@OneToOne

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

Functional Requirements:
Create a Spring Boot application with two entities: "Student" and "StudentProfile."
- Each Student can have one StudentProfile, and each StudentProfile can be assigned to one Student.
- Implement a bidirectional one-to-one mapping between these entities using Spring Data JPA.

Project Structure:
- Create folders named `controller`, `model`, `repository`, and `service` inside the `WORKSPACE/springapp/src/main/java/com/example/springapp`.

Inside the `controller` folder:
- StudentController:
  - Handles requests related to the `Student` entity.
- StudentProfileController:
  - Handles requests related to the `StudentProfile` entity.

Inside the `model` folder:
- Student Entity:
  - `studentId` - int (auto-generated primary key)
  - `name` - String
  - `major` - String
  - `email` - String
  - `studentProfile` - `StudentProfile` (@OneToOne, mappedBy = "student", @JsonManagedReference)

- StudentProfile Entity:
  - `profileId` - int (auto-generated primary key)
  - `profilePicture` - String
  - `bio` - String
  - `student` - `Student` (@OneToOne, @JoinColumn, @JsonBackReference)

Inside the `repository` folder:
- StudentRepo:
  - Extend `JpaRepository<Student, Integer>`.
- StudentProfileRepo:
  - Extend `JpaRepository<StudentProfile, Integer>`.

Inside the `service` folder:
- StudentService and StudentProfileService:
  - Create interfaces with necessary methods like `createStudent()`, `getStudent()`, `assignProfileToStudent()`.
- StudentServiceImpl and StudentProfileServiceImpl:
  - Implement the above methods, handling exceptions and validations.

API Endpoints:
1. POST - /api/student
  - Request: JSON with `name`, `major`, `email`.
  - Response: Returns `201 Created` with the `Student` object or `500 Internal Server Error`.
  - Validation:
    - If a student with the same email already exists, return `409 Conflict` with the error message: `Student with email {email} already exists!`.

2. POST - /api/student/{studentId}/profile
  - Request: JSON with `profilePicture`, `bio`.
  - Response: Returns `201 Created` with the `StudentProfile` object if the profile is successfully assigned or `500 Internal Server Error`.
  - Validation:
- If the student already has a profile, return `409 Conflict` with the message: `Student already has an assigned profile!`.

3. GET - /api/student/{studentId}
- Response: Returns `200 OK` with the `Student` object, including `StudentProfile` details, if found. Otherwise, `404 Not Found`.

4. GET - /api/student
  - Response: Returns a list of all students in `200 OK`. The list should be sorted by:
    - Ascending order of `name`.
    - Descending order of `major`.
  - Custom Query in the repository to implement the sorting.

5. GET - /api/student/sort-name-desc
- Response: Returns a list of all students in `200 OK`, sorted in descending order of `name`.
- @Query Annotation can be used to query sorted results.

6. GET - /api/studentprofile/bio/{bio}
- Response: Returns a list of `StudentProfile` objects based on the specified `bio`, in `200 OK`, or `404 Not Found` if no profile matches.

7. DELETE - /api/studentprofile/{profileId}
- Response: Returns `200 OK` with the message `StudentProfile {profileId} deleted successfully` or `404 Not Found` if the profile is not found.

Exception Handling:
- DuplicateEmailException: Thrown when an email already exists for another student.
- ProfileAlreadyAssignedException: Thrown when trying to assign a profile to a student who already has one.

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

Functional Requirements:
Create a Spring Boot application with two entities: "Employee" and "EmployeeBadge."
- Each Employee can have one EmployeeBadge, and each EmployeeBadge can be assigned to one Employee.
- Implement a bidirectional one-to-one mapping between these entities using Spring Data JPA.

Project Structure:
- Create folders named `controller`, `model`, `repository`, and `service` inside the `WORKSPACE/springapp/src/main/java/com/company/springapp`.

Inside the `controller` folder:
- EmployeeController:
  - Handles requests related to the `Employee` entity.
- EmployeeBadgeController:
  - Handles requests related to the `EmployeeBadge` entity.

Inside the `model` folder:
- Employee Entity:
  - `employeeId` - int (auto-generated primary key)
  - `name` - String
  - `designation` - String
  - `email` - String
  - `employeeBadge` - `EmployeeBadge` (@OneToOne, mappedBy = "employee", @JsonManagedReference)

- EmployeeBadge Entity:
  - `badgeId` - int (auto-generated primary key)
  - `badgeNumber` - String
  - `issueDate` - String
  - `employee` - `Employee` (@OneToOne, @JoinColumn, @JsonBackReference)

Inside the `repository` folder:
- EmployeeRepo:
  - Extend `JpaRepository<Employee, Integer>`.
- EmployeeBadgeRepo:
  - Extend `JpaRepository<EmployeeBadge, Integer>`.

Inside the `service` folder:
- EmployeeService and EmployeeBadgeService:
  - Create interfaces with necessary methods like `createEmployee()`, `getEmployee()`, `assignBadgeToEmployee()`.
- EmployeeServiceImpl and EmployeeBadgeServiceImpl:
  - Implement the above methods, handling exceptions and validations.

API Endpoints:
1. POST - /api/employee
  - Request: JSON with `name`, `designation`, `email`.
  - Response: Returns `201 Created` with the `Employee` object or `500 Internal Server Error`.
  - Validation:
    - If an employee with the same email exists, return `409 Conflict` with the message: `Employee with email {email} already exists!`.

2. POST - /api/employee/{employeeId}/badge
  - Request: JSON with `badgeNumber`, `issueDate`.

- Response: Returns `201 Created` with the `EmployeeBadge` object if the badge is successfully assigned or `500 Internal Server Error`.
- Validation:
  - If the employee already has a badge, return `409 Conflict` with the message: `Employee already has an assigned badge!`.

3. GET - /api/employee/{employeeId}
- Response: Returns `200 OK` with the `Employee` object, including `EmployeeBadge` details, if found. Otherwise, `404 Not Found`.

4. GET - /api/employee
- Response: Returns a list of all employees in `200 OK`. The list should be sorted by:
  - Ascending order of `name`.
  - Descending order of `designation`.
- Custom Query in the repository to implement sorting.

5. GET - /api/employeebadge/issue/{issueDate}
- Response: Returns a list of `EmployeeBadge` objects based on the specified `issueDate`, in `200 OK`, or `404 Not Found` if no badge matches.

6. DELETE - /api/employeebadge/{badgeId}
- Response: Returns `200 OK` with the message `EmployeeBadge {badgeId} deleted successfully` or `404 Not Found` if the badge is not found.

Exception Handling:
- DuplicateEmailException: Thrown when an email already exists for another employee.
- BadgeAlreadyAssignedException: Thrown when trying to assign a badge to an employee who already has one.

3. Spring Boot Application with "Customer" and "CustomerCard" Entities:

Functional Requirements:
Create a Spring Boot application with two entities: "Customer" and "CustomerCard."
- Each Customer can have one CustomerCard, and each CustomerCard can be assigned to one Customer.
- Implement a bidirectional one-to-one mapping between these entities using Spring Data JPA.

Project Structure:
- Create folders named `controller`, `model`, `repository`, and `service` inside the `WORKSPACE/springapp/src/main/java/com/shop/springapp`.

Inside the `controller` folder:
- CustomerController:
  - Handles requests related to the `Customer` entity.
- CustomerCardController:
  - Handles requests related to the `CustomerCard` entity.

Inside the `model` folder:
- Customer Entity:
  - `customerId` - int (auto-generated primary key)
  - `name` - String
  - `email` - String
  - `phone` - String
  - `customerCard` - `CustomerCard` (@OneToOne, mappedBy = "customer", @JsonManagedReference)

- CustomerCard Entity:
  - `cardId` - int (auto-generated primary key)
  - `cardNumber` - String
  - `expirationDate` - String
  - `customer` - `Customer` (@OneToOne, @JoinColumn, @JsonBackReference)

Inside the `repository` folder:
- CustomerRepo:
  - Extend `JpaRepository<Customer, Integer>`.
- CustomerCardRepo:
  - Extend `JpaRepository<CustomerCard, Integer>`.

Inside the `service` folder:
- CustomerService and CustomerCardService:
  - Create interfaces with necessary methods like `createCustomer()`, `getCustomer()`, `assignCardToCustomer()`.
- CustomerServiceImpl and CustomerCardServiceImpl:
  - Implement the above methods, handling exceptions and validations.

API Endpoints:
1. POST - /api/customer
  - Request: JSON with `name`, `email`, `phone`.
  - Response: Returns `201 Created` with the `Customer` object or `500 Internal Server Error`.
  - Validation:
    - If a customer with the same email exists, return `409 Conflict` with the message: `Customer with email {email} already exists!`.

2. POST - /api/customer/{customerId}/card
  - Request: JSON with `cardNumber`, `expirationDate`.

- Response: Returns `201 Created` with the `CustomerCard` object if the card is successfully assigned or `500 Internal Server Error`.
- Validation:
  - If the customer already has a card, return `409 Conflict` with the message: `Customer already has an assigned card!`.

3. GET - /api/customer/{customerId}
- Response: Returns `200 OK` with the `Customer` object, including `CustomerCard` details, if found. Otherwise, `404 Not Found`.

4. GET - /api/customer
- Response: Returns a list of all customers in `200 OK`. The list should be sorted by:
  - Ascending order of `name`.
  - Descending order of `phone`.
- Custom Query in the repository to implement sorting.

5. GET - /api/customercard/status/{status}
- Response: Returns a list of `CustomerCard` objects based on the specified `status`, in `200 OK`, or `404 Not Found` if no cards match.

6. DELETE - /api/customercard/{cardId}
- Response: Returns `200 OK` with the message `CustomerCard {cardId} deleted successfully` or `404 Not Found` if the card is not found.

Exception Handling:
- DuplicateEmailException: Thrown when an email already exists for another customer.
- CardAlreadyAssignedException: Thrown when trying to assign a card to a customer who already has one.