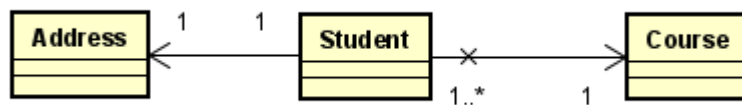


Übung Nr. 5
Implementierung von Persistence (Many-to-One and OneToOne Relationships)

Aufgabe 1 – Einen Studenten anlegen, einen Kurs auswählen

In dieser Aufgabe werden wir die folgenden Beziehungen implementieren:



- 1.1 Erstellen Sie ein neues Spring Boot-Starterprojekt. Stellen Sie dieses Mal sicher, dass Sie die Dependencies für Spring JPA, Validierung, Thymeleaf-Dialekt und H2 (Datenbank) einschließen.

Fügen Sie auch einige Dependencies hinzu wie:

```
<!-- https://mvnrepository.com/artifact/org.webjars/bootstrap -->
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>5.3.3</version>
</dependency>
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>webjars-locator</artifactId>
  <version>0.30</version>
</dependency>
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>jquery</artifactId>
  <version>3.7.1</version>
</dependency>
<dependency>
  <groupId>org.webjars.npm</groupId>
  <artifactId>jquery-mask-plugin</artifactId>
  <version>1.14.16</version>
</dependency>
```

- 1.2 Konfigurieren Sie die Datei application.properties wie unten:

```
spring.thymeleaf.cache=false
spring.thymeleaf.prefix=classpath:/templates/
spring.thymeleaf.suffix=.html
spring.messages.basename=messages
server.error.include-binding-errors=always
#DB
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.defer-datasource-initialization=true
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

1.3 Testen Sie den Zugriff auf die Datenbank:

Benutzer: sa; Passwort: password

Url: jdbc:h2:mem:testdb

<http://localhost:8080/h2-console>

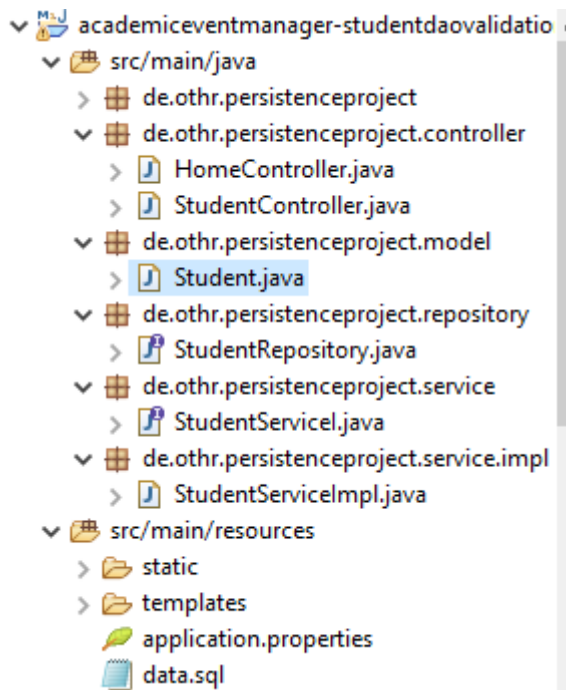
Weitere Informationen zur H2-Datenbank finden Sie unter diesem Link:

<https://www.baeldung.com/spring-boot-h2-database>

1.4 Erstellen Sie diesmal folgende Pakete:

- Controller
- Service and service.impl
- Repository
- Model

Wie in der folgenden Abbildung gezeigt:



1.5 Erstellen Sie die Modellklassen Student (ID, Name, E-Mail, Adresse, Kurs), Kurs (ID, Beschreibung) und Adresse (ID, Nummer, Straße, ZPL). Annotieren Sie sie mit @Entity, @Id, @GeneratedValue usw. Sie sollten die Schnittstelle Serializable implementieren.

1.6 Erstellen Sie eine GenderEnum-Enummerierung unter dem Paket „utils“:

```
public enum GenderEnum {

    MALE ("M"), FEMALE("F"), OTHER ("O");

    private final String displayValue;

    private GenderEnum(String displayValue) {
        this.displayValue = displayValue;
    }

    public String getDisplayValue() {
        return displayValue;
    }

}
```

Aktualisieren Sie die Klasse Student mit dem folgenden Attribut:

```
@Enumerated(EnumType.STRING)
private GenderEnum gender;
```

Generieren Sie die Getters und Setters.

1.7 Erstellen Sie die Beziehung @OneToOne zwischen Student und Adresse in der Klasse Student (unidirektional).

```
@OneToOne (cascade = CascadeType.ALL)  
@JoinColumn(name = "address_id", referencedColumnName = "id")  
private Address address;
```

1.8 Erstellen Sie die Beziehung @ManyToOne auch in der Klasse Student (unidirektional)

```
@ManyToOne  
@JoinColumn(name = "course_id", referencedColumnName = "id")  
private Course course;
```

1.9 Erstellen Sie die Datei data.sql im Ordner „resources“:

```
INSERT INTO course (description) VALUES ( 'Business Administration');
INSERT INTO course (description) VALUES ( 'Computer Science');
INSERT INTO course (description) VALUES ( 'Law');
INSERT INTO course (description) VALUES ( 'Mathematic');

INSERT INTO ADDRESS(HOUSE_NUMBER, STREET, ZPL) VALUES ('335', 'Ludwingstrasse', '93047');
INSERT INTO student (name, email, gender, address_id, course_id) VALUES ( 'Jonas Smith',
'j.smith@gmail.com', 'MALE', 1, 1);

INSERT INTO ADDRESS(HOUSE_NUMBER, STREET, ZPL) VALUES ('2', 'Markusplatz', '93047');
INSERT INTO student (name, email, gender, address_id, course_id) VALUES ( 'Diana Fischer',
'd.fischer@gmail.com', 'FEMALE', 2, 2);
```

Sie werden einen Fehler im Feld „HOUSE_NUMBER“ finden.

Versuchen Sie, das Problem zu lösen, indem Sie sich merken, welche Property in der Adressklasse annotiert werden soll.

1.10 Kopieren Sie die Ordner „fragments“ und „students“ und die Datei layout.html aus der Zip-Datei in Elo.

Im Ordner „Students“ sollten Sie folgende Views haben:

- Student-add
- Student-all
- Student-update
- Student-select-course

Achtung: In student-add.html, Sie sollten die folgende Zeile mit dem Namen Ihres Pakets anpassen.
<option th:each="genderOpt : \${T(de.othr.persistenceproject.utils.GenderEnum).values()}"

1.11 Erstellen Sie ein StudentRepository und ein CourseRepository Schnittstellen, die das MyBaseRepository erweitern.

```
public interface StudentRepositoryI extends MyBaseRepository<Student, Long>{

}

public interface CourseRepositoryI extends MyBaseRepository<Course, Long> {
    List<Course> findByDescriptionContainingIgnoreCase (String description);
}
```

Erstellen Sie ein StudentRepositoryImp und ein CourseRepositoryImp Interfaces, die das CrudRepository von JPA und die letzte Interfaces erweitern.

```

@Repository
public interface StudentRepositoryImp extends StudentRepositoryI,
CrudRepository<Student, Long>{

}

@Repository
public interface CourseRepositoryImp extends CourseRepositoryI,
CrudRepository<Course, Long>{

    List<Course> findByDescriptionContainingIgnoreCase (String description);

}

```

1.12 Erstellen Sie eine StudentServiceI-Schnittstelle unter dem Paket „service“ mit den folgenden CRUD-Operationen:

```

public interface StudentServiceI {

    List<Student> getAllStudents();

    Student saveStudent(Student student);

    Student getStudentById(Long id);

    Student updateStudent(Student student);

    void delete(Student student);

}

```

1.13 Erstellen Sie ein StudentServiceImpl, das StudentServiceI implementiert und ein StudentRepository eingefügt hat. Verwenden Sie das Repository, um die CRUD-Operationen auszuführen.

```

@Service
public class StudentServiceImpl implements StudentService {

    private StudentRepository studentRepository;

    public StudentServiceImpl(StudentRepository studentRepository) {
        super();
        this.studentRepository = studentRepository;
    }

    @Override
    public List<Student> getAllStudents() {
        // TODO Auto-generated method stub
        return (List<Student>) studentRepository.findAll();
    }

    @Override
    public Student saveStudent(Student student) {
        // TODO Auto-generated method stub

        return studentRepository.save(student);
    }

    @Override
    public Student getStudentById(Long id) {
        // TODO Auto-generated method stub
        return studentRepository.findById(id).get();
    }

    @Override
    public Student updateStudent(Student student) {
        // TODO Auto-generated method stub
        System.out.println(student.getGender()+"***");
        return studentRepository.save(student);
    }

    @Override
    public void delete(Student student) {
        // TODO Auto-generated method stub
        studentRepository.delete(student);
    }

}

```

1.14 Erstellen Sie eine CourseService-Schnittstelle unter dem Paket „service“ mit den folgenden CRUD-Operationen:

```

public interface CourseService {

    List<Course> getAllCourses();

    Course saveCourse(Course course);

    Course getCourseById(Long id);

    Course updateCourse(Course course);

    void delete(Course course);

}

```

1.15 Erstellen Sie ein CourseServiceImpl unter dem serviceimp-Paket, das CourseService implementiert und ein CourseRepository injiziert. Verwenden Sie das Repository, um die CRUD-Operationen auszuführen.

1.16 Erstellen Sie einen StudentController mit den folgenden Methoden:

GET /student/add
POST /student/add
GET /student/update/id
POST /student/update
GET /student/delete/id
GET /student/all

GET /student/course/select
POST /student/course/select
GET /student/course/select/id

Der Studenten-Controller muss ein StudentService- und ein CourseService-Objekt wie unten injiziert haben. Rufen Sie die Studentenservice-Instanz auf, um alle Methoden im Controller auszuführen.

```
@RequestMapping(value = "/student")
@Controller
public class StudentController {

    private StudentServiceI studentService;
    private CourseServiceI courseService;

    public StudentController(StudentServiceI studentService,
                             CourseServiceI courseService) {
        super();
        this.studentService = studentService;
        this.courseService = courseService;
    }

    .....
}
```