JAVA IS BACK

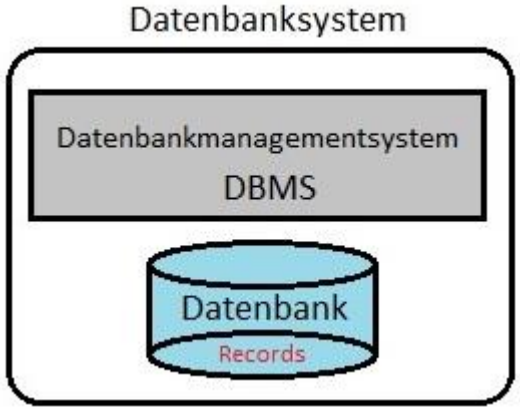
Workshop 27.01.2015 – 29.01.2015

EINFÜHRUNG

- Tag 2:
  - Spring Jdbc
  - Spring Data und JPA
  - Spring Data und NoSQL
  - REST mit Spring

SPRING JDBC

- Relationale Datenbank
  - *Datenbank*: elektronisches Verwaltungssystem, das besonders mit großen Datenmengen effizient, widerspruchsfrei und dauerhaft umgehen muss und logische Zusammenhänge digital abbilden kann.
  - Datenbanksytem
    - Datenbank: persistente Daten
    - Datenbankmanagementsystem: Software zur Verwaltung der Daten wie Zugriff und Speicherung

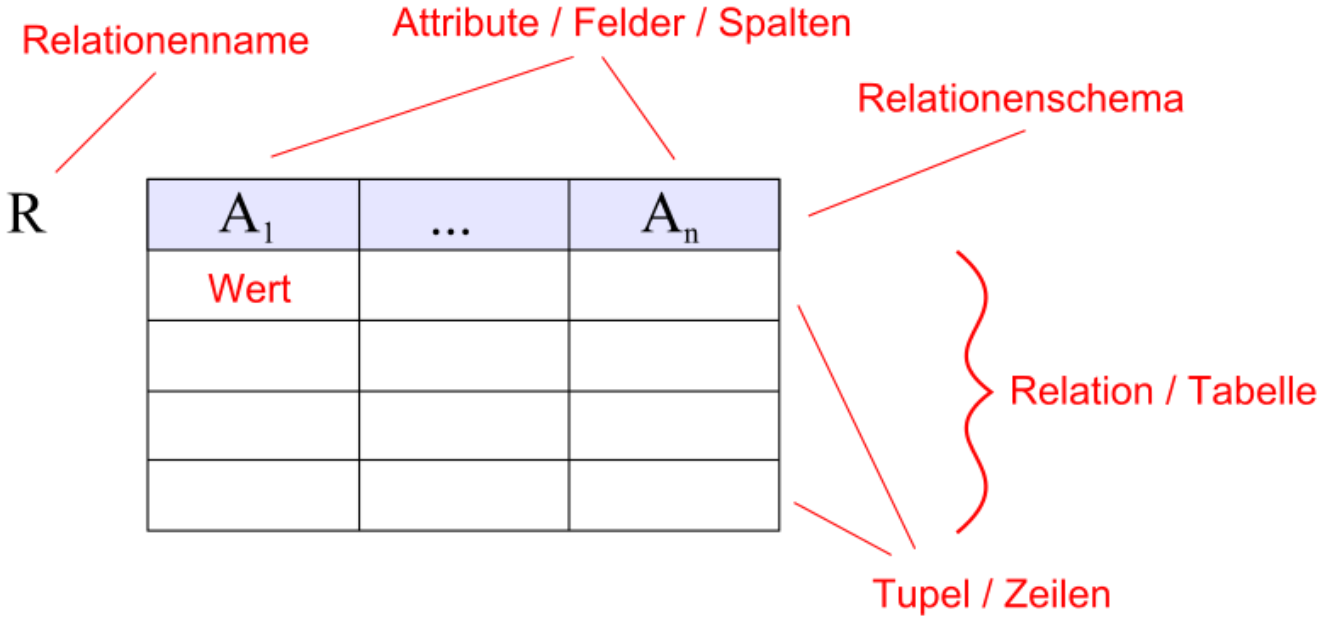


SPRING JDBC

- Relationale Datenbank
  - aktuell weitverbreitetste Datenbankmodell
  - 1970 von Edgar F. Cood
  - Relation: mathematische Beschreibung einer Tabelle und ihre Beziehung zu anderen möglichen Tabellen
  - Relationale Algebra
    - Mengenoperationen
      - Vereinigung - $M \cup N := \{\rho \mid \rho \in M \vee \rho \in N\}$
      - Schnittmenge
      - Differenz
      - Selektion
      - Projektion
      - …

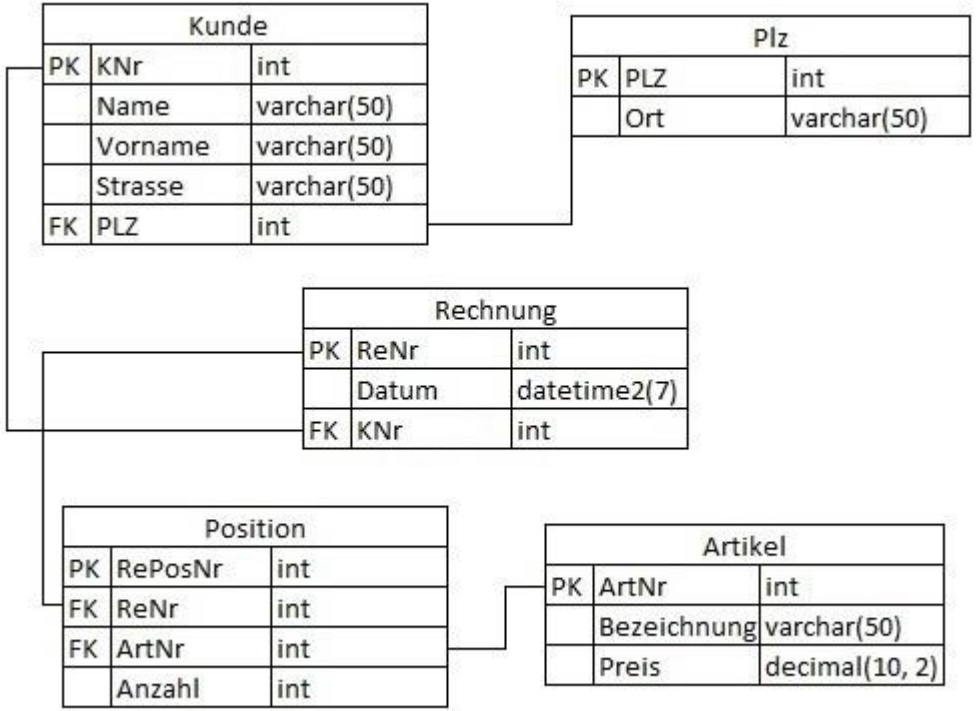
SPRING JDBC

- Relationale Datenbank
  - Relation – Tabelle



SPRING JDBC

- Relationale Datenbank
  - Primärschlüssel
  - Beziehungen zu anderen Tabellen
  - PK := Primary Key
  - FK := Foreign Key



SPRING JDBC

- Relationale Datenbank
  - Referentielle Integrität
  - Redundanz: Anomalie – semantisch identisches Attribut in mehreren Tabellen
  - Normalisierung: Konsistenzerhöhung durch Redundanzvermeidung
  - Normalformen
  - Erste Normalform (1NF)

    Jedes Attribut der Relation muss einen atomaren Wertebereich haben, und die Relation muss frei von Wiederholungsgruppen sein.
    - Abfragen, Sortierungen und Gruppierungen werden möglich
    - Atomarität von Attributen

SPRING JDBC

- Relationale Datenbank
  - 1-NF verletzt

D_ID	Album	Jahr der Gründung	Titelliste
4711	Anastacia – Not That Kind	1999	{1. Not That Kind, 2. I'm Outta Love, 3. Cowboys & Kisses}
4712	Pink Floyd – Wish You Were Here	1964	{1. Shine On You Crazy Diamond}
4713	Anastacia – Freak of Nature	1999	{1. Paid my Dues}

SPRING JDBC

CD_ID	Albumtitel	Interpret	Jahr der Gründung	*Track*	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

SPRING JDBC

- Relationale Datenbank
  - Zweite Normalform (2NF)

    Eine Relation ist in der zweiten Normalform, wenn die erste Normalform vorliegt und kein Nichtschlüsselattribut funktional abhängig von einer echten Teilmenge eines Schlüsselkandidaten ist.
    - Jede Relation modelliert nur **einen** Sachverhalt
    - Starke Kohäsion
    - Vermeidung von Redundanz

SPRING JDBC

CD_ID	Albumtitel	Interpret	Jahr der Gründung	*Track*	Titel
4711	Not That Kind	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

SPRING JDBC

CD_ID	Albumtitel	Interpret	Jahr der Gründung	*Track*	Titel
4711	**I don't mind**	Anastacia	1999	1	Not That Kind
4711	Not That Kind	Anastacia	1999	2	I'm Outta Love
4711	Not That Kind	Anastacia	1999	3	Cowboys & Kisses
4712	Wish You Were Here	Pink Floyd	1964	1	Shine On You Crazy Diamond
4713	Freak of Nature	Anastacia	1999	1	Paid my Dues

SPRING JDBC

CD_ID	Albumtitel	Interpret	Jahr der Gründung
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1964
4713	Freak of Nature	Anastacia	1999

CD_ID	*Track*	Titel
4711	1	Not That Kind
4711	2	I'm Outta Love
4711	3	Cowboys & Kisses
4712	1	Shine On You Crazy Diamond
4713	1	Paid my Dues

SPRING JDBC

- Relationale Datenbank
  - Dritte Normalform (3NF)

    *Die dritte Normalform ist genau dann erreicht, wenn sich das Relationenschema in 2NF befindet, und **kein** Nichtschlüsselattribut von einem anderen Nichtschlüsselattribut funktional abhängig ist.*

    - Transitive Abhängigkeiten

CD_ID	Albumtitel	Interpret	Jahr der Gründung
4711	Not That Kind	Anastacia	1999
4712	Wish You Were Here	Pink Floyd	1964
4713	Freak of Nature	Anastacia	1999

SPRING JDBC

ForeignKey FK

CD_ID	Albumtitel	Interpret_ID
4711	Not That Kind	**311**
4712	Wish You Were Here	**312**
4713	Freak of Nature	**311**

Interpret_ID	Interpret	Jahr der Gründung
311	Anastacia	1999
312	Pink Floyd	1964

SPRING JDBC

- Relationale Datenbank
  - SQL
    - 1979: *SQL* in *Oracle V2*
    - Name nach Vorgänger SEQUEL
    - Datenbanksprache zur Definition von Datenstrukturen
    - ISO – SQL:2011: ISO/IEC 9075-1:2011
  - HSQLDB http://hsqldb.org
    - 100% Java Database
    - SQL:2011 core language, einige SQL:2011 optional features

SPRING JDBC

- Relationale Datenbank
  - DDL: Befehle zur Definition des Datenbankschemas

```
create table CUSTOMER (
  id BIGINT GENERATED BY DEFAULT AS IDENTITY (START WITH 1) PRIMARY KEY,
  first_name VARCHAR(255) NOT NULL,
  last_name VARCHAR(255) NOT NULL,
  normalized VARCHAR(512) NOT NULL,
  email_address VARCHAR(255) NOT NULL
)


CREATE UNIQUE INDEX ix_customer_email ON CUSTOMER (email_address ASC)
CREATE INDEX ix_customer_normalized ON CUSTOMER (normalized ASC)
```

SPRING JDBC

- Relationale Datenbank
  - DML: Befehle zur Datenmanipulation
    - C – Create
    - R – Read
    - U – Update
    - D – Delete

```
insert into CUSTOMER(first_name, last_name, normalized, email_address)
values('firstname', 'lastname', 'lastname_firstname', 'test1@test.de');
select first_name, last_name from CUSTOMER where id = 1234;
update CUSTOMER set first_name = 'first_name_updated' where id = 1234;
delete from CUSTOMER where id = 1234;
```

SPRING JDBC

- Relationale Datenbank
  - Transaktionen
    - Mehrere DML-Änderungen werden zusammenausgeführt
    - Commit – Transaktion wird erfolgreich ausgeführt
    - Rollback – alle Änderungen einer Transaktion werden rückgängig gemacht

    - A – Atomar
    - C – Consistent
    - I – Isolated
    - D – Durable

SPRING JDBC

- Relationale Datenbank
  - Transaktionsanomalien
  - **Dirty Read**: Daten einer noch nicht abgeschlossenen Transaktion werden von einer anderen Transaktion gelesen.
  - **Lost Updates**: Zwei Transaktionen modifizieren parallel denselben Datensatz und nach Ablauf dieser beiden Transaktionen wird nur die Änderung von einer von ihnen übernommen.
  - **Non-Repeatable Read**: Wiederholte Lesevorgänge liefern unterschiedliche Ergebnisse.
  - **Phantom Read**: Suchkriterien treffen während einer Transaktion T1 auf unterschiedliche Datensätze zu, weil eine andere Transaktion T2, die während des Ablaufs von T1 läuft, Datensätze hinzugefügt, entfernt oder verändert hat.

SPRING JDBC

- Relationale Datenbank
  - Transaktionsisolation

Isolationsebene	Dirty Read	Lost Updates	Non-Repeatable Read	Phantom
Read Uncommitted	möglich	möglich	möglich	möglich
Read Committed	unmöglich	unmöglich	möglich	möglich
Repeatable Read	unmöglich	unmöglich	unmöglich	möglich
Serializable	unmöglich	unmöglich	unmöglich	unmöglich

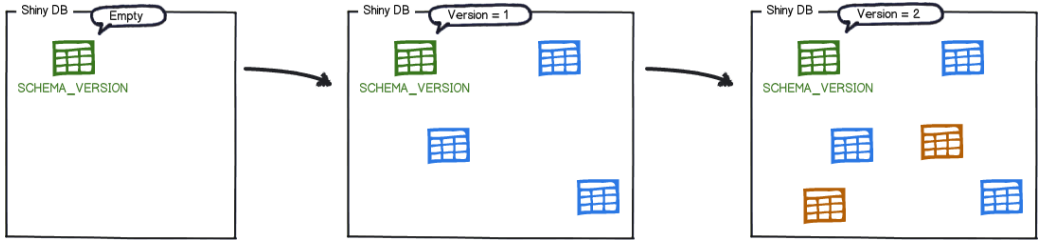
SPRING JDBC

- Relationale Datenbank
  - Transaktionspropagation
  - **Required:** Bereits begonnene Transaktion wird fortgesetzt, ansonsten eine neue gestartet.
  - **RequiresNew:** Es wird immer eine neue Transaktion gestartet. Eine bereits laufende Transaktion wird suspendiert. Ein Commit bzw. Rollback in dieser neuen Transaktion führt nicht zum Commit bzw. Rollback der bereits laufenden Transaktion. Unabhängig vom Ergebnis dieser neuen Transaktion wird anschließend mit der bereits laufenden Transaktion fortgefahren.
  - **Supports:** Bereits begonnene Transaktion wird fortgesetzt, jedoch keine neue gestartet.
  - **NotSupported:** Immer ohne Transaktion, auch wenn bereits vorher eine Transaktion gestartet wurde.
  - **Mandatory:** Transaktion muss aktiv sein. Sonst wird eine Exception geworfen.
  - **Never:** Es darf keine Transaktion aktiv sein. Sonst wird eine Exception geworfen.

SPRING JDBC

- Praxis 30 min
  - Installiert den SQL-Client *Squirrel* http://squirrel-sql.sourceforge.net
  - Installiert in *Squirrel* den Treiber HSQLDB Standalone für HSQLDB http://hsqldb.org
  - Legt in *Squirrel* einen neuen Alias an:
    - URL: `jdbc:hsqldb:file:c:/tmp/spring-data-jdbc/db/test1;hsqldb.write_delay=false;shutdown=true`
    - User Name: **user**
    - Password: **password**
  - Verbindet Euch mit *Squirrel* zur HSQLDB-Datenbank
  - Legt Tabellen für das Domänenmodell an und lest, schreibt und löscht Daten

SPRING JDBC



- Flyway http://flywaydb.org
  - Agile Datenbankentwicklung
    - Liquibase - http://www.liquibase.org
  - Versionierung
    - SCM
    - Definierte Datenbankstrukturen
    - Reproduzierbares Deployment

schema_version

version_rank	installed_rank	version	description	type	script	checksum	installed_by	installed_on	execution_time	success
1	1	1	Initial Setup	SQL	V1__Initial_Setup.sql	1996767037	axel	2010-05-04 22:23:00.0	546	true
2	2	2	First Changes	SQL	V2__First_Changes.sql	1279644856	axel	2010-05-06 09:18:00.0	127	true
3	3	2.1	Minor Refactoring	SPRING_JDBC	V2_1__Minor_Refactoring		axel	2010-05-10 17:45:06.4	251	true

SPRING JDBC

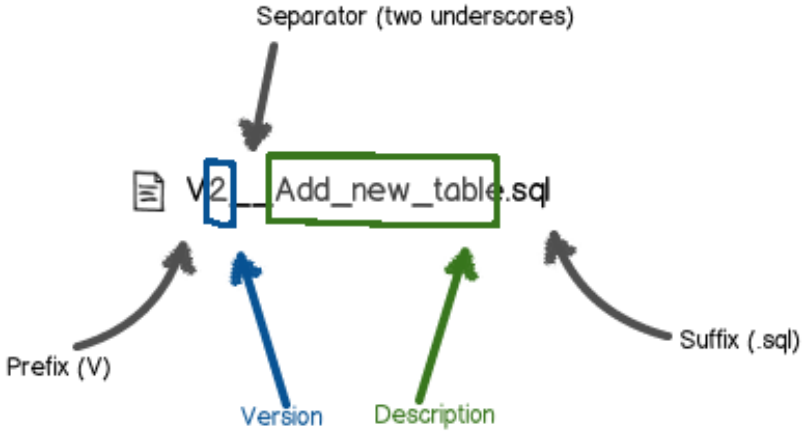
- Flyway
  - Basiert auf SQL-Skripten oder Java
  - Integration in Maven
    - Phasen:
      - `clean`: löscht ggf. die Datenbank
      - `process-resources`: spielt die Migrationsskripte ein und validiert diese
    - Konfiguration über Properties
      - Möglichkeit für unterschiedliche Profile

```
<properties>
   <flywaydb.version>3.1</flywaydb.version>
   <hsqldb.version>2.3.2</hsqldb.version>
</properties>
<plugin>
   <groupId>org.flywaydb</groupId>
   <artifactId>flyway-maven-plugin</artifactId>
   <version>${flywaydb.version}</version>
   <dependencies>
      <dependency>
         <groupId>org.hsqldb</groupId>
         <artifactId>hsqldb</artifactId>
         <version>${hsqldb.version}</version>
      </dependency>
   </dependencies>
   <executions>
      <execution>
         <id>clean</id>
         <phase>clean</phase>
         <configuration><skip>${profile.clean.skip}</skip></configuration>
         <goals><goal>clean</goal></goals>
      </execution>
      <execution>
         <id>migrate</id>
         <phase>process-resources</phase>
         <goals>
            <goal>migrate</goal>
            <goal>validate</goal
         </goals>
      </execution>
   </executions>
   <configuration>
      <encoding>UTF-8</encoding>
      <validateOnMigrate>true</validateOnMigrate>
      <configFile>${profile.propertyFile}</configFile>
   </configuration>
</plugin>
```

SPRING JDBC

- Flyway
  - Integration in Maven
    - Maven-Kommands:
      - `mvn flyway:clean` – löscht die konfigurierten Schemata
      - `mvn flyway:migrate` – führt die Migration durch
      - `mvn flyway:validate` – validiert die durchgeführten Migrationen
      - `mvn flyway:info` – gibt Überblick über die Migrationen
  - Kompatibilität zu HSQLDB
    - Standard Sql-Syntax mit Delimiter ;
    - DDL exportiert von HSQLDB können benutzt werden

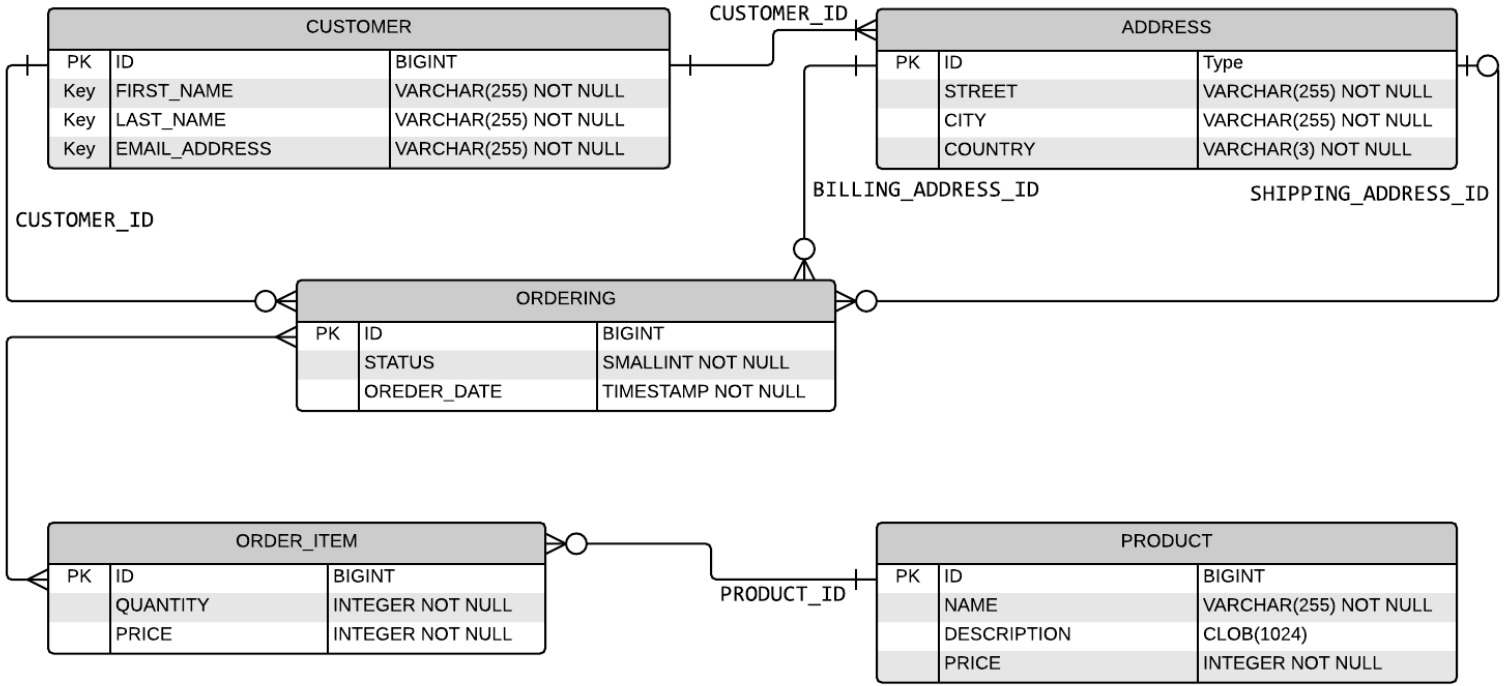
SPRING JDBC



- Flyway
  - Skripte
    - Name
      - `Prefix`: Konfigurierbar, default: V
      - `Version`: Punkte oder Unterstriche, z.B. 1.0.0 oder 1_2_3_4
      - `Separator`: Konfigurierbar, trennt Version von Beschreibung, default: __ (2 Unterstriche)
      - `Description`: Sinnvoller Beschreibung, Unterstriche zwischen den Wörtern
      - `Suffix`: Konfigurierbar, default: .sql
    - Beispiel: `V1.0.0.1__Create_Tables.sql`
    - Skripte unter `src/main/resources/db/migration`

SPRING JDBC

- Datenbankschema des Domänenmodells



SPRING JDBC

- Praxis 20 min
  - Legt ein neues Maven-Spring-Projekt an:
    - das Flyway einbindet,
    - dasdie SQL-Skripte für das Domänenmodell unter `src\main\resources\db\migration\1.0.0\hsqldb` ablegt
    - folgende Properties einbindet:

```
# Properties for hsqldb 2.x
flyway.driver=org.hsqldb.jdbcDriver
flyway.url=
jdbc:hsqldb:file:c:/tmp/spring-data-jdbc/db/
 testdb-jdbc;hsqldb.write_delay=false;shutdown=true
flyway.user=user
flyway.password=password
```

```
<properties>
   <flywaydb.version>3.1</flywaydb.version>
   <hsqldb.version>2.3.2</hsqldb.version>
</properties>
<plugin>
   <groupId>org.flywaydb</groupId>
   <artifactId>flyway-maven-plugin</artifactId>
   <version>${flywaydb.version}</version>
   <dependencies>
      <dependency>
         <groupId>org.hsqldb</groupId>
         <artifactId>hsqldb</artifactId>
         <version>${hsqldb.version}</version>
      </dependency>
   </dependencies>
   <executions>
      <execution>
         <id>clean</id>
         <phase>clean</phase>
         <configuration><skip>${profile.clean.skip}</skip>
         </configuration>
         <goals><goal>clean</goal></goals>
      </execution>
      <execution>
         <id>migrate</id>
         <phase>process-resources</phase>
         <goals>
            <goal>migrate</goal>
            <goal>validate</goal
         </goals>
      </execution>
   </executions>
   <configuration>
      <encoding>UTF-8</encoding>
      <validateOnMigrate>true</validateOnMigrate>
      <configFile>${profile.propertyFile}</configFile>
   </configuration>
</plugin>
```

SPRING JDBC

- Überblick

Was ist zu tun?	Spring	Du
Definiere Parameter für die Verbindung zur Datenbank		X
Öffne die Verbindung	X	
Spezifiziere die SQL-Statements		X
Deklariere Parameter and stelle entsprechende Werte bereit		X
Bereite die Statements vor und führe sie aus	X	
Baue das Resultat, ggf. eine Iteration	X	
Do the work für jeden Iterationsschritt		X
Behandle die Exceptions	X	
Handhabe Transaktionen	X	
Schliesse die Verbindung, die Statement s und Resultset	X	

SPRING JDBC

- JDBC-Zugriff
  - `org.springframework.jdbc.core.JdbcTemplate`
    - Klassischer Springansatz
  - `org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate`
    - Wrapper um `JdbcTemplate`, um benamte Parameter nutzen zu können
  - `org.springframework.jdbc.core.simple.SimpleJdbcInsert`, `org.springframework.jdbc.core.simple.SimpleJdbcCall`
    - Spezialisierte, wiederverwendbare Objekte
  - `org.springframework.jdbc.object.MappingSqlQuery`, `org.springframework.jdbc.object.SqlUpdate`, `org.springframework.jdbc.object.StoredProcedure`
    - Wiederverwertbare Kommandos

SPRING JDBC

- `org.springframework.jdbc.core.JdbcTemplate`
  - Benötigt `javax.sql.DataSource`
  - Erzeugen und freigeben von Resourcen
  - Threadsafe per `DataSource`
  - CRUD-Methoden

```
int rowCount = jdbcTemplate.queryForObject("select count(*) from PRODUCT", Integer.class);

String SEL_BY_ID = "select ID, FIRST_NAME, LAST_NAME, EMAIL_ADDRESS from CUSTOMER where ID = ?";
Customer customer = jdbcTemplate.queryForObject(SEL_BY_ID, new Object[]{id},
        new CustomerRowMapper());
final class CustomerRowMapper implements RowMapper<Customer> {
  @Override
  public Customer mapRow(final ResultSet rs, int i) throws SQLException {
    final Customer customer = new Customer(rs.getLong(CUSTOMER_ID));
    customer.withName(rs.getString(CUSTOMER_FIRST_NAME), rs.getString(CUSTOMER_LAST_NAME));
    customer.withEmailAddress(new EmailAddress(rs.getString(CUSTOMER_EMAIL_ADDRESS)));
    return customer;
  }
```

SPRING JDBC

- `org.springframework.jdbc.core.JdbcTemplate`
  - Insert mit `org.springframework.jdbc.core.PreparedStatementCreator`

```
Customer addCustomer(final Customer customer) {
  final KeyHolder keyHolder = new org.springframework.jdbc.support.GeneratedKeyHolder();
  int result = mJdbcTemplate.update(new CustomerInsert(customer), keyHolder);
  if(1 == result) {
    final Customer updatedCustomer = new Customer(keyHolder.getKey().longValue());
    updatedCustomer.withName(customer.getFirstName(), customer.getLastName());
    updatedCustomer.withEmailAddress(customer.getEmailAddress());
    return updatedCustomer;
  }
  return null;
}

String INSERT_SQL = "insert into CUSTOMER(FIRST_NAME, LAST_NAME, NORMALIZED, EMAIL_ADDRESS) values(?, ?, ?, ?)";
final class CustomerInsert implements PreparedStatementCreator {
  private final Customer mCustomer;
  CustomerInsert(final Customer customer) {
    mCustomer = customer;
  }
  @Override
  public PreparedStatement createPreparedStatement(final Connection connection) throws SQLException {
    final PreparedStatement ps = connection.prepareStatement(INSERT_SQL, new String[] {CUSTOMER_ID});
    ps.setString(1, mCustomer.getFirstName());
    ps.setString(2, mCustomer.getLastName());
    ps.setString(3, mCustomer.getNormalizedName());
    ps.setString(4, mCustomer.getEmailAddress().getAsString());
    return ps;
  }
}
```

SPRING JDBC

- `javax.sql.DataSource`
  - Konfiguration über Spring
    - XML

```
<bean id="pooledDataSource" class="org.apache.tomcat.jdbc.pool.DataSource"
     destroy-method="close">
    <property name="driverClassName" value="${flyway.driver}"/>
    <property name="url" value="${flyway.url}"/>
    <property name="username" value="${flyway.user}"/>
    <property name="password" value="${flyway.password}"/>
    <!-- these are special properties of the Tomcat JDBC Connection Pool-->
    <property name="initialSize" value="32"/>
    <property name="maxActive" value="128"/>
    <property name="maxIdle" value="64"/>
    <property name="jmxEnabled" value="true"/>
    <property name="fairQueue" value="true"/>
    <property name="defaultAutoCommit" value="false"/>
    <!--java.sql.Connection.TRANSACTION_READ_COMMITTED -->
    <property name="defaultTransactionIsolation" value="2"/>
</bean>
```

SPRING JDBC

- Transaktionen
  - Annotation `@org.springframework.transaction.annotation.Transactional`
  - Konfiguration – nutzt den Transaktionsmechanismus der `DataSource`
    - XML

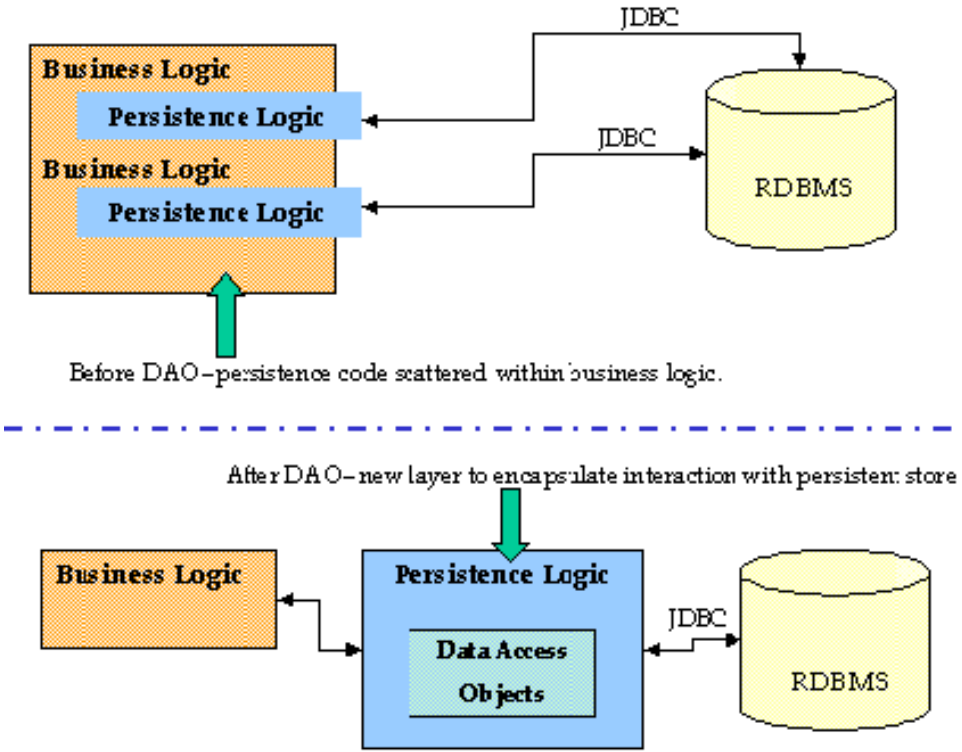
```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://.../spring-beans.xsd
            http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd>

      <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
          <constructor-arg ref="pooledDataSource"/>
      </bean>
      <tx:annotation-driven transaction-manager="transactionManager"/>
</beans>
```

  - Parameter
    - `propagation`
    - readOnly
    - isolation

```
@Transactional(propagation= Propagation.REQUIRED, readOnly=false)
public Customer addCustomer(final Customer customer) { … }
```

SPRING JDBC

- DAO-Pattern
  - **D**ata **A**ccess **O**bject
  - Abstraktionsebene zur Persistenz-Technologie
  - Entkopplung der Geschäftslogik vom Datenbankzugriff
  - Höhere Testbarkeit



SPRING JDBC

- DAO-Pattern

```
public interface CustomerDao {
  Customer addCustomer(Customer customer);
  Customer selectCustomer(long id);
}

class CustomerDaoImpl implements CustomerDao {
  private JdbcTemplate mJdbcTemplate;

  public void setDataSource(final DataSource dataSource) {
    mJdbcTemplate = new JdbcTemplate(dataSource);
  }

  @Override
  public Customer addCustomer(final Customer customer) {
        …  }

  @Override
  public Customer selectCustomer(long id) {
        …  }
}

<bean id="customerDao" class="com.jambit.workshop.jib.spring.data.jdbc.dao.impl.CustomerDaoImpl">
    <property name="dataSource" ref="pooledDataSource"/>
</bean>
```

SPRING JDBC

- DAO-Pattern
  - Test

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = { "classpath*:spring/testApplicationContext.xml" })
public class CostumerDaoTest extends AbstractJUnit4SpringContextTests {
  … }
```

  - Eigene Konfiguration für Test
    - Basierend auf der Standard-Konfiguration
  - Spring-Test-Unterstüzung durch

```
<groupId>org.springframework</groupId>
<artifactId>spring-test</artifactId>
```

SPRING JDBC

- Praxis 35 min
  - Erweitert Eure Spring-Anwendung um Persistenz für eine Domänenklasse:
    - benutzt das DAO-Pattern mit JDBC-Template
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür

Zusätzlich zu den andren Dependencies:

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context-support</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jdbc</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-tx</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.hsqldb</groupId>
   <artifactId>hsqldb</artifactId>
   <version>2.3</version>
</dependency>
```

SPRING JDBC

- `org.springframework.jdbc.core.support.JdbcDaoSupport`
  - Komfortklasse für `JdbcTemplate`
  - Kapselt `JdbcTemplate`, benötigt `DataSource`

```
@Autowired
private DataSource mDataSource;

@PostConstruct
private void initialize() {
  setDataSource(mDataSource);
}
```

  - Für Subklassen: `getJdbcTemplate()`

```
int result = getJdbcTemplate().update(new ProductInsert(product), keyHolder);
```

SPRING JDBC

- Praxis 15 min
  - Erweitert Eure Spring-Anwendung um Persistenz für eine weitere Domänenklasse:
    - benutzt das DAO-Pattern mit JdbcDaoSupport
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür

Zusätzlich zu den andren Dependencies:

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context-support</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jdbc</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-tx</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.hsqldb</groupId>
   <artifactId>hsqldb</artifactId>
   <version>2.3</version>
</dependency>
```

SPRING JDBC

- `org.springframework.jdbc.core.simple.SimpleJdbcInsert`
  - benötigt `DataSource` oder `JdbcTemplate`
  - Fluent Interface

```
String TABLE_ADDRESS = "ADDRESS"
String ADDRESS_ID = "ID";
new SimpleJdbcInsert(dataSource).withTableName(TABLE_ADDRESS).usingGeneratedKeyColumns(ADDRESS_ID);

final SqlParameterSource parameters = new MapSqlParameterSource()
      .addValue(ADDRESS_CUSTOMER_ID, customer.getOptionalId().get())
      .addValue(ADDRESS_CITY, address.getCity())
      .addValue(ADDRESS_STREET, address.getStreet())
      .addValue(ADDRESS_COUNTRY, address.getCountry().getAlpha3());
return Optional.ofNullable((Long) mInsertAddress.executeAndReturnKey(parameters));
```

  - Interface `org.springframework.jdbc.core.namedparam. SqlParameterSource`
    - Werte für benamte SQL-Parameter

SPRING JDBC

- Abstrakte Klasse `org.springframework.jdbc.object. MappingSqlQuery<T>`
  - benötigt `DataSource` und SQL-String für Abfrage
  - Wiederverwendbare Abfrage
  - Subklassen `T mapRow(ResultSet rs, int rowNum)`

```
Address mapRow(final ResultSet rs, int rowNumber) throws SQLException {
  final Address found = new Address(rs.getLong(ADDRESS_ID));
  found.with(rs.getString(ADDRESS_STREET), rs.getString(ADDRESS_CITY),
        CountryCode.getByCodeIgnoreCase(rs.getString(ADDRESS_COUNTRY)));
  return found;
}
```

  - Kapslung der Queries

```
final class AddressMappingQueries {
static abstract class AbstractAddressMappingQuery extends MappingSqlQuery<Address> { … }
static final class AddressMappingQueryById extends AbstractAddressMappingQuery { … }
static final class AddressMappingQueryByCustomer extends AbstractAddressMappingQuery { … }
}
```

SPRING JDBC

- Exceptionhandling
  - Hilfsklassen in `org.springframework.jdbc.support`
  - Interface `SQLExceptionTranslator`

```
org.springframework.dao.DataAccessException translate(String task, String sql, SQLException ex)

class CustomSQLErrorCodesTranslator extends SQLErrorCodeSQLExceptionTranslator {
   public DataAccessException translatePermissionDenied(String task, String sql, SQLException sqlex) {
      final String errorCode = Integer.toString(sqlex.getErrorCode());
      if (Arrays.stream(SQLErrorCodes.getPermissionDeniedCodes())
            .filter(s -> s.equalsIgnoreCase(errorCode)).findAny()) ) {
        return new org.springframework.dao.PermissionDeniedDataAccessException(task, sqlex);
      }
      return null;
   }
}
```

  - Integration in `JdbcTemplate`

```
void setExceptionTranslator(SQLExceptionTranslator translator)
```

SPRING JDBC

- Exceptionhandling
  - Package `org.springframework.dao`
  - Exception-Hierarchie, basierend auf `DataAccessException`
  - `org.springframework.core.NestedRuntimeException`
    - Wrapper für `RuntimeException` mit Auslöserexception

```
Throwable getRootCause()
Throwable getMostSpecificCause()
```

  - `DataAccessResourceFailureException` - keine Verbindung zur Datenbank
  - `QueryTimeoutException` – Abfragen dauern zu lange
  - `DuplicateKeyException` – Verletzung der PK beim Insert
  - `IncorrectResultSizeDataAccessException` – Größe des ResultSet inkorrekt
  - `EmptyResultDataAccessException` – ResultSet ist unerwartet leer

SPRING JDBC

- Praxis 30 min
  - Erweitert Eure Spring-Anwendung um Persistenz für eine weitere Domänenklasse:
    - benutzt das DAO-Pattern mit `SimpleJdbcInsert` und `MappingSqlQuery`
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür
    - braucht Ihr ein spezielles Exceptionhandling?

Zusätzlich zu den andren Dependencies:

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-context-support</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-jdbc</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-tx</artifactId>
   <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
   <groupId>org.hsqldb</groupId>
   <artifactId>hsqldb</artifactId>
   <version>2.3</version>
</dependency>
```

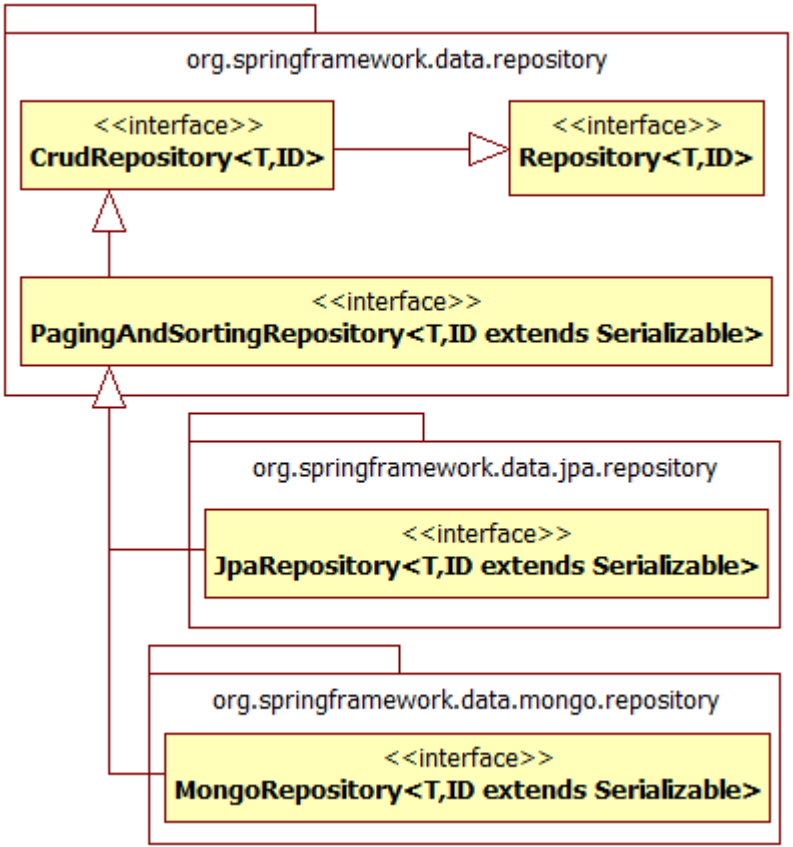
SPRING DATA JPA

- Spring Data http://projects.spring.io/spring-data
  - Ziele
    - Gemeinsames API für relationale und NoSQL-Datenbanken
    - Vereinfachung
  - Mantelprojekt
    - Spring Data Commons
    - Spring Data JPA
    - Spring Data MongoDB
    - Community Projekte wie Spring Data Cassandra

SPRING DATA JPA

- Spring Data Commons
  - allgemeine Infrastruktur und Interfaces
  - einheitliches API
    - CRUD-Operationen
    - Sortierung
    - Seitenweises Lesen (Pagination)

  - Package `org.springframework.data.repository`
    - `Repository` Marker-Interface
    - NICHT `@Repository`



SPRING DATA JPA

- Spring Data Commons

```
interface OrderRepository extends PagingAndSortingRepository<Order, Long>
```

  - Interface `org.springframework.data.repository.PagingAndSortingRepository`

```
Iterable<T> findAll(Sort sort)
Page<T> findAll(Pageable pageable)
List<T> findAll(Sort sort)
```

  - Interface `org.springframework.data.domain.Pageable`
  - Basisklasse `org.springframework.data.domain.PageRequest`

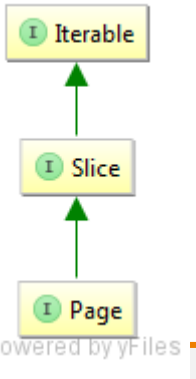
```
Page<Order> page = orderRepository.findAll(new PageRequest(2, 20)) // Seite 3 mit 20 Einträgen
List<Order> orders = page.getContent()
```

  - Interface `org.springframework.data.domain.Page`

```
int getTotalPages()
long getTotalElements()
```

  - Interface `org.springframework.data.domain.Slice`

```
int getNumber()
int getNumberOfElements()
List<T> getContent()
```



SPRING DATA JPA

- Spring Data Commons

```
interface OrderRepository extends PagingAndSortingRepository<Order, Long>
```

  - Interface org.springframework.data.repository.CrudRepository<T, ID>

```
<S extends T> S save(S entity)
T findOne(ID id)
Iterable<T> findAll()
Iterable<T> findAll(Iterable<ID> ids)
void delete(T entity)
void delete(ID id)
```

```
Order order = new Order( … );
Order savedOrder = orderRepository.save(order);

Order foundOrder = orderRepository.findOne(savedOrder.getId());
```

SPRING DATA JPA

- Spring Data JPA
  - (fast) transparent
  - JPA Teil des JEE-Stacks
    - Java Persistence API
    - Aktuelle Version 2.1
    - Standardisierte Schnittstelle zum Persistieren von POJOs in relationalen DB
    - Eigene Abfragesprache JPQL

```
interface JpaRepository<T, ID> extends
PagingAndSortingRepository

class SimpleJpaRepository<T, ID extends
Serializable> implements JpaRepository
```

```
<properties>
  <hibernate-entitymanager.version>4.3.8.Final</hibernate-entitymanager.version>
  <spring-data-jpa.version>1.7.1.RELEASE</spring-data-jpa.version>
  <hibernate-jpa-2.1-api.version>1.0.0.Final</hibernate-jpa-2.1-api.version>
<properties>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>${spring-data-jpa.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>

<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>${hibernate-jpa-2.1-api.version}</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate-entitymanager.version}</version>
    <scope>runtime</scope>
</dependency>
```

SPRING DATA JPA

- Spring Data JPA

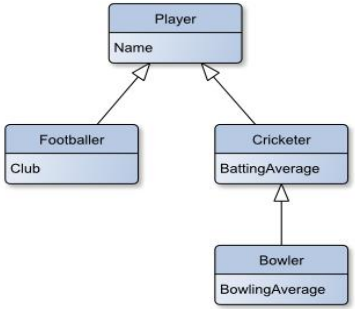
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:jdbc="http://www.springframework.org/schema/jdbc"
   xmlns:tx="http://www.springframework.org/schema/tx"
   xmlns:jpa="http://www.springframework.org/schema/data/jpa"
   xsi:schemaLocation="http://www.springframework.org/schema/jdbc http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
      http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/data/jpa http://www.springframework.org/schema/data/jpa/spring-jpa.xsd
      http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd">

    <bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager">
      <property name="entityManagerFactory" ref="entityManagerFactory" />
   </bean>

    <tx:annotation-driven transaction-manager="transactionManager"/>

    <jpa:repositories base-package="com.jambit.workshop.jib.spring.data.jpa" />

   <jdbc:embedded-database id="dataSource" type="HSQL" /> <!-- jdbc:hsqldb:mem:testdb sa -->
   <!--import resource="classpath:spring/spring-db-config.xml"/-->

   <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
      <property name="dataSource" ref="dataSource" />
      <property name="packagesToScan" value="com.jambit.workshop.jib.spring.data.jpa" />
      <property name="jpaVendorAdapter">
         <bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
            <property name="database" value="HSQL" />
            <property name="generateDdl" value="true" />
               <property name="showSql" value="true" />
         </bean>
      </property>
   </bean>
</beans>
```

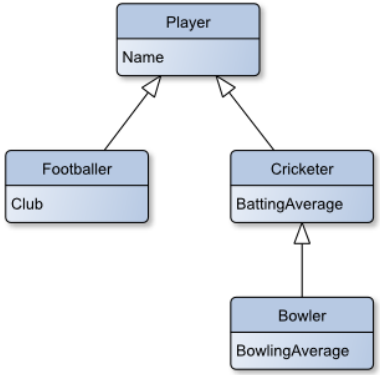
SPRING DATA JPA

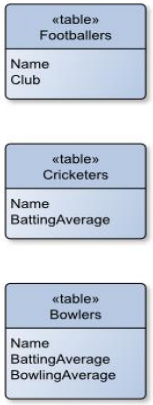
- Spring Data JPA
  - Hibernate http://hibernate.org
    - Objekt-Relationaler Mapper (ORM)
    - Aktuelle Version 4.3.8
    - Klassen auf Tabellen abgebildet
    - Identität des Objektes entspricht PK
    - Referenz auf anderes Objekt entspricht FK
  - Konkurrenten
    - myBatis http://mybatis.org
    - EclipseLink http://www.eclipse.org/eclipselink
    - Apache Cayenne http://cayenne.apache.org

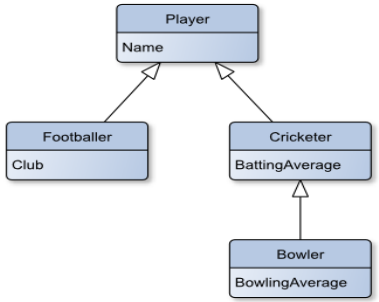
SPRING DATA JPA

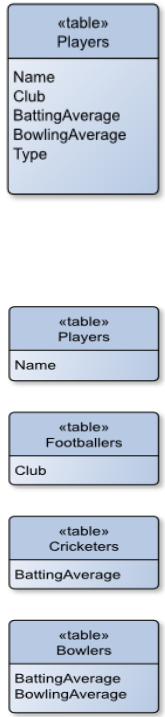
- Spring Data JPA
  - Vererbungshierachien
    - *Single Table:* eine Tabelle pro Hierarchie
    - *Class Table:* Tabelle pro Basisklasse und für jede abgeleitete Unterklasse
    - *Concrete Table:* Tabelle pro konkreter Klasse











SPRING DATA JPA

- Spring Data JPA
  - Package `javax.persistence`
    - `@Entity`
      - Klasse voll via JPA persistiert werden
      - Nicht final, auch Methoden und zu speichernde Member
      - Default-Konstruktor
    - `@Table(name = "company")`
      - Zusammen mit `@Enitity` Tabelle zum Persisitieren
    - `@Id`
      - Repräsentiert PK
      - Muss serialisierbar sein
    - `@GeneratedValue`
      - (strategy = GenerationType.AUTO)
      - `Enum` mit Werten `TABLE, SEQUENCE, IDENTITY, AUTO`

SPRING DATA JPA

- Spring Data JPA
  - `@Column`
    - Mapping zwischen Member und Tabellenspalte
    - Eigenschaften wie `nullable, unique, length: @Column(name = „street")`
  - `@Basic`
    - Defaultmapping für Member
    - Member muss Primitivtyp oder serialisierbar sein
    - Eigenschaften
      - `optional`
      - `fetch Enum FetchType: EAGER, LAZY`
  - `@Transient`
    - Member nicht persistent
  - `@Embeddable`
    - Objekte dieser Klasse werden als Teil der Klasse gespeichert, deren Member sie sind
    - Einbindung über `@Column`
  - `@MappedSuperclass`
    - Gemappte Attribute werden in den Unterklassen persistiert

SPRING DATA JPA

Persistence Context:

- Konzeptionelle Idee
- Abstraktion, in dem Objektinstanzen und deren persistente Repräsentierung konsistent gehalten werden

- Spring Data JPA
  - Lebenszyklus
    - `New`
      - Neu erzeugtes Objekt
      - Nicht persistiert, ohne persistente Identität
    - `Managed`
      - Erfolgreich über JPA persistiert
      - Eigene persistente Identität
      - Teil des Persistence Context
    - `Removed`
      - Persistente Identität ist gelöscht ebenso kein Teil des Persistence Context
      - Zustand des aktiven Objektes unverändert!
    - `Detached`
      - Objekt mit persistenter Identität, doch nicht im aktuell genutzten Persistence Context

SPRING DATA JPA

- Spring Data JPA
  - Interface `javax.persistence.EntityManager`
    - Zentraler Zugriff auf Persistence Context
    - CRUD für Objekte des Persistence Context

```
void persist(Object entity)
<T> T merge(T entity)
void remove(Object entit
void flush()  - synchronisiert Persistence Context mit der DB
<T> T find(Class<T> entityClass, Object primaryKey)
void detach(Object entity)
boolean contains(Object entity)
void lock(Object entity, LockModeType lockMode)
```

SPRING DATA JPA

- Spring Data JPA
  - Locking-Strategien
    - Bei Transaktionen
    - Optimistic
      - Versionierung der Tabelle oder Tabellenzeile z.B. mit Zeitstempel
      - Annahme: wenige schreibende, aber viele lesende Zugriffe
        - Lesende Zugriffe lösen daher keine Sperren aus
      - Bei Änderungen, nochmaliges Lesen notwendig
      - Speicherung ändert Version
    - Pessimistic
      - Annahme: viele schreibende Zugriffe
      - Änderungen sperren Zugriff auf Tabelle bzw. Tabellenzeile
      - Daten werden erst freigegeben, wenn Änderungen mit `commit` oder `rollback` gespeichert sind.

SPRING DATA JPA

- Spring Data JPA
  - Enum `javax.persistence.LockModeType`
    - OPTIMISTIC – optimistisches Lock
    - PESSIMISTIC_READ
      - Lock scheitert, wenn ein Objekt im Persistence Context bereits durch ein PESSIMISTIC_WRITE gesperrt ist
      - shared lock
    - PESSIMISTIC_WRITE
      - Lock scheitert, wenn ein Objekt im Persistence Context bereits ein PESSIMISTIC_READ oder PESSIMISTIC_WRITE hält
      - exclusive lock

`TransactionRequiredException` – `PersistenceException`, wenn Transaktion notwendig, aber nicht aktiv ist

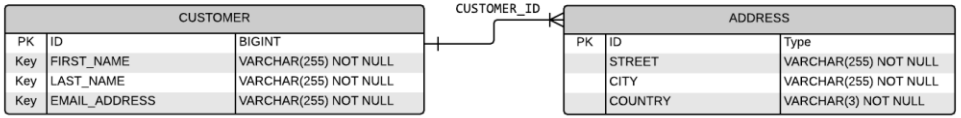
`PessimisticLockException` – `PersistenceException` bei Konflikten durch pessimistisches Locking

`OptimisticLockException` – `PersistenceException` bei Konflikten durch optimistisches Locking

SPRING DATA JPA

- Spring Data JPA
  - Beziehungen zwischen Entitäten
    - Jede Beziehung hat einen Besitzer, der das Update bestimmt
    - `@JoinTable`
      – Verknüpfungstabelle, v.a. für N-to-N und 1-to-N
    - `@JoinColumn`
      – Spalte zum Mapping
    - `@One-to-Many`
      – `targetEntity` – Mapping-Klasse
      – `cascade`
      – `orphanRemoval`

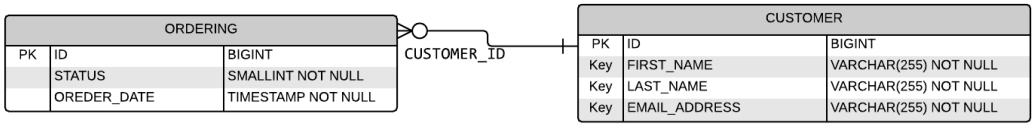
```
@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name = "CUSTOMER_ID")
private Set<Address> addresses;
```



SPRING DATA JPA

- Spring Data JPA
  - Beziehungen zwischen Entitäten
    - `@ManyToOne`
      - `optional` – default `true`
      - `fetch` – `FetchType.EAGER`, `FetchType.LAZY`

```
@ManyToOne(optional = false)
private Customer customer;
```



    - `@OneToOne`
      - `optional` – default `true`
      - `fetch` – `FetchType.EAGER`, `FetchType.LAZY`
    - `@ManyToMany`
      - `optional` – default `true`
      - `fetch` – `FetchType.EAGER`, `FetchType.LAZY`

SPRING DATA JPA

- Spring Data JPA
  - In-Memory HSQLDB (*jdbc:hsqldb:mem:testdb User: sa)*

```
<jdbc:embedded-database id="dataSource" type="HSQL" />
```

  - Schema `http://www.springframework.org/schema/jdbc`

```
http://www.springframework.org/schema/jdbc/spring-jdbc.xsd
<jdbc:initialize-database data-source="dataSource" ignore-failures="ALL">
      <jdbc:script location="classpath:data.sql" />
</jdbc:initialize-database>
```

  - Konfiguration mit Java

```
DataSource dataSource() {
   final EmbeddedDatabaseBuilder builder = new EmbeddedDatabaseBuilder();
   return builder.setType(EmbeddedDatabaseType.HSQL).build();
}
```

SPRING DATA JPA

- Spring Data JPA

```
private static final String POPULATION_DATA = "data.sql";

@Autowired
DataSource dataSource;

public void populateDatabase() throws SQLException {
    final ResourceDatabasePopulator populator = createPopulator();
    doPopulate(populator);
}

private ResourceDatabasePopulator createPopulator() {
    final ResourceDatabasePopulator populator = new ResourceDatabasePopulator();
    populator.addScript(new ClassPathResource(POPULATION_DATA));
    return populator;
}

private void doPopulate(final ResourceDatabasePopulator populator) {
    Connection connection = null;
    try {
        connection = DataSourceUtils.getConnection(dataSource);
        populator.populate(connection);
    } finally {
        if (connection != null) {
            DataSourceUtils.releaseConnection(connection, dataSource);
        }
    }
}
```

SPRING DATA JPA

- Praxis 45 min
  - Erstellt eine Spring-Anwendung, die zur Persistenz Spring-Data-JPA benutzt. Baut dazu für eine Domänenklasse
    - benutzt die JPA-Annotationen
    - das DAO-Pattern mit `PagingAndSortingRepository`
    - benutzt die In-Memory-HSQLDB
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür

```
<properties>
  <hibernate-entitymanager.version>4.3.8.Final</hibernate-entitymanager.version>
  <spring-data-jpa.version>1.7.1.RELEASE</spring-data-jpa.version>
  <hibernate-jpa-2.1-api.version>1.0.0.Final</hibernate-jpa-2.1-api.version>
<properties>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>${spring-data-jpa.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>

<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>${hibernate-jpa-2.1-api.version}</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate-entitymanager.version}</version>
    <scope>runtime</scope>
</dependency>
```

SPRING DATA JPA

- Spring Data JPA
  - Spring-db.config.xml

```
<bean id="dataSource" class="org.apache.tomcat.jdbc.pool.DataSource" destroy-method="close">
    <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
    <property name="url" value=
        "jdbc:hsqldb:file:c:/tmp/spring-data-jdbc/db/testdb-jpa;hsqldb.write_delay=false;shutdown=true"/>
    <property name="username" value="user"/>
    <property name="password" value="password"/>
    <!-- these are special properties of the Tomcat JDBC Connection Pool-->
    <property name="initialSize" value="32"/>
    <property name="maxActive" value="128"/>
    <property name="maxIdle" value="64"/>
    <property name="jmxEnabled" value="true"/>
    <property name="fairQueue" value="true"/>
    <property name="defaultAutoCommit" value="false"/>
    <!--java.sql.Connection.TRANSACTION_READ_COMMITTED -->
    <property name="defaultTransactionIsolation" value="2"/>
</bean>
```

  - In application-context.xml
    - Statt `<jdbc:embedded-database … />`

```
<import resource="classpath:spring/spring-db-config.xml">
```

SPRING DATA JPA

- Praxis 15 min
  - Tauscht die DB Eurer Spring-Anwendung aus
    - Statt In-Memory-HSQLDB die File-basierte HSQLDB
    - Funktionieren Eure Tests noch?
    - Vergleicht das von Hibernate generierte DB-Schema mit dem selbstgeschriebenen. Gibt es Unterschiede?

```
<properties>
  <hibernate-entitymanager.version>4.3.8.Final</hibernate-entitymanager.version>
  <spring-data-jpa.version>1.7.1.RELEASE</spring-data-jpa.version>
  <hibernate-jpa-2.1-api.version>1.0.0.Final</hibernate-jpa-2.1-api.version>
<properties>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>${spring-data-jpa.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>

<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>${hibernate-jpa-2.1-api.version}</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate-entitymanager.version}</version>
    <scope>runtime</scope>
</dependency>
```

SPRING DATA JPA

- Spring Data JPA
  - JPA basierte Implementierung
    - `@PersistenceContext`
      – deklariert Container-Managed EntityManager und den verknüpften Persistence Context

```
@PersistenceContext
private EntityManager mEntityManager;

Customer save(Customer customer) {
    if (canBeSaved(customer)) {
        mEntityManager.persist(customer);
        return customer;
    }
    return mEntityManager.merge(customer);
}

private boolean canBeSaved(final Customer customer) {
    return null != customer && null != customer.getId();
}
```

SPRING DATA JPA

- Praxis 15 min
  - Erweitert Eure Spring-Anwendung um Persistenz für eine weitere Domänenklasse:
    - benutzt das DAO-Pattern mit einer eigenen JPA basierten Implementierung
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür

```
<properties>
  <hibernate-entitymanager.version>4.3.8.Final</hibernate-entitymanager.version>
  <spring-data-jpa.version>1.7.1.RELEASE</spring-data-jpa.version>
  <hibernate-jpa-2.1-api.version>1.0.0.Final</hibernate-jpa-2.1-api.version>
<properties>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>${spring-data-jpa.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>${spring.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>

<dependency>
    <groupId>org.hibernate.javax.persistence</groupId>
    <artifactId>hibernate-jpa-2.1-api</artifactId>
    <version>${hibernate-jpa-2.1-api.version}</version>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>${hibernate-entitymanager.version}</version>
    <scope>runtime</scope>
</dependency>
```

SPRING DATA JPA

- Spring Data JPA
  - Java Persistence Query Language (JPQL)
    - Plattformunabhängige, objektorientierte Abfragesprache für relationale DB
    - Teil von JPA
    - Select, Update, Delete
    - `@org.springframework.data.jpa.repository.Query`

```
SELECT c from Customer
```

```
@Query("select p from Product p where p.price >= :from and p.price <= :to")
List<Product> findByPriceRange(@Param("from") BigDecimal p1, @Param("to") BigDecimal p2)
```

```
@Query("select p from Product p where p.price >= ?1")
List<Product> findFromPrice(BigDecimal from);
```

SPRING DATA JPA

- Spring Data JPA
  - QueryDSL http://www.querydsl.com
    - Konstruktion typsichere Abfragen
    - Basiert auf JPA, unabhängig von gewähltem ORMapper
  - Erweiterung des JpaRepository mit `org.springframework.data.querydsl.QueryDslPredicateExecutor`

```
interface ProductRepository extends JpaRepository<Product, Long>,
QueryDslPredicateExecutor<Product>
```

  - Generierung des Query-Modells

    `target/generated-sources`

SPRING DATA JPA

- Spring Data JPA
  - Maven

```
<dependency>
    <groupId>com.mysema.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
    <version>3.6.0</version>
</dependency>

<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>${apt-maven-plugin.version}</version>
    <dependencies>
        <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>1.1.3</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>target/generated-sources</outputDirectory>
                <processor>com.mysema.query.apt.jpa.JPAAnnotationProcessor</processor>
            </configuration>
        </execution>
    </executions>
</plugin>
```

SPRING DATA JPA

- Spring Data JPA
  - Modell für `Product`

```
/**
 * QProduct is a Querydsl query type for Product
 */
@Generated("com.mysema.query.codegen.EntitySerializer")
public class QProduct extends EntityPathBase<Product> {

    private static final long serialVersionUID = 984670051L;

    public static final QProduct product = new QProduct("product");

    public final QAbstractEntity _super = new QAbstractEntity(this);

    public final MapPath<String, String, StringPath> attributes = this.<String, String, StringPath>createMap("attributes", String.class, String.class,
            StringPath.class);

    public final StringPath description = createString("description");

    //inherited
    public final NumberPath<Long> id = _super.id;

    public final StringPath name = createString("name");

    public final NumberPath<java.math.BigDecimal> price = createNumber("price", java.math.BigDecimal.class);

    public QProduct(String variable) {
        super(Product.class, forVariable(variable));
    }

    public QProduct(Path<? extends Product> path) {
        super(path.getType(), path.getMetadata());
    }

    public QProduct(PathMetadata<?> metadata) {
        super(Product.class, metadata);
    }
}
```

SPRING DATA JPA

- Spring Data JPA
  - Queries

```
static final QProduct product = QProduct.product;

@Autowired
ProductRepository repository;

@Test
public void testFindProductsByQuerydslPredicate() {

    Product iPad = repository.findOne(product.name.eq("iPad"));
    Predicate tablets = product.description.contains("tablet");

    Iterable<Product> result = repository.findAll(tablets);
    assertThat(result, is(Matchers.<Product>iterableWithSize(1)));
    assertThat(result, hasItem(iPad));
}
```

SPRING DATA JPA

http://www.querydsl.com/static/querydsl/3.6.0/apidocs/

- Spring Data JPA
  - Klasse com.mysema.query.types.expr.SimpleExpression<T>

```
BooleanExpression eq(T right)
BooleanExpression eqAll(CollectionExpression<?,? super T> right)
BooleanExpression eqAny(CollectionExpression<?,? super T> right)
BooleanExpression in(Collection<? extends T> right)
BooleanExpression in(CollectionExpression<?,? extends T> right)
BooleanExpression notIn(Collection<? extends T> right)
SimpleExpression<T> nullif(T other)
```

  - Klasse com.mysema.query.types.expr. StringExpression

```
BooleanExpression contains(String str)
BooleanExpression containsIgnoreCase(String str)
BooleanExpression containsIgnoreCase(String str)
BooleanExpression isEmpty()
BooleanExpression isNotEmpty()
```

SPRING DATA JPA

- Spring Data JPA
  - Interface `org.springframework.data.querydsl.QueryDslPredicateExecutor<T>`
    - Führt Instanzen von QueryDsl´s `com.mysema.query.types.Predicate` aus

```
interface ProductRepository extends JpaRepository<Product, Long>,
QueryDslPredicateExecutor<Product>

T findOne(com.mysema.query.types.Predicate predicate)
Iterable<T> findAll(com.mysema.query.types.Predicate predicate)
Page<T> findAll(com.mysema.query.types.Predicate predicate, Pageable pageable)
long count(com.mysema.query.types.Predicate predicate)

Product iPad = repository.findOne(product.name.eq("iPad"))
```

SPRING DATA JPA

- Praxis 30 min
  - Erweitert Eure Spring-Anwendung um Persistenz für eine weitere Domänenklasse:
    - benutzt das DAO-Pattern mit `QueryDslPredicateExecutor` und `JpaRepository`
    - implementiert das Anlegen und das Auslesen
    - schreibt die Abfragen mit QueryDSL in JUnit-Tests

```
<dependency>
    <groupId>com.mysema.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
    <version>3.6.0</version>
</dependency>

<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>1.1.3</version>
    <dependencies>
        <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>1.1.3</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>
                    target/generated-sources
                </outputDirectory>
                <processor>
            com.mysema.query.apt.jpa.JPAAnnotationProcessor
                </processor>
            </configuration>
        </execution>
    </executions>
</plugin>
```

SPRING DATA JPA

- Praxis 10 min
  - Vergleicht Eure Spring-Anwendung basierend auf JPA mit der auf JDBC basierenden
    - Welche Lösung gefällt Euch besser?
    - Welche würdet Ihr einsetzen wollen?
    - Diskutiert!

```
<dependency>
    <groupId>com.mysema.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
    <version>3.6.0</version>
</dependency>

<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>1.1.3</version>
    <dependencies>
        <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>1.1.3</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>
                    target/generated-sources
                </outputDirectory>
                <processor>
            com.mysema.query.apt.jpa.JPAAnnotationProcessor
                </processor>
            </configuration>
        </execution>
    </executions>
</plugin>
```

SPRING DATA MONGODB

- Spring Data MongoDB
  - Integration von MongoDB in Spring
  - Unterstützung von QueryDSL
  - Map-Reduce-Integration
  - Exceptionhandling nach Spring

```
interface MongoRepository<T, ID> extends
PagingAndSortingRepository

class SimpleMongoRepository<T, ID extends
Serializable> implements MongoRepository
```

```
<properties>
    <spring-data-mongodb.version>1.6.1.RELEASE
    </spring-data-mongodb.version>
    <jackson.version>2.5.0</jackson.version>
<properties>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
    <version>${spring-data-mongodb.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>${jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>${jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jdk8</artifactId>
    <version>${jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>${jackson.version}</version>
</dependency>
```

SPRING DATA MONGODB

- Spring Data MongoDB
  - MongoDB http://www.mongodb.org
    - *hu**mongo**us* für gigantisch
    - Erste Version Februar 2009, open-source
    - Schemafrei, benutzt dynamische Schemas
    - Dokumentenorientiert, gespeichert in Collections
    - BSON - Binary JSON http://bsonspec.org
    - Unterstützt durch Treiber: C, C++, C#, Haskell, Java, JavaScript, Perl, PHP, Python, Ruby, Scala
    - hohe Skalierbarkeit
    - Unterstützt CRUD-Operationen
    - Keine Joins
    - Transaktionen nur für einzelne Operationen, kein ACID

SPRING DATA MONGODB

- Praxis 15 min
  - Installiert lokal mongodb:

https://www.mongodb.org/downloads
http://docs.mongodb.org/manual/installation/

  - Start des Servers durch `mongod.exe –dbpath c:\data\mongodb`
  - Start des Clients durch `mongo.exe`
    - `help`
    - `show dbs`
    - `use workshop`
    - `show collections`
    -
```
db.products.insert( {"_id" : 1234,
                     "_class" : "com.jambit.workshop.jib.spring.data.mongo.domain.Product",
                     "name" : "insert1",
                     "description" : "desc for insert1",
                     "price" : 1111}
                   )
```
    - `db.products.help()`
    - `db.products.count()`
    - `db.products.find({_id:1234})`
    - …

SPRING DATA MONGODB

- Konfiguration in Spring

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:c="http://www.springframework.org/schema/c"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:mongo="http://www.springframework.org/schema/data/mongo"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/data/mongo http://www.springframework.org/schema/data/mongo/spring-mongo.xsd
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

    <context:property-placeholder location="classpath:/properties/mongo.properties" ignore-unresolvable="true"/>

    <mongo:mongo id="mongoDB" host="${mongo.host}" port="${mongo.port}">
        <mongo:options
                connections-per-host="${mongo.connectionsPerHost}"
                threads-allowed-to-block-for-connection-multiplier="${mongo.threadsAllowedToBlockForConnectionMultiplier}"
                connect-timeout="${mongo.connectTimeout}"
                max-wait-time="${mongo.maxWaitTime}"
                auto-connect-retry="${mongo.autoConnectRetry}"
                socket-keep-alive="${mongo.socketKeepAlive}"
                socket-timeout="${mongo.socketTimeout}"
                slave-ok="${mongo.slaveOk}"
                write-number="1"
                write-timeout="0"
                write-fsync="${mongo.fsync}"/>
    </mongo:mongo>

    <mongo:db-factory id="mongoDbFactory" dbname="${mongo.dbname}" mongo-ref="mongoDB"/>

    <bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate"
                  c:mongoDbFactory-ref="mongoDbFactory" />

    <mongo:repositories base-package="com.jambit.workshop.jib.spring.data.mongo.repo"/>
</beans>
```

SPRING DATA MONGODB

- Konfiguration in Spring
  - `<mongo:db-factory id="mFactory" dbname="${mongo.dbname}" mongo-ref="mongoDB"/>`
    - Definiert eine MongoDbFactory für die Verbindung zur spezifizierten DB
  - `<bean id="mongoTemplate" c:mongoDbFactory-ref="mongoDbFactory" class="org.springframework.data.mongodb.core.MongoTemplate"/>`
    - Erzeugt Bean vom Typ `org.springframework.data.mongodb.core.MongoTemplate`
  - `<mongo:repositories base-package="com.jambit.workshop.jib.spring.data.mongo.repo"/>`
    - Definiert Repositories für MongoDB
  - `<mongo:mongo>`
    - Definiert eine Mongo-Instanz
  - `<mongo: mapping-converter>`
    - Definiert Mapper zwischen Java und BSON

SPRING DATA MONGODB

- Konfiguration in Spring
  - Properties

```
mongo.host=127.0.0.1
mongo.port=27017
mongo.dbname=workshop
mongo.connectionsPerHost=8
mongo.threadsAllowedToBlockForConnectionMultiplier=4
mongo.connectTimeout=1000
mongo.maxWaitTime=1500
mongo.autoConnectRetry=true
mongo.socketKeepAlive=true
mongo.socketTimeout=1500
mongo.slaveOk=true
mongo.fsync=true
```

  - ApplicationContext

```
<beans …
      <import resource="classpath:spring/spring-data-mongo.xml"/>

    <context:annotation-config/>
    <context:component-scan base-package="com.jambit.workshop.jib.spring.data.mongo" />

</beans>
```

SPRING DATA MONGODB

- Java – JSON

```
{ "_id" : 1234,
  "_class" : "com.jambit.workshop.jib.spring.data.mongo.domain.Product",
  "name" : "insert1",
  "description" : "desc for insert1",
  "price" : 1111
}
```

```
@Document(collection = "products")
public class Product {

  @Id
  private final String mId;
  @TextIndexed
  @Field("name")
  private String mName;
  @Field("description")
  private String mDescription;
  @Field("price")
  private Integer mPriceInCents;
}
```

SPRING DATA MONGODB

- Domänen-Mapping
  - Package `org.springframework.data.mongodb.core.mapping`
    - `@Document`
      - Klasse soll in MongoDB persistiert werden
      - Eigenschaft `collection` gibt an, in welcher Collection gespeichert werden soll

```
@Document(collection = "addresses")
```

    - `@Field`
      - Mapping zwischen Member und JSON-Eigenschaft
      - Eigenschaft value gibt das JSON-Element an `"street":"..."`

```
@Field("street")
private String mStreet;
```

    - `@DBRef`
      - Referenz von Dokument A zu Dokument B, basierend auf `_id`

```
@DBRef
private Customer mCustomer;
```

SPRING DATA MONGODB

- Domänen-Mapping
  - `@org.springframework.data.annotation.Id`
    - PK der Klasse
  - `@org.springframework.data.mongodb.core.index.Indexed`
    - Allgemeiner Index für Member
    - Eigenschaften `unique, direction` mit `IndexDirection.ASCENDING, IndexDirection.DESCENDING`

```
@Indexed(direction = IndexDirection.ASCENDING)
@Field("country")
private CountryCode mCountry;
```

  - `@org.springframework.data.mongodb.core.index.TextIndexed`
    - Member wird Teil des Textindexes der Collection, in dem alle mit `@TextIndexed` annotierten Elemente zusammengefasst werden

```
@Document(collection = "addresses")
public class Address {
  @TextIndexed
  @Field("street")
  private String mStreet;
```

SPRING DATA MONGODB

- Domänen-Mapping
  - `@org.springframework.data.annotation.PersistenceConstructor`
    - Deklariert einen Konstruktor, der beim Erzeugen von Objekten benutzt werden soll
  - `@org.springframework.beans.factory.annotation.Value`
    - Mapping zu Java

```
@PersistenceConstructor
public Product(@Value("#root.id") final String pId)
```

SPRING DATA MONGODB

- Repository
  - Interface `org.springframework.data.mongodb.repository.MongoRepository`
    - Repository für MongoDB
    - Defaultimplementierung `org.springframework.data.mongodb.repository.support.SimpleMongoRepository`
    - Konstruktorinjektion von `org.springframework.data.mongodb.core.MongoTemplate`

```
@Repository("productRepository")
@Transactional
public interface  ProductRepository extends MongoRepository<Product, String>
```

SPRING DATA MONGODB

- Testen
  - Wie bisher

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath*:spring/testApplicationContext.xml"})
public class ProductRepositoryTest extends AbstractJUnit4SpringContextTests {
    @Autowired
    @Qualifier("productRepository")
    ProductRepository mProductRepository;


@Test
public void testAddProduct() {
    final String id = createId();
    final Product product = new Product(id);
    product.withDescription("test_description").withName("test_name").withPriceInCents(100);
    final Product added = mProductRepository.save(product);
    assertNotNull(added);
    assertEquals(product, added);
    final Product found = mProductRepository.findOne(id);
    assertNotNull(found);
    assertEquals(product, found);
}
```

SPRING DATA MONDODB

- Praxis 30 min
  - Erstellt eine Spring-Anwendung, die zur Persistenz Spring-Data-MongoDB benutzt. Baut dazu für eine Domänenklasse
    - das DAO-Pattern mit MongoRepository
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür
    - Für Integrationstests muss der MongoDB-Server laufen!

```
<properties>
    <spring-data-mongodb.version>1.6.1.RELEASE
    </spring-data-mongodb.version>
    <jackson.version>2.5.0</jackson.version>
<properties>

<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
    <version>${spring-data-mongodb.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
</dependency>

<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>${jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>${jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jdk8</artifactId>
    <version>${jackson.version}</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.datatype</groupId>
    <artifactId>jackson-datatype-jsr310</artifactId>
    <version>${jackson.version}</version>
</dependency>
```

SPRING DATA MONGODB

- Converter
  - OrderStatus

```
/**
 * Represents the status of a given order.
 */
public enum OrderStatus {
  NEW(0), PENDING(1), PROCESSING(2), COMPLETE(3), CANCELED(4), HOLDED(5), CLOSED(6);

  private final int mStatusId;
  private OrderStatus(int statusId) {
    mStatusId = statusId;
  }

  public int getStatusId() {
    return mStatusId;
  }

  public boolean hasStatusId(int statusId) {
    return mStatusId == statusId;
  }

  public static OrderStatus getByStatusId(int id) {
    for (OrderStatus orderStatus : values()) {
      if(orderStatus.hasStatusId(id)) {
        return orderStatus;
      }
    }
    return NEW;
  }
}
```

SPRING DATA MONGODB

- Converter
  - Spring Type Conversion - ConversionService

```
package org.springframework.core.convert.converter;
public interface Converter<S, T> {
   T convert(S source);
}
```

  - Konverter für `OrderStatus`
    - Java -> MongoDB
    - MongoDB -> Java

SPRING DATA MONGODB

- Converter
  - Konverter für `OrderStatus` Java -> MongoDB

```
/**
 * Converter for converting objects of type <code>OrderStatus</code> to the mongodb.
 */
public class OrderStatusWriteConverter implements Converter<OrderStatus, DBObject> {
    @Override
    public DBObject convert(final OrderStatus source) {
        final DBObject dbo = new BasicDBObject();
        dbo.put("status", source.getStatusId());
        dbo.put("name", source.name());
        return dbo;
    }
}
```

  - Interface `com.mongodb.DBObject extends org.bson.BSONObject`
    - Key-Value-Map

SPRING DATA MONGODB

- Converter
  - Konverter für `OrderStatus` MongoDB -> Java

```
/**
 * Converter for converting objects of type <code>OrderStatus</code> from the mongodb.
 */
public class OrderStatusReadConverter implements Converter<DBObject, OrderStatus> {
    @Override
    public OrderStatus convert(final DBObject source) {
        return OrderStatus.getByStatusId((Integer) source.get("status"));
    }
}
```

  - Konvertierung via Spring Type Conversion für alle Typen möglich
    http://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/core/convert/package-summary.html

SPRING DATA MONGODB

- Converter
  - Konfiguration in ApplicationContext

```
<mongo:mapping-converter>
    <mongo:custom-converters>
        <mongo:converter>
            <bean class="com.jambit.workshop.jib.spring.data.mongo.converter.OrderStatusReadConverter"/>
        </mongo:converter>
        <mongo:converter>
            <bean class="com.jambit.workshop.jib.spring.data.mongo.converter.OrderStatusWriteConverter"/>
        </mongo:converter>
    </mongo:custom-converters>
</mongo:mapping-converter>

<bean id="mongoTemplate" class="org.springframework.data.mongodb.core.MongoTemplate"
      c:mongoDbFactory-ref="mongoDbFactory"
      c:mongoConverter-ref="mappingConverter" />
```

SPRING DATA MONGODB

- QueryDSL
  - Konfiguration in Maven

```
<!-- Querydsl -->
<dependency>
    <groupId>com.mysema.querydsl</groupId>
    <artifactId>querydsl-mongodb</artifactId>
    <version>${querydsl.version}</version>
</dependency>
<dependency>
    <groupId>org.mongodb.morphia</groupId>
    <artifactId>morphia</artifactId>
    <version>0.109</version>
</dependency>
```

SPRING DATA MONGODB

- QueryDSL
  - Konfiguration in Maven

```
<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>${apt-maven-plugin.version}</version>
    <dependencies>
        <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>${querydsl.version}</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>target/generated-sources</outputDirectory>
                <processor>com.mysema.query.apt.morphia.MorphiaAnnotationProcessor</processor>
            </configuration>
        </execution>
    </executions>
</plugin>
```

SPRING DATA MONGODB

- QueryDSL
  - Repository updaten

```
interface ProductRepository extends JpaRepository<Product, Long>,
QueryDslPredicateExecutor<Product>
```

  - Domänenklasse updaten

```
@Entity
@Document(collection = "products")
public class Product {
```

  - Annotation org.mongodb.morphia.annotations.Entity
    - Von QueryDSL benutzte Annotation zur Generierung der Objekte des Types `com.mysema.query.types.EntityPath`

SPRING DATA MONDODB

- Praxis 30 min
  - Erweitert Eure Spring-Anwendung:
    - Fügt für eine Domänenklasse das DAO-Pattern mit `MongoRepository` und `QueryDslPredicateExecutor`
    - implementiert das Anlegen und das Auslesen
    - schreibt JUnit-Tests dafür
    - Schreibt in Junit-Tests für QueryDSL
    - Für Integrationstests muss der MongoDB-Server laufen!

```
<plugin>
    <groupId>com.mysema.maven</groupId>
    <artifactId>apt-maven-plugin</artifactId>
    <version>${apt-maven-plugin.version}</version>
    <dependencies>
        <dependency>
            <groupId>com.mysema.querydsl</groupId>
            <artifactId>querydsl-apt</artifactId>
            <version>${querydsl.version}</version>
        </dependency>
    </dependencies>
    <executions>
        <execution>
            <phase>generate-sources</phase>
            <goals>
                <goal>process</goal>
            </goals>
            <configuration>
                <outputDirectory>
                     target/generated-sources
                </outputDirectory>
                <processor>
com.mysema.query.apt.morphia.MorphiaAnnotationProcessor
                </processor>
            </configuration>
        </execution>
    </executions>
</plugin>
```

```
<dependency>
    <groupId>com.mysema.querydsl</groupId>
    <artifactId>querydsl-mongodb</artifactId>
    <version>${querydsl.version}</version>
</dependency>
<dependency>
    <groupId>org.mongodb.morphia</groupId>
    <artifactId>morphia</artifactId>
    <version>0.109</version>
</dependency>
```

SPRING DATA MONDODB

- Praxis 10 min
  - Wie beurteilt Ihr die 3 Datenbankzugriffe?
  - Welche Vor- und Nachteile seht Ihr?
  - Welcher war am einfachsten?
  - Welcher am schwierigsten?

  - Diskutiert.

SPRING REST

- REST
  - Representational State Transfer
    - Resources
      - Vom System angeboten
    - Identifiers
      - Resources können eindeutig adressiert werden, bei HTTP über URI
    - Verbs
      - Jede Resource kann wohldefiniert abgefragt und manipuliert werden.
      - In HTTP: GET, PUT, POST, DELETE, HEAD, OPTIONS, TRACE, CONNECT
      - Eine Resource muss nicht alle Verbs unterstützen
    - Representations
      - Interaktion mit Resources durch Representations.
      - Definiert durch Media-Typ etwa `application/xml`, `application/json` oder `application/atom+xml`

SPRING REST

- REST
  - Representational State Transfer
    - Hypermedia
      - Representation einer Resource kann Links zu anderen Resourcen beinhalten
      - Links können zur Navigation benutzt werden
      - Navigation abhängig vom Zustand der Resource und den beinhalteten Links

```
GET http://127.0.0.1:8080/spring-jdbc/rest/hello
GET http://127.0.0.1:8080/spring-jdbc/rest/product/1
POST http://127.0.0.1:8080/spring-jdbc/rest/product/add
GET http://127.0.0.1:8080/spring-jdbc/rest/customer/1
```

SPRING REST

- REST
  - Verbs für HTTP
    - GET
      - fordert eine Repräsentation einer Ressource an, die durch eine URI eindeutig identifiziert wird
      - die Ressource wird nicht verändert
      - Idempotent

```
GET /spring-jdbc/rest/product/1 HTTP/1.1
Host: 127.0.0.1:8080
Accept: application/json
```

    - DELETE
      - Löscht die Resource, die über die URI identifiziert werden kann
      - Existiert diese Resource nicht, wird kein Fehler geliefert
      - Idempotent

```
DELETE /spring-jdbc/rest/product/1 HTTP/1.1
Host: 127.0.0.1:8080
```

SPRING REST

- REST
  - Verbs für HTTP
    - POST
      - Erzeugt eine oder mehrere Ressourcen
      - Ändert eine oder mehere Resourcen
      - Daten werden nicht in der URI mitangegeben, sondern hängen am Request an
      - die identifizierende URI wird als Antwort zurückgegeben
      - Nicht idempotent

```
POST /spring-jdbc/rest/product/add HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/xml
Content-Length: 242
…

<?xml version="1.0" encoding="UTF-8"?>
<product>
  <name>Apfel iPad</name>
  <description>Ipad by Apfel Inc</description>
  <price>12345</price>
</product>
```

SPRING REST

- REST
  - Verbs für HTTP
    - PUT
      - Ändert die Identifizierte Resource mit den angehängten Daten
      - Existiert die Resource noch nicht, wird sie angelegt
      - Antwort 201 bei Erzeugung, Antwort 200 bei Update
      - Idempotent

```
PUT /spring-jdbc/rest/product/123 HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/xml
Content-Length: 242
…

<?xml version="1.0" encoding="UTF-8"?>
<product>
  <name>Apfel iPad</name>
  <description>Ipad by Apfel Inc</description>
  <price>12345</price>
</product>
```

SPRING REST

- REST
  - Verbs für HTTP
    - HEAD
      - Analog GET, doch Antwort ohne Daten
      - Idempotent

```
HEAD /spring-jdbc/rest/product/1 HTTP/1.1
Host: 127.0.0.1:8080
```

    - OPTIONS
      - Prüft, welche Methoden für eine identifizierte Resource zur Verfügung stehen
      - Idempotent

```
OPTIONS /spring-jdbc/rest/product/1 HTTP/1.1
Host: 127.0.0.1:8080


HTTP/1.1 200 OK
Date: Tue, 27 Jan 2015 16:43:31 GMT+1
Allow: GET,HEAD,POST,OPTIONS,TRACE
Content-Length: 0
Content-Type: text/plain
```

SPRING REST

- Spring und REST
  - Webapp-basierend, Servlet-API 3.x
  - HTTP-Requests werden durch Controller bearbeitet
  - Annotation `@org.springframework.web.bind.annotation.RestController`

```
@RestController("helloController")
```

  - Annotation @org.springframework.stereotype.Controller
    - Annotierte Klasse ist Controller
  - Annotation `@org.springframework.web.bind.annotation.ResponseBody`
    - Returnwert einer Methode wird als Body einer Response gebunden
  - `@RestController` inkludiert `@Controller` und `@ResponseBody`

SPRING REST

- HelloWorld

```
@RestController("helloController")
public class HelloController {

  private static final DateTimeFormatter MDateTimeFormatter =
DateTimeFormatter.ofPattern("dd.MM.yyyy HH:mm:ss");

  @RequestMapping(value= "/hello", method= RequestMethod.GET)
  public String sayHello() {
    return "Hello World for Spring Data MongoDB @" +
            LocalDateTime.now().format(MDateTimeFormatter);
  }
}
```

```
GET http://127.0.0.1:8080/spring-data-mongo/rest/hello
```

SPRING REST

- HelloWorld
  - Annotation `@org.springframework.web.bind.annotation.RequestMapping`
    - `consumes` – welcher Mediatype kann verarbeitet werden z.B. `application/json`
    - `method` – HTTP-Methoden via Enum org.springframework.web.bind.annotation.RequestMethod z.B. GET
    - `name` – Name des Mappings
    - `value` – Endpunkt der URI
    - `produces` – welcher Mediatype wird zurückgegeben
    - `params` – die Parameter des Mappings

    - `java.util.Optional<T>` wird als Methodenparameter unterstützt

SPRING REST

- HelloWorld
  - ApplicationContext

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
       http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd">
   <import resource="classpath:spring/spring-mvc.xml"/>
   <context:annotation-config/>
    <context:component-scan base-package="com.jambit.workshop.jib.spring.data.mongo" />
</beans>

<!-- spring/spring-mvc.xml -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:mvc="http://www.springframework.org/schema/mvc"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd">
    <mvc:annotation-driven />
</beans>
```

SPRING REST

- HelloWorld
  - Spring MVC
    - MVC : Model-View-Controller
    - Deployed in Servlet-Engine
    - `org.springframework.web.servlet.DispatcherServlet`
      - Zentraler Dispatcher für HTTPRequests
      - Verteilt die HTTPRequests an registrierte Controller
      - Konfiguriert in der Serverkonfiguration – web.xml
      - Registrierung z.B. über Annotation `@Controller` und aktiviertem ComponentScan via `<mvc:annotation-driven />`
    - Interface `org.springframework.web.context.WebApplicationContext`
      - Konfiguration für eine Webapplikation
      - Hierarchisch: pro Applikation ein Rootcontext, jedes Servlet kann eigenen Childcontext haben
    - Klasse `org.springframework.web.context.ContextLoaderListener`
      - Startet Springs Root-`WebApplicationContext`

SPRING REST

- HelloWorld
  - Webapp-Konfiguration unter `/src/main/webapp/WEB-INF/web.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">

  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath*:spring/emptyContext.xml</param-value>
  </context-param>
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <servlet>
    <servlet-name>dispatcher4rest</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath*:spring/applicationContext.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>dispatcher4rest</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
</web-app>
```

SPRING REST

- HelloWorld
  - Maven-Konfiguration

```
<packaging>war</packaging>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
        <webXml>src/main/webapp/WEB-INF/web.xml</webXml>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
</plugin>
```

SPRING REST

- HelloWorld
  - Maven-Konfiguration

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>


<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
```

SPRING REST

- Praxis 30 min
  - Installiert Tomcat 8
  - Erweitert Eure Spring-Data-Anwendung:
    - Baut eine Helloworld-Controller ein
    - Endpunkt der URI /hello
    - GET
  - Deployed das generierte WAR im Tomcat

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

<packaging>war</packaging>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
        <webXml>src/main/webapp/WEB-INF/web.xml</webXml>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
</plugin>
```

SPRING REST

- Anbindung an Backend
  - ApplicationContext


```
<context:annotation-config/>
<context:component-scan base-package="com.jambit.workshop.jib.spring.data.mongo"/>
```

```
@RestController("productController")
public class ProductController {
  @Autowired
  @Qualifier("productManager")
  ProductManager mProductManager;
```

```
@Service("productManager")
public class ProductManagerImpl implements ProductManager {
  @Autowired
  @Qualifier("productRepository")
  ProductRepository mProductRepository;
```

```
@Repository("productRepository")
public interface  ProductRepository extends MongoRepository<Product, String>
```

SPRING REST

- Test des Controllers
  - Integrationstest

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath*:spring/testApplicationContext.xml"})
@WebAppConfiguration
public class ProductControllerTest extends AbstractJUnit4SpringContextTests {

   @Autowired
   @Qualifier("productController")
   ProductController mController;

   @Test
   public void testGetExistingProduct() {
     Product productToAdd = new Product();
     productToAdd.withDescription("test_description").withName("test_name").withPriceInCents(1000);
     assertFalse(productToAdd.hasValidId());
     Product added = mController.addProduct(productToAdd);
     assertNotNull(added);
     assertTrue(added.hasValidId());
     assertEquals("test_name", added.getProductName());
     assertEquals("test_description", added.getDescription());
     assertEquals(Integer.valueOf(1000), added.getPriceInCents());
     Product found = mController.getProduct(added.getOptionalId().get());
     assertEquals(added, found);
   }
}
```

SPRING REST

- Test des Controllers
  - Unittest mit Spring MockMvc

```
@RunWith(SpringJUnit4ClassRunner.class)
@ContextConfiguration(locations = {"classpath*:spring/testApplicationContext.xml"})
@WebAppConfiguration
public class MockProductControllerTest extends AbstractJUnit4SpringContextTests {

    static final MediaType APPLICATION_JSON_UTF8 = new MediaType(MediaType.APPLICATION_JSON.getType(),
MediaType.APPLICATION_JSON.getSubtype(), Charset.forName("utf8"));

    @Autowired
    WebApplicationContext mWebApplicationContext;
    @Autowired
    MockHttpSession session;
    @Autowired
    MockHttpServletRequest request;

    private MockMvc mMockMvcWithApplicationContext;

    @Before
    public void setup() {
        assertNotNull(mWebApplicationContext);
        assertTrue(mWebApplicationContext.containsBean("productController"));
        mMockMvcWithApplicationContext =
                    MockMvcBuilders.webAppContextSetup(mWebApplicationContext).build();
    }
```

SPRING REST

- Test des Controllers
  - Unittest mit Spring MockMvc

```
@Test
public void testProductsAdd() throws Exception {
    Product productToAdd = new Product();
    productToAdd.withDescription("test_description1").withName("test_name1").withPriceInCents(8888);
    assertFalse(productToAdd.hasValidId());
    ResultActions resultActions = mMockMvcWithApplicationContext.perform(post("/products/add")
            .contentType(APPLICATION_JSON_UTF8)
            .content(convertObjectToJson(productToAdd)));
    resultActions.andExpect(status().isCreated())
            .andExpect(content().contentTypeCompatibleWith(APPLICATION_JSON_UTF8))
            .andExpect(jsonPath("$.name").value("test_name1"))
            .andExpect(jsonPath("$.description").value("test_description1"))
            .andExpect(jsonPath("$.priceInCents").value(8888))
            .andExpect(jsonPath("$.id").exists())
            .andExpect(jsonPath("$.optionalId").exists())
            .andExpect(jsonPath("$.optionalId.present").value(true));
}
```

SPRING REST

- Test des Controllers
  - Little helper

```
private static String convertObjectToJson(final Product product) throws IOException {
    final org.json.simple.JSONObject.JSONObject jsonObject = new JSONObject();
    jsonObject.put("name", product.getProductName());
    jsonObject.put("description", product.getDescription());
    jsonObject.put("priceInCents", product.getPriceInCents());
    if (product.hasValidId()) {
        jsonObject.put("id", product.getId());
    }
    return jsonObject.toJSONString();
}

private static Product convertJsonToObject(final String json) throws ParseException {
    final org.json.simple.parser.JSONParser parser = new JSONParser();
    final JSONObject parsed = (JSONObject) parser.parse(json);
    return createProduct(parsed);
}
```

SPRING REST

- JSON
  - Jackson http://wiki.fasterxml.com/JacksonHome
  - Aktuelle Version 2.4.4
  - Bevorzugtes JSON-Framework von Spring

  - Annotation `@com.fasterxml.jackson.annotation.JsonProperty`
    - Mapping zwischen Attribut und JSON-Element
    - Notwendig, wenn keine eindeutige Zuordnung

```
@JsonProperty("name")
private String mName;

@JsonProperty("check")
private Boolean check;
public Boolean isCheck() { return check; }
```

SPRING REST

- JSON
  - Einbindung in Spring
    - ApplicationContext

```
<bean id="objectMapper"
      class="org.springframework.http.converter.json.Jackson2ObjectMapperFactoryBean"
      p:failOnEmptyBeans="true"
      p:autoDetectGettersSetters="false"
      p:indentOutput="true"
      p:findModulesViaServiceLoader="true"
      p:modulesToInstall="com.fasterxml.jackson.datatype.jdk8.Jdk8Module,
                          com.fasterxml.jackson.datatype.jsr310.JSR310Module"/>

<bean id="jsonMessageConverter"
      class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"
      p:objectMapper-ref="objectMapper"
      p:prettyPrint="true"/>

<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"
      p:messageConverters-ref="jsonMessageConverter"
      p:supportedMethods="GET, HEAD, POST, PUT, DELETE"/>
```

SPRING REST

- Parameter
  - `/products/{id}`
  - Annotation `@org.springframework.web.bind.annotation.PathVariable`
    - `value` – Name des Pfadparameters

```
@RequestMapping(value ="/products/{id}", method = RequestMethod.GET)
Product getProduct(@PathVariable("id") String id)
```

  - `/products/find?name=xyz`
  - Annotation `@org.springframework.web.bind.annotation.RequestParam`
    - `value` – Name des Requestparameters
    - `defaultValue` - als Fallback
    - `required` - boolean

```
@RequestMapping(value = ="/products/find", method = RequestMethod.GET)
public List<Product> findProductByName(@RequestParam(value ="name", required = true) final String name)
```

SPRING REST

- Praxis 60 min
  - Erweitert Eure Spring-Data-Anwendung:
    - Baut einen Controller für eine Domänenklasse ein
    - Abfrage nach einem dedizierten Objekt mit GET und Resultat in JSON
    - Abfrage nach allen Objekten mit GET und Resultat in JSON
    - Hinzufügen eines Objektes mit POST
    - Schreibt JUnit-Tests für Integration und Unit
  - Deployed das generierte WAR in Tomcat

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context-support</artifactId>
    <version>4.1.4.RELEASE</version>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>

<packaging>war</packaging>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <configuration>
        <webXml>src/main/webapp/WEB-INF/web.xml</webXml>
    </configuration>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-surefire-plugin</artifactId>
</plugin>
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-resources-plugin</artifactId>
</plugin>
```

VIELEN DANK!

WIR FREUEN UNS AUF DIE GEMEINSAME ARBEIT!

Thorsten Weber
thorsten.weber@jambit.com

Bis Morgen!

jambit
WHERE INNOVATION WORKS

SOFTWARE & SYSTEM DEVELOPER
INNOVATION PARTNER
COFFEE LOVER

Sitz: München, gegründet 1999
Ziel: 100% erfolgreiche Softwareprojekte

Geschäftsführer:
Peter F. Fellinger, Markus Hartinger

Erika-Mann-Straße 63
80636 München
Tel. +49.89.45 23 47-0

office@jambit.com
www.jambit.com