Follow-up Assignment:
Modeling a University Domain with JPA Relationships

January 6, 2026

1. Project preparation

   - Create a new project with Gradle (Kotlin) as the build system.
     - Open the file `build.gradle.kts` and replace its contents with the contents of the `build.gradle.kts` file attached to this assignment.
     - While still in `build.gradle.kts`, run "Sync Gradle Changes" (*Ctrl+Shift+O*).
   - In the directory `src -> main -> java`, create the package `logic` and place there the classes below that describe the JPA logic.
   - In the project directory, create a directory in which the database will be stored. Then create the database similarly to how it was done in task $10 - 01$, but do not create any tables.
   - In the directory `src -> main -> resources`, create the directory `META-INF`, then copy the file `persistence.xml` into it and complete it.

2. Declare a JPA entity by defining a public class `Address` representing an address in the domain model.

   - Mark the class with the annotation `@Entity`, so it is managed by the JPA mechanism.
   - Specify the database table name using the annotation `@Table(name = "addresses")`.

   (a) Define the entity identifier field.
   - Add a field `id` of type `Long`.
   - Mark the field with `@Id` to indicate the primary key.
   - Configure automatic identifier value generation using `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

   (b) Add the basic fields describing the address.
   - Define a field `street` of type `String` that stores the street name.
   - Define a field `city` of type `String` that stores the city name.
   - Define a field `postalCode` of type `String` that stores the postal code.

   (c) Configure the one-to-one relationship as the inverse side of the association with the student.

   - Add a field `student` of type `Student`.
   - Mark it with `@OneToOne(mappedBy = "address")`, indicating that the foreign key is located on the `Student` entity side.
   - Emphasize that the `Address` entity does not own the relationship and does not contain the foreign key.

   (d) Provide the standard structural elements of the class.
   - Add constructors enabling entity object creation.
   - Provide getters and setters for all fields to enable access to and modification of the entity state.

3. Declare a JPA entity by defining a public class `Department` representing a department (faculty) in the domain model.
   - Mark the class with the annotation `@Entity`, so it is managed by the JPA mechanism.
   - Specify the database table name using the annotation `@Table(name = "departments")`.

   (a) Define the entity identifier field.
   - Add a field `id` of type `Long`.
   - Mark the field with `@Id` to indicate the primary key.
   - Configure automatic identifier value generation using `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

   (b) Add the basic fields describing the department.
   - Define a field `name` of type `String` that stores the full name of the department.
   - Define a field `code` of type `String` that stores the department abbreviation, for example `"INF"` or `"MAT"`.

   (c) Configure the one-to-many relationship between the department and students.
   - Add a field `students` of type `List<Student>` and initialize it as an empty list.
   - Mark the relationship with `@OneToMany(mappedBy = "department")`, indicating that the owning side of the relationship is the `Student` entity.
   - Emphasize that one department may be associated with multiple students.

   (d) Configure the one-to-many relationship between the department and instructors.
   - Add a field `instructors` of type `List<Instructor>` and initialize it as an empty list.
   - Mark the relationship with `@OneToMany(mappedBy = "department")`, indicating that the owning side of the relationship is the `Instructor` entity.
   - Emphasize that one department may be associated with multiple instructors.

4. Declare a JPA entity by defining a public class `Instructor` representing an instructor in the domain model.
   - Mark the class with the annotation `@Entity`, so it is managed by the JPA mechanism.

- Specify the database table name using the annotation `@Table(name = "instructors")`.

(a) Define the entity identifier field.
   - Add a field `id` of type `Long`.
   - Mark the field with `@Id` to indicate the primary key.
   - Configure automatic identifier value generation using `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

(b) Add the basic fields describing the instructor.
   - Define a field `firstName` of type `String` that stores the first name.
   - Define a field `lastName` of type `String` that stores the last name.
   - Define a field `title` of type `String` that stores the academic title, for example `dr` or `prof`.
   - Define a field `email` of type `String` that stores the e-mail address.

(c) Configure the many-to-one relationship between the instructor and the department.
   - Add a field `department` of type `Department`.
   - Mark the relationship with `@ManyToOne(fetch = FetchType.LAZY)`, so the department data is loaded lazily.
   - Specify the foreign key column using `@JoinColumn(name = "department_id")`.

(d) Configure the one-to-many relationship between the instructor and courses.
   - Add a field `courses` of type `List<Course>` and initialize it as an empty list.
   - Mark the relationship with `@OneToMany(mappedBy = "instructor", cascade = CascadeType.ALL)`.
   - Ensure that operations on the instructor cascade to associated courses.

5. Declare a JPA entity by defining a public class `Course` representing a course in the domain model.

   - Mark the class with the annotation `@Entity`, so it is managed by the JPA mechanism.
   - Specify the database table name using the annotation `@Table(name = "courses")`.

(a) Define the entity identifier field.
   - Add a field `id` of type `Long`.
   - Mark the field with `@Id` to indicate the primary key.
   - Configure automatic identifier value generation using `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

(b) Add the basic fields describing the course.
   - Define a field `name` of type `String` that stores the course name.
   - Define a field `code` of type `String` that stores the course code, for example `"PRG-101"`.
   - Define a field `ects` of type `Integer` that specifies the number of ECTS points.

- Define a field `maxStudents` of type `Integer` that specifies the maximum number of enrolled students.

(c) Configure the many-to-one relationship between the course and the instructor.

- Add a field `instructor` of type `Instructor`.
- Mark the relationship with `@ManyToOne(fetch = FetchType.LAZY)`, so the instructor data is loaded lazily.
- Specify the foreign key column using `@JoinColumn(name = "instructor_id")`.

(d) Configure the one-to-many relationship between the course and enrollments.

- Add a field `enrollments` of type `List<Enrollment>` and initialize it as an empty list.
- Mark the relationship with `@OneToMany(mappedBy = "course", cascade = CascadeType.ALL, orphanRemoval = true)`.
- Ensure cascading of operations and automatic removal of orphaned enrollments.

(e) Add a helper method providing access to the students enrolled in the course.

- Implement a method `getStudents()`, which transforms the `Enrollment` list into a `Student` list.
- Use a stream to map enrollments to students and collect them into a new list.

6. Declare a JPA entity by defining a public class `Student` representing a student in the domain model.

- Mark the class with the annotation `@Entity`, so it is managed by the JPA mechanism.
- Specify the database table name using the annotation `@Table(name = "students")`.

(a) Define the entity identifier field.

- Add a field `id` of type `Long`.
- Mark the field with `@Id` to indicate the primary key.
- Configure automatic identifier value generation using `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

(b) Add the basic fields describing the student.

- Define a field `firstName` of type `String` that stores the first name.
- Define a field `lastName` of type `String` that stores the last name.
- Define a field `indexNumber` of type `String` that stores the student record number.
- Mark the field `indexNumber` with `@Column(unique = true)` to ensure uniqueness in the database.

(c) Configure the one-to-one relationship between the student and the address as the owning side.

- Add a field `address` of type `Address`.

  - Mark the relationship with `@OneToOne(cascade = CascadeType.ALL, orphanRemoval = true)`.
  - Specify the foreign key column using `@JoinColumn(name = "address_id")`.
  - Emphasize that the foreign key is stored in the `students` table.

(d) Configure the many-to-one relationship between the student and the department.

  - Add a field `department` of type `Department`.
  - Mark the relationship with `@ManyToOne(fetch = FetchType.LAZY)`, so the department data is loaded lazily.
  - Specify the foreign key column using `@JoinColumn(name = "department_id")`.

(e) Configure the one-to-many relationship between the student and enrollments.

  - Add a field `enrollments` of type `List<Enrollment>` and initialize it as an empty list.
  - Mark the relationship with `@OneToMany(mappedBy = "student", cascade = CascadeType.ALL, orphanRemoval = true)`.
  - Ensure cascading of operations and automatic removal of orphaned enrollments.

(f) Add a helper method enabling student enrollment in a course.

  - Implement a method `enrollIn(Course course)`, which creates a new `Enrollment` object associated with the current student and the provided course.
  - Add the new enrollment to the `enrollments` collection.

(g) Add a helper method enabling reading the courses the student is enrolled in.

  - Implement a method `getCourses()`, which maps `Enrollment` objects to `Course` objects.
  - Return the list of courses obtained by processing the enrollments collection.

7. Define the enum type `EnrollmentStatus` describing possible enrollment states.

   - Add values `ACTIVE`, `DROPPED`, and `COMPLETED`.
   - Use this type to clearly represent the current state of the student–course relationship.

8. Declare a JPA entity by defining a public class `Enrollment` representing a student's enrollment in a course.

   - Mark the class with the annotation `@Entity`, so it is managed by JPA.
   - Specify the database table name using `@Table(name = "enrollments")`.

(a) Define the entity identifier field.

  - Add a field `id` of type `Long`.
  - Mark the field with `@Id` to indicate the primary key.
  - Configure automatic identifier value generation using `@GeneratedValue(strategy = GenerationType.IDENTITY)`.

(b) Configure the many-to-one relationship between the enrollment and the student.

- Add a field `student` of type `Student`.
- Mark the relationship with `@ManyToOne(fetch = FetchType.LAZY)`.
- Specify the foreign key column `student_id` with `nullable = false`.

(c) Configure the many-to-one relationship between the enrollment and the course.

- Add a field `course` of type `Course`.
- Mark the relationship with `@ManyToOne(fetch = FetchType.LAZY)`.
- Specify the foreign key column `course_id` with `nullable = false`.

(d) Add additional fields describing the state and progress of the enrollment.

- Define a field `enrollmentDate` of type `LocalDate` storing the date of enrollment.
- Define a field `grade` of type `Double` storing the final grade.
- Define a field `completed` of type `Boolean` indicating whether the course has been completed.

(e) Define the enrollment status field as an enum type.

- Add a field `status` of type `EnrollmentStatus`.
- Mark the field with `@Enumerated(EnumType.STRING)` so the enum value is stored as text.

(f) Provide a no-argument constructor required by JPA.

- Define a public no-parameter constructor.

(g) Implement a constructor that initializes an enrollment for a student in a course.

- Accept `Student` and `Course` objects as parameters.
- Assign the provided entities to the corresponding fields.
- Set `enrollmentDate` to the current date.
- Set `status` to `EnrollmentStatus.ACTIVE`.
- Set `completed` to `false`.

(h) Implement the methods `getStudent` and `getCourse` that return the corresponding values.

9. Open the file `persistence.xml` and extend it with all entity classes defined in the program (a *TODO* section has been prepared for this purpose).

10. For testing purposes, create a class `Test` and in its `main` method add the statement `EntityManagerFactory emf = Persistence.createEntityManagerFactory("pjUniTest");`. Then run the program and verify in the database that the tables have been created. Present the result to the instructor.

11. Based on the code from assignment 12, prepare a test program showing how to create objects of all classes and then map them into the corresponding tables. Pay particular attention to relationship handling between objects and the database behavior.