@OneToMany

1. Spring Boot Application with "Course" and "Student" Entities:

Overview:

Create a Spring Boot application with two entities: "Course" and "Student." A course can have multiple students enrolled in it, while each student can be enrolled in only one course. Implement a one-to-many bidirectional mapping between these entities using Spring JPA. Utilize JPQL for retrieving course details along with enrolled students and handle `DuplicateCourseException` and `InvalidDataException`.

Functional Requirements:

Project Structure:

Create folders named `controller`, `model`, `repository`, `exception`, and `service` inside `WORKSPACE/springapp/src/main/java/com/example/springapp`.

Model Classes:

- Course:
  - Attributes:
    - `courseId` - int (auto-generated primary key)
    - `courseName` - String
    - `courseCode` - String
    - `students` - List<Student> (OneToMany, mappedBy = "course", @JsonManagedReference)

- Student:
  - Attributes:
    - `studentId` - int (auto-generated primary key)
    - `name` - String
    - `email` - String
    - `course` - Course (ManyToOne, @JsonBackReference)

Repository Interfaces:

- `CourseRepo`
- `StudentRepo`

Service Interfaces and Implementations:

- Interfaces: `CourseService` and `StudentService`
- Implementations: `CourseServiceImpl` and `StudentServiceImpl`

Controllers:

- `CourseController`

- `StudentController`

Custom Exceptions:

- `DuplicateCourseException`

- `InvalidDataException`

API Endpoints:

1. POST - /course

- Create a new course. Returns `201 Created` if successful, or `500 Internal Server Error` if a course with the same name already exists.

- Validation:

  - `courseName` must have at least 3 characters.

  - `courseCode` must be unique across all courses.

  - `courseCode` must be alphanumeric with no special characters.

  - InvalidDataException: If the `courseName` length is less than 3 or `courseCode` contains invalid characters.

2. POST - /student/{courseId}/enroll

- Assign a student to a course. Returns `201 Created` if successful, or `500 Internal Server Error` if the student is already enrolled in another course.

3. GET - /course/{courseId}

- Retrieve course details, including all enrolled students. Returns `200 OK` if successful, or `404 Not Found` if the course doesn't exist.

4. GET - /student

- Retrieve a list of all students. Returns `200 OK` if successful, or `404 Not Found` if no students exist.

5. GET - /student/sorted

- Retrieve a list of all students sorted by `name` in ascending order. Returns `200 OK` if successful, or `404 Not Found` if no students exist.

6. DELETE - /student/{studentId}

- Delete a student by their ID. Returns `200 OK` on successful deletion or `404 Not Found` if the student does not exist.

7. UPDATE - /course/{courseId}

- Update course details. Returns `200 OK` on successful update, or `500 Internal Server Error` if the course data is invalid.

- Validation:

  - `courseName` must have at least 3 characters.

  - `courseCode` must not contain special characters.

  - InvalidDataException: If the `courseName` length is less than 3 or if a negative number is passed for `budget` if applicable.

2. Spring Boot Application with "Department" and "Employee" Entities:

Overview:

Create a Spring Boot application with two entities: "Department" and "Employee." A department can have multiple employees, but each employee belongs to only one department. Implement a one-to-many bidirectional mapping between these entities using Spring JPA. Use JPQL to retrieve all employees belonging to a particular department, and handle `DuplicateDepartmentException`, `InvalidDataException`, and `NegativeSalaryException`.

Functional Requirements:

Project Structure:

Create folders named `controller`, `model`, `repository`, `exception`, and `service` inside `WORKSPACE/springapp/src/main/java/com/company/springapp`.

Model Classes:

- Department:
  - Attributes:
    - `departmentId` - int (auto-generated primary key)
    - `departmentName` - String
    - `location` - String
    - `employees` - List<Employee> (OneToMany, mappedBy = "department", @JsonManagedReference)

- Employee:
  - Attributes:
    - `employeeId` - int (auto-generated primary key)
    - `name` - String
    - `position` - String
    - `salary` - double
    - `department` - Department (ManyToOne, @JsonBackReference)

Repository Interfaces:

- `DepartmentRepo`
- `EmployeeRepo`

Service Interfaces and Implementations:

- Interfaces: `DepartmentService` and `EmployeeService`

- Implementations: `DepartmentServiceImpl` and `EmployeeServiceImpl`

Controllers:

- `DepartmentController`

- `EmployeeController`

Custom Exceptions:

- `DuplicateDepartmentException`

- `InvalidDataException`

- `NegativeSalaryException`

API Endpoints:

1. POST - /department

- Create a new department. Returns `201 Created` if successful, or `500 Internal Server Error` if the department name already exists.

- Validation:

  - `departmentName` must have at least 3 characters.

  - InvalidDataException: If the `departmentName` length is less than 3.

2. POST - /employee/{departmentId}/assign

- Assign an employee to a department. Returns `201 Created` if successful, or `500 Internal Server Error` if the employee is already assigned to another department.

- Validation:

  - Employee's `salary` should not be negative.

  - NegativeSalaryException: If the salary is negative.

3. GET - /department/{departmentId}

- Retrieve department details, including all employees assigned. Returns `200 OK` if successful, or `404 Not Found` if the department does not exist.

4. GET - /employee

- Retrieve a list of all employees. Returns `200 OK` if successful, or `404 Not Found` if no employees exist.

5. GET - /employee/position/{position}

- Retrieve a list of all employees by position. Returns `200 OK` if successful, or `404 Not Found` if no employees match the position.

6. DELETE - /employee/{employeeId}

- Delete an employee by their ID. Returns `200 OK` on successful deletion or `404 Not Found` if the employee does not exist.

7. UPDATE - /department/{departmentId}

- Update department details. Returns `200 OK` on successful update, or `500 Internal Server Error` if the department name is invalid.

- Validation:

  - `departmentName` must have at least 3 characters.

  - InvalidDataException: If the `departmentName` length is less than 3.

3. Spring Boot Application with "Library" and "Book" Entities:

Overview:

Create a Spring Boot application with two entities: "Library" and "Book." A library can contain multiple books, but each book is assigned to a single library. Implement a one-to-many bidirectional mapping between these entities using Spring JPA. Use JPQL for retrieving all books in a specific library and handle `DuplicateBookException`, `InvalidDataException`, and `NegativePriceException`.

Functional Requirements:

Project Structure:

Create folders named `controller`, `model`, `repository`, `exception`, and `service` inside `WORKSPACE/springapp/src/main/java/com/library/springapp`.

Model Classes:

- Library:
  - Attributes:
    - `libraryId` - int (auto-generated primary key)
    - `libraryName` - String
    - `location` - String
    - `books` - List<Book> (OneToMany, mappedBy = "library", @JsonManagedReference)

- Book:
  - Attributes:
    - `bookId` - int (auto-generated primary key)
    - `title` - String
    - `author` - String
    - `price` - double
    - `library` - Library (ManyToOne, @JsonBackReference)

Repository Interfaces:

- `LibraryRepo`
- `BookRepo`

Service Interfaces and Implementations:

- Interfaces: `LibraryService` and `BookService`

- Implementations: `LibraryServiceImpl` and `BookServiceImpl`

Controllers:

- `LibraryController`

- `BookController`

Custom Exceptions:

- `DuplicateBookException`

- `InvalidDataException`

- `NegativePriceException`

API Endpoints:

1. POST - /library

- Create a new library. Returns `201 Created` if successful, or `500 Internal Server Error` if a library with the same name already exists.

- Validation:

  - `libraryName` must have at least 3 characters.

  - InvalidDataException: If the `libraryName` length is less than 3.

2. POST - /book/{libraryId}/add

- Add a book to a library. Returns `201 Created` if successful, or `500 Internal Server Error` if the book already exists in the library.

- Validation:

  - `price` must not be negative.

  - NegativePriceException: If the price is negative.

3. GET - /library/{libraryId}

- Retrieve library details, including all books. Returns `200 OK` if successful, or `404 Not Found` if the library does not exist.

4. GET - /book

- Retrieve all books. Returns `200 OK` if successful, or `404 Not Found` if no books exist.

5. GET - /book/author/{authorName}

- Retrieve books written by a specific author. Returns `200 OK` if successful, or `404 Not Found` if no books are found.

6. DELETE - /book/{bookId}

- Delete a book by its ID. Returns `200 OK` on successful deletion or `404 Not Found` if the book does not exist.

7. UPDATE - /library/{libraryId}

- Update library details. Returns `200 OK` on successful update, or `500 Internal Server Error` if the library data is invalid.

- Validation:

  - `libraryName` must have at least 3 characters.

  - InvalidDataException: If the `libraryName` length is less than 3.