Id
    @GeneratedValue
    private Long id;

    private String name;
    private String description;
    private BigDecimal price;

    private Date createdAt;
    private Date updatedAt;
```

```java
public class ProductDTO {
    private String name;
    private String description;
    private BigDecimal price;
```

Mapping mit MapStruct

```java
@RestController
@RequestMapping("/api/products")
public class ProductAPI {
    private final ProductService productService;
    private final ProductMapper productMapper;

    public ProductAPI(ProductService productService,
                      ProductMapper productMapper) {
        this.productService = productService;
        this.productMapper = productMapper;
    }

    @GetMapping
    @ResponseStatus(HttpStatus.OK)
    public List<ProductDTO> findAll() {
        List<Product> results = productService.findAll();
        return productMapper.toProductDTOs(results);
    }
```

Mapping mit MapStruct

…

```
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public ProductDTO create(@RequestBody ProductDTO productDTO) {
        Product entity = productMapper.toProduct(productDTO);
        productService.save(entity);

        return productDTO;
    }

    @GetMapping("/{id}")
    public ResponseEntity<ProductDTO> findById(@PathVariable Long id) {
        Optional<Product> optProduct = productService.findById(id);

        if (optProduct.isEmpty())
            return ResponseEntity.notFound().build();
        ProductDTO dto = productMapper.toProductDTO(optProduct.get());
        return ResponseEntity.ok(dto);
    }
```

DEMO

«spring-mapstruct-slides-examples»

Exercise 27

https://github.com/Michaeli71/AMTC_Spring_Workshop

Questions?

Hilfe

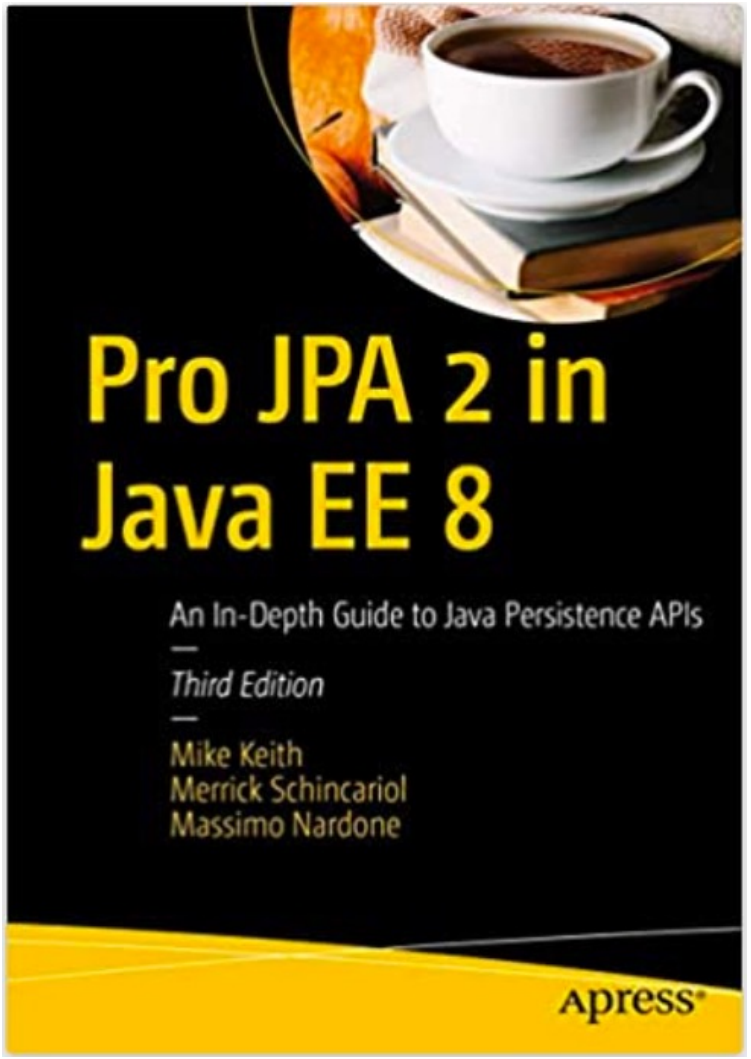

HELP
IS ON THE
WAY

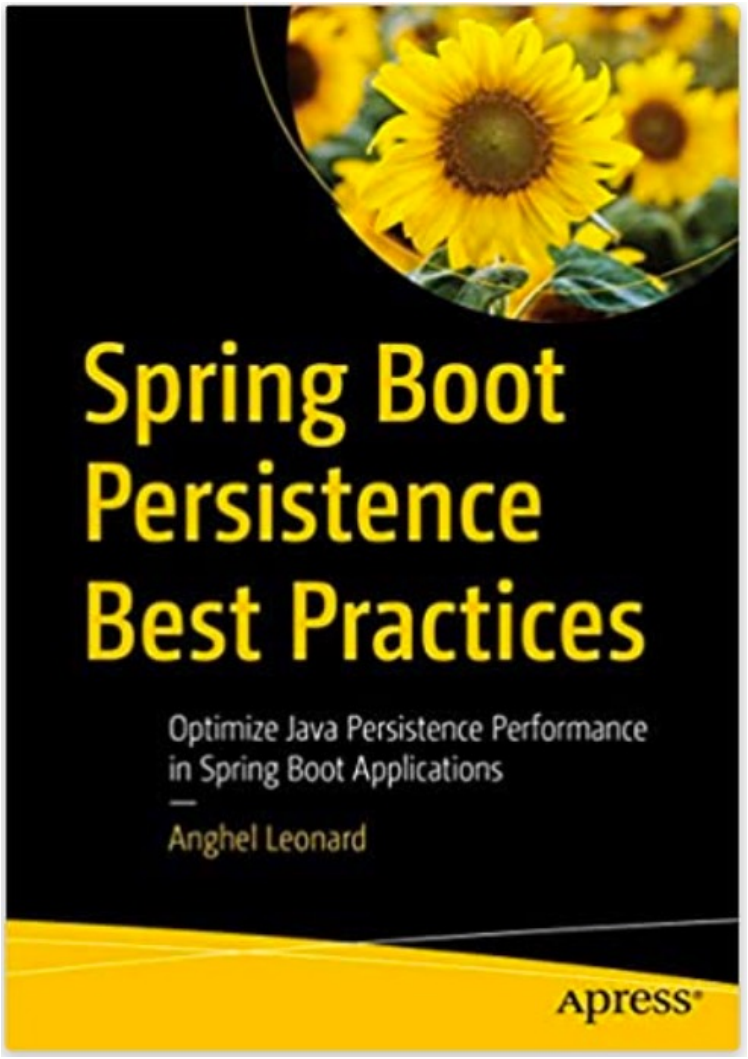

5.
Auflage
Michael Inden
Der Weg zum
Java-Profi
Konzepte und Techniken für die
professionelle Java-Entwicklung
dpunkt.verlag

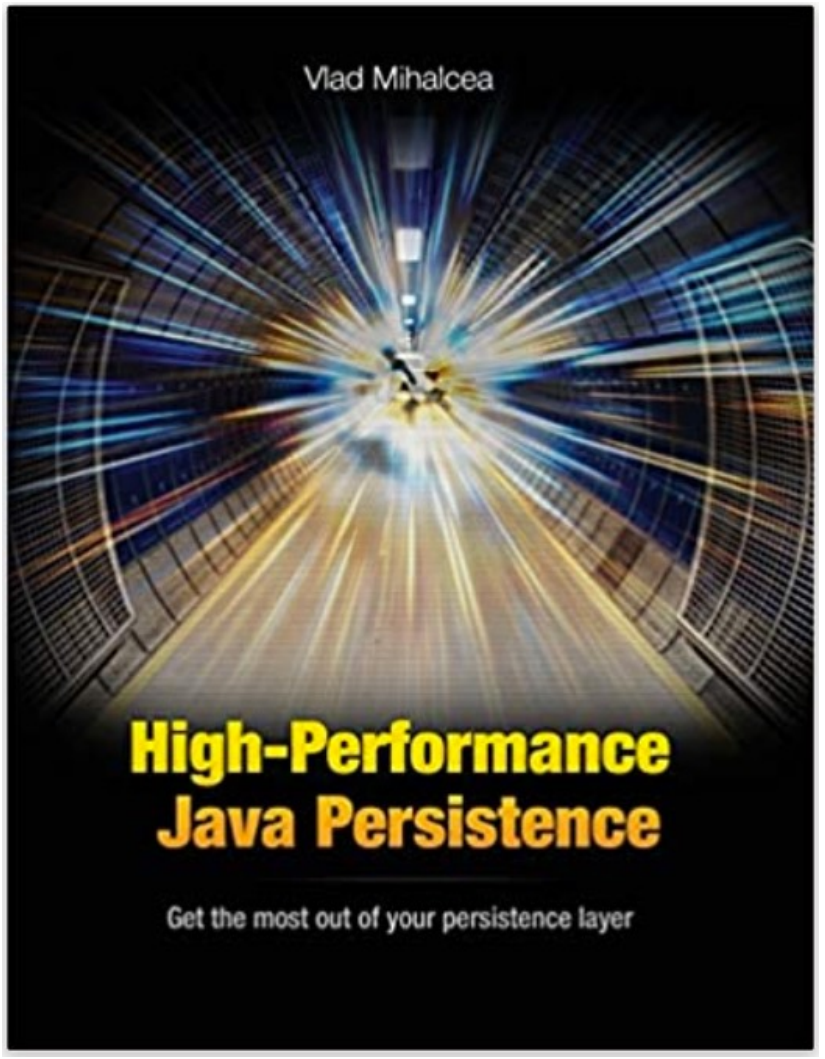

Michael Inden
Der Java-Profi:
Persistenzlösungen
und REST-Services
Datenaustauschformate,
Datenbankentwicklung
und verteilte Anwendungen
dpunkt.verlag


Java
Michael Inden
Challenge
Fit für das Job-Interview und die Praxis –
mit mehr als 100 Aufgaben
und Musterlösungen
dpunkt.verlag

Empfehlenswerte Bücher







Weitere Infos / Quellen

- **ORM**
  - https://thorben-janssen.com/jpa-generate-primary-keys/
  - https://www.objectdb.com/java/jpa/entity/generated
  - https://vladmihalcea.com/orphanremoval-jpa-hibernate/
  - https://www.baeldung.com/jpa-one-to-one
  - https://www.baeldung.com/jpa-cascade-remove-vs-orphanremoval
  - https://www.baeldung.com/hibernate-inheritance
  - https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/
  - https://www.objectdb.com/api/java/jpa/MappedSuperclass
  - https://www.logicbig.com/tutorials/java-ee-tutorial/jpa/mapped-super-class.html
  - https://vladmihalcea.com/the-best-way-to-map-a-onetoone-relationship-with-jpa-and-hibernate/
  - https://www.baeldung.com/jpa-many-to-many
  - https://vladmihalcea.com/the-best-way-to-use-the-manytomany-annotation-with-jpa-and-hibernate/
  - https://stackabuse.com/a-guide-to-jpa-with-hibernate-relationship-mapping/
  - https://thorben-janssen.com/best-practices-for-many-to-many-associations-with-hibernate-and-jpa/

Weitere Infos / Quellen

- **Validation**
  - https://www.baeldung.com/javax-validation
  - https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/

- **MapStruct**
  - https://mapstruct.org/
  - https://www.baeldung.com/mapstruct
  - https://stackabuse.com/guide-to-mapstruct-in-java-advanced-mapping-library/
  - https://www.tutorialspoint.com/mapstruct/index.htm
  - https://auth0.com/blog/how-to-automatically-map-jpa-entities-into-dtos-in-spring-boot-using-mapstruct/
  - https://www.jug.ch/events/slides/190827_Get_smart_with_MapStruct.pdf
  - https://hellokoding.com/mapping-jpa-hibernate-entity-and-dto-with-mapstruct/

Thank You