VENTSPILS AUGSTSKOLA in collaboration with accenture

Seminar 6

Spring JPA usage with database

Simple application - to learn how to use Spring JPA technology to automate table creation in DB and to use JPA advantages of simple query creation.

Assignment 0 (DB tables and ERD)

First of all it is necessary to create database model which consists of DB tables and ERD (Entity Relationship Diagram). Take a look to example:

Tables:

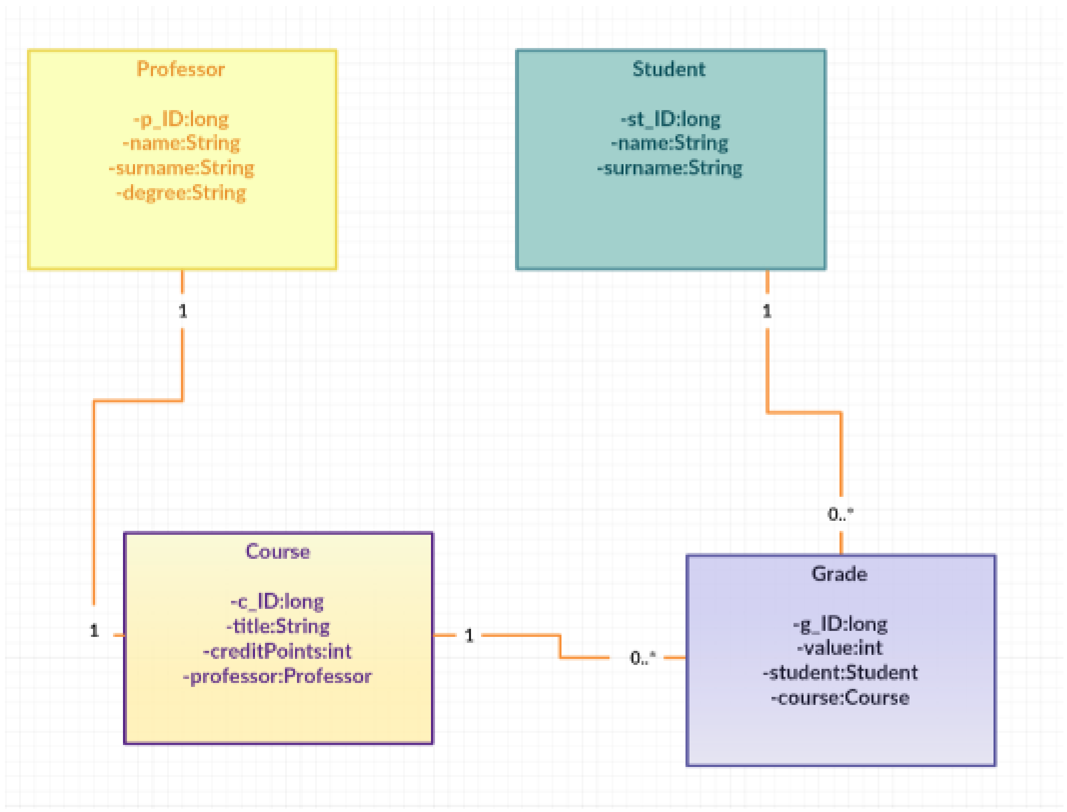
Table: Student		
ID_st	Name	Surname
1	Janis	Berzins
2	Baiba	Jauka
3	Liga	Forsa

Table: Professor			
ID_p	Name	Surname	Title
1	Karina	Skirmante	Mg.sc.comp
2	Estere	Vitola	Mg.sc.comp
3	Gints	Neimanis	Mg.sc.comp

Table: Subject			
ID_s	Title	CP	ID_p
1	Matematika	8	2
2	JAVA	4	1
3	Datubazes	4	3

Table: Grade			
ID_g	ID_s	ID_st	value
1	1	1	6
2	2	1	8
3	3	1	4
4	1	2	7
5	3	2	8
6	1	3	5
7	2	3	10

ERD:



Assignment 1 (Model classes)

Create model classes related to DB model (if it is possible please copy source files of mentioned classes from Seminar 5 - Student, Professor, Course, Grade).

Assignment 2 (Model classes with JPA annotations)

Update model classes to create linkage between model and db tables.

- Add new dependency of JPA in maven configuration file (pom.xml)
- Add new dependency of database driver in maven configuration file (pom.xml). In simple case it is possible to use H2 database (in-memory database). Take a look here: https://www.h2database.com/html/main.html. It is possible to see H2 database in localhost:8080/h2-console (just add an enabling code line

```
spring.h2.console.enabled=true
```

  in Your project *application.properties* file
- Add JPA annotation to each model class. Short description of annotations below. (More info here: https://www.oracle.com/technetwork/middleware/ias/toplink-jpa-annotations-096251.html)

  - @Entity - designates a plain old Java object (POJO) class as an entity and make it eligible for JPA services.
  - @Table - defines that all the persistent fields of an entity are stored in a single database table
  - @Column - defines that each of an entity's persistent attributes is stored in a database table column
  - @Id - designates one or more persistent fields or properties as the entity's primary key.
  - @GeneratedValue - is responsible for supplying and setting entity identifiers

  Relationship Mappings

  - @OneToOne - defines a OneToOne mapping for a single-valued association to another entity that has one-to-one multiplicity and infers the associated target entity from the type of the object being referenced.
  - @OneToMany - defines a OneToMany mapping for a many-valued association with one-to-many multiplicity.
  - @ManyToOne - defines a ManyToOne mapping for a single-valued association to another entity class that has many-to-one multiplicity.
  - @ManyToMany - defines a ManyToMany mapping for a many-valued association with many-to-many multiplicity.

  Others

  - @JoinColumn - (in an entity association) assumes a database schema based on existing names (such as field or property names) so that it can automatically determine the single join column (the column that contains the foreign key) to use
  - @JoinTable - uses a join table when mapping entity associations on the owning side of a many-to-many association, or in a unidirectional one-to-many association.

VENTSPILS AUGSTSKOLA in collaboration with accenture

Assignment 3 (Repositories and new queries)

For each table it is necessary to create repository which is linkage between object in java code and DB sql queries. More about repo here:
https://docs.spring.io/spring-data/jpa/docs/1.5.0.RELEASE/reference/html/jpa.repositories.html
https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html

- Create an interface which extends interface of repository implementation (CRUD Repo, JPA Repo) and specify the model class related to this repo.

  ```
  public interface StudentRepo extends CrudRepository<Student, Long>
  ```

  `Student` – defines that repo will store information about Student table
  `Long` – defines that Student id is a Long type

- Implemented methods in CRUD repository interface (these are already exist and no need to write the implementation of Your own)
  - `S save(S entity)` - saves the given entity;
  - `T findOne(ID primaryKey)` - returns the entity identified by the given id;
  - `Iterable<T> findAll()` - returns all entities;
  - `Long count()` - returns the number of entities
  - `void delete(T entity)` - deletes the given entity
  - `boolean exists(ID primaryKey)` - indicates whether an entity with the given id exists
  - Etc.
- It is possible to create Your queries (more:https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods)
  - In simple case - declare query methods on the interface and JPA automates the implementation of SQL query - no need to write a SQL code. Standard CRUD functionality repositories usually have queries on the underlying datastore
    `List<Student> findBySurname(String surname);`
    Or
    `Course findByTitle(String title);`
  - Create SQL to define a query - use @Query annotation where define the SQL query

    ```
    @Query(value = "SELECT s FROM StudentTable s")
    List<Student> findAllStudents();
    ```

- Repository usage

- Create object of repository and use autowired annotation (this annotation allows Spring to resolve and inject collaborating beans into your bean. Once annotation injection is enabled, autowiring can be used on properties, setters, and constructors).

```
@Autowired
StudentRepo studentRepo;
```

- Create an object of Student and save it in DB using repo object

```
Student s1 = new Student("Janis", "Berzins", 18);
studentRepo.save(s1);
```

- Select all students from DB table

```
ArrayList<Student> allStudentFromDB = (ArrayList<Student>)
studentRepo.findAll();
```

- Select “Janis” students from DB table

```
Student s = studentRepo.findByName("Janis");
```

Assignment 4 (Service)

- Create Service class or classes with service methods related to data input, data filtering, etc. Possible services functions:
  - createDataForTesting() - create data for IS testing(multiple Student, Professor, Subject and Grade objects) and save in DB using repositories;
  - selectAllStudents() - return all students;
  - selectAllProfessors() - return all professors;
  - selectAllSubjects() - return all subjects;
  - selectAllGrades() - return all grades;
  - selectGradesByStudent() - return all grades of specific student;
  - selectCoursesByStudent() - return all courses where specific student is involved;
  - selectCoursesByProfessor() - return all courses of specific professor;
  - calculateAVGGradeInCourse() - return average grade in specific course;
  - etc.

Assignment 5 (Controller)

- Create Controller class or classes with controller function where Service functions are called and results are passed to View. Possible controller functions
  - @GetMapping(value = "/insertTestData") - where Service function createDataForTesting() is called

- @GetMapping(value="/info/showAllStudents") - show all students in html, using Service function selectAllStudents()
- @GetMapping(value="info//showAllProfessors") - show all professors in html, using Service function selectAllProfessors()
- @GetMapping(value="/info/showAllSubjects") - show all subjects in html, using Service function selectAllSubjects()
- @GetMapping(value="/info/showAllGrades") - show all grades in html, using Service function selectAllGrades()
- @GetMapping(value = "/filter//filterGradesByStudent") and @PostMapping(value = "/filter/filterGradesByStudent) - define student name and surname in html and get all grades for defined student, using Service function selectGradesByStudent()
- @GetMapping(value = "/filter//filterCoursesByStudent") and @PostMapping(value = "/filter/filterCoursesByStudent) - define student name and surname in html and get all courses where student is registered, using Service function selectCoursesByStudent()
- @GetMapping(value = "/filter//filterCoursesByProfessor") and @PostMapping(value = "/filter/filterCoursesByProfessor) - define profesor name and surname in html and get all courses lead by professor, using Service function selectCoursesByProfessor()
- @GetMapping(value = "/calculate/calculateAvgGradeInCourse") and @PostMapping(value = "/calculate/calculateAvgGradeInCourse) - define course title in html and calculate average grade of defined course, using Service function calculateAVGGradeInCourse()
- etc., for example - @GetMapping(value = "/filter//filterGradesByStudent/{id}"), @GetMapping(value = "/filter//filterTop3StudentsByGrades”)

*Assignment 5 (ManyToMany relations)

Change relation from OneToOne to ManyToMany between Subjects table and Professors tables. Please update all code related to this change.

Good Luck!