1.Create a Spring Boot application with two entities: "Station" and "Bus Depot". A Station can have multiple Bus Depots, and a Bus Depot can belong to only one Station. Implement a one-to-many bidirectional mapping between these entities using Spring JPA and incorporate handling for DuplicateStationException.

Functional Requirements:

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

Inside the controller folder, create classes named "StationController" and "BusDepotController".

Inside the model folder:

- Create a class named "Station" with the following attributes:
    1. stationId - int (auto-generated primary key)
    2. stationName - String
    3. location - String
    4. busDepots - List<BusDepot> (OneToMany, mappedBy = "station", JsonManagedReference)
- Create another class named "BusDepot" with the following attributes:
    1. busDepotId - int (auto-generated primary key)
    2. depotName - String
    3. capacity - int
    4. station - Station (ManyToOne, JsonBackReference)

Implement getters, setters, and constructors for the Station and Bus Depot entities.

Inside the repository folder, create interfaces named "StationRepo" and "BusDepotRepo".

Inside the service folder, create interfaces named "StationService" and "BusDepotService".

Inside the exception folder, create a class named "DuplicateStationException".

Also, create classes StationServiceImpl and BusDepotServiceImpl that should implement the StationService and BusDepotService.

API Endpoints:

- **POST** - "/station" - Returns response status 201 with station object on successful creation. In case of a DuplicateStationException, specifically when the station name already exists, it returns a status of 500 with an appropriate error message as "Station with name {stationName} already exists".
- **POST** - "/busdepot/station/{stationId}" - Returns response status 201 with a bus depot object on successfully mapping the bus depot to the stationId or else 500.
- **GET** - "/station/{stationId}" - Returns response status 200 with station object, which includes details of bus depots on successful retrieval or else 404.
- **GET** - "/busdepot" - Returns response status 200 with List<BusDepot> object on successful retrieval or else 404.
- **PUT** - "/busdepot/{busDepotId}" - Returns response status 200 with updated bus depot object on successful update or else 404.
- **DELETE** - "/busdepot/{busDepotId}" - Returns response status 200 with String "Bus Depot deleted successfully" on successful deletion or else "Bus Depot not found with ID: " + busDepotId.

2. **Create a Spring Boot application with two entities: "Course" and "Enrollment". A Course can have multiple Enrollments, and an Enrollment can belong to only one Course. Implement a one-to-many bidirectional mapping between these entities using Spring JPA and incorporate handling for DuplicateCourseException.**

Functional Requirements:

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

Inside the controller folder, create classes named "CourseController" and "EnrollmentController".

Inside the model folder:

- Create a class named "Course" with the following attributes:
    1. courseId - int (auto-generated primary key)
    2. courseName - String
    3. duration - String
    4. enrollments - List<Enrollment> (OneToMany, mappedBy = "course", JsonManagedReference)
- Create another class named "Enrollment" with the following attributes:
    1. enrollmentId - int (auto-generated primary key)

2. studentName - String
3. enrollmentDate - LocalDate
4. course - Course (ManyToOne, JsonBackReference)

Implement getters, setters, and constructors for the Course and Enrollment entities.

Inside the repository folder, create interfaces named “CourseRepo” and "EnrollmentRepo".

Inside the service folder, create interfaces named "CourseService" and "EnrollmentService".

Inside the exception folder, create a class named "DuplicateCourseException".

Also, create classes CourseServiceImpl and EnrollmentServiceImpl that should implement the CourseService and EnrollmentService.

API Endpoints:

- **POST** - "/course" - Returns response status 201 with course object on successful creation. In case of a DuplicateCourseException, specifically when the course name already exists, it returns a status of 500 with an appropriate error message as "Course with name {courseName} already exists".
- **POST** - "/enrollment/course/{courseId}" - Returns response status 201 with an enrollment object on successfully mapping the enrollment to the courseId or else 500.
- **GET** - "/course/{courseId}" - Returns response status 200 with course object, which includes details of enrollments on successful retrieval or else 404.
- **GET** - "/enrollment" - Returns response status 200 with List<Enrollment> object on successful retrieval or else 404.
- **PUT** - "/enrollment/{enrollmentId}" - Returns response status 200 with updated enrollment object on successful update or else 404.
- **DELETE** - "/enrollment/{enrollmentId}" - Returns response status 200 with String "Enrollment deleted successfully" on successful deletion or else "Enrollment not found with ID: " + enrollmentId.
- **GET** - "/course" - Returns response status 200 with List<Course> object on successful retrieval of all courses or else 404.
- **GET** - "/enrollment/student/{studentName}" - Returns response status 200 with List<Enrollment> object on successful retrieval of enrollments for a specific student or else 404.

3. Create a Spring Boot application with two entities: "Person" and "Loan Account". A Person can have multiple Loan Accounts, and a Loan Account can belong to only one Person. Implement a one-to-many bidirectional mapping between these entities using Spring JPA and incorporate handling for DuplicatePersonException.

Functional Requirements:

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

Inside the controller folder, create classes named "PersonController" and "LoanAccountController".

Inside the model folder:

- Create a class named "Person" with the following attributes:
    1. personId - int (auto-generated primary key)
    2. name - String
    3. email - String
    4. loanAccounts - List<LoanAccount> (OneToMany, mappedBy = "person", JsonManagedReference)
- Create another class named "LoanAccount" with the following attributes:
    1. accountId - int (auto-generated primary key)
    2. accountNumber - String
    3. loanAmount - double
    4. person - Person (ManyToOne, JsonBackReference)

Implement getters, setters, and constructors for the Person and LoanAccount entities.

Inside the repository folder, create interfaces named "PersonRepo" and "LoanAccountRepo".

Inside the service folder, create interfaces named "PersonService" and "LoanAccountService".

Inside the exception folder, create a class named "DuplicatePersonException".

Also, create classes PersonServiceImpl and LoanAccountServiceImpl that should implement the PersonService and LoanAccountService.

API Endpoints:

- **POST** - "/person" - Returns response status 201 with person object on successful creation. In case of a DuplicatePersonException, specifically when the person's email already exists, it returns a status of 500 with an appropriate error message as "Person with email {email} already exists".
- **POST** - "/loanaccount/person/{personId}" - Returns response status 201 with a loan account object on successfully mapping the loan account to the personId or else 500.
- **GET** - "/person/{personId}" - Returns response status 200 with person object, which includes details of loan accounts on successful retrieval or else 404.
- **GET** - "/loanaccount" - Returns response status 200 with List<LoanAccount> object on successful retrieval or else 404.
- **PUT** - "/loanaccount/{accountId}" - Returns response status 200 with updated loan account object on successful update or else 404.
- **DELETE** - "/loanaccount/{accountId}" - Returns response status 200 with String "Loan Account deleted successfully" on successful deletion or else "Loan Account not found with ID: " + accountId.
- **GET** - "/person" - Returns response status 200 with List<Person> object on successful retrieval of all persons or else 404.
- **GET** - "/loanaccount/person/{personId}" - Returns response status 200 with List<LoanAccount> object on successful retrieval of loan accounts for a specific person or else 404.

4.Create a Spring Boot application with two entities: "Product" and "Order". A Product can have multiple Orders, and an Order can belong to only one Product. Implement a one-to-many bidirectional mapping between these entities using Spring JPA and incorporate handling for DuplicateProductException.

Functional Requirements:

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

Inside the controller folder, create classes named "ProductController" and "OrderController".

Inside the model folder:

- Create a class named "Product" with the following attributes:
    1. productId - int (auto-generated primary key)
    2. productName - String
    3. price - double
    4. orders - List<Order> (OneToMany, mappedBy = "product", JsonManagedReference)
- Create another class named "Order" with the following attributes:
    1. orderId - int (auto-generated primary key)
    2. quantity - int
    3. orderDate - LocalDate
    4. product - Product (ManyToOne, JsonBackReference)

    Implement getters, setters, and constructors for the Product and Order entities.

    Inside the repository folder, create interfaces named “ProductRepo” and "OrderRepo" with JPQL queries for the following operations:

- Find all products.
- Find orders by product name.
- Count total orders for a specific product.
- Delete orders by ID.
- Update order quantity.

    Inside the service folder, create interfaces named "ProductService" and "OrderService".

    Inside the exception folder, create a class named "DuplicateProductException".

    Also, create classes ProductServiceImpl and OrderServiceImpl that should implement the ProductService and OrderService.

    API Endpoints:

- **POST** - "/product" - Returns response status 201 with product object on successful creation. In case of a DuplicateProductException, specifically when the product name already exists, it returns a status of 500 with an appropriate error message as "Product with name {productName} already exists".
- **POST** - "/order/product/{productId}" - Returns response status 201 with an order object on successfully mapping the order to the productId or else 500.

- **GET** - "/product/{productId}" - Returns response status 200 with product object, which includes details of orders on successful retrieval or else 404.
- **GET** - "/order" - Returns response status 200 with List<Order> object on successful retrieval or else 404.
- **PUT** - "/order/{orderId}" - Returns response status 200 with updated order object on successful update or else 404.
- **DELETE** - "/order/{orderId}" - Returns response status 200 with String "Order deleted successfully" on successful deletion or else "Order not found with ID: " + orderId.
- **GET** - "/product" - Returns response status 200 with List<Product> object on successful retrieval of all products or else 404.
- **GET** - "/order/product/{productId}" - Returns response status 200 with List<Order> object on successful retrieval of orders for a specific product or else 404.
- **GET** - "/order/count/{productId}" - Returns response status 200 with the total number of orders for a specific product or else 404.

5. **Create a Spring Boot application with two entities: "Customer" and "Order". A Customer can have multiple Orders, and an Order can belong to only one Customer. Implement a one-to-many bidirectional mapping between these entities using Spring JPA and incorporate handling for DuplicateCustomerException.**

Functional Requirements:

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

Inside the controller folder, create classes named "CustomerController" and "OrderController".

Inside the model folder:

- Create a class named "Customer" with the following attributes:
    1. customerId - int (auto-generated primary key)
    2. customerName - String
    3. email - String
    4. orders - List<Order> (OneToMany, mappedBy = "customer", JsonManagedReference)
- Create another class named "Order" with the following attributes:

1. orderId - int (auto-generated primary key)
2. product - String
3. quantity - int
4. orderDate - LocalDate
5. customer - Customer (ManyToOne, JsonBackReference)

Implement getters, setters, and constructors for the Customer and Order entities.

Inside the repository folder, create interfaces named “CustomerRepo” and "OrderRepo" with JPQL queries for the following operations:

- Find all customers.
- Find orders by customer name.
- Count total orders for a specific customer.
- Delete orders by ID.
- Update order quantity.

Inside the service folder, create interfaces named "CustomerService" and "OrderService".

Inside the exception folder, create a class named "DuplicateCustomerException".

Also, create classes CustomerServiceImpl and OrderServiceImpl that should implement the CustomerService and OrderService.

API Endpoints:

- **POST** - "/customer" - Returns response status 201 with customer object on successful creation. In case of a DuplicateCustomerException, specifically when the customer email already exists, it returns a status of 500 with an appropriate error message as "Customer with email {email} already exists".
- **POST** - "/order/customer/{customerId}" - Returns response status 201 with an order object on successfully mapping the order to the customerId or else 500.
- **GET** - "/customer/{customerId}" - Returns response status 200 with customer object, which includes details of orders on successful retrieval or else 404.
- **GET** - "/order" - Returns response status 200 with List<Order> object on successful retrieval or else 404.
- **PUT** - "/order/{orderId}" - Returns response status 200 with updated order object on successful update or else 404.

- **DELETE** - "/order/{orderId}" - Returns response status 200 with String "Order deleted successfully" on successful deletion or else "Order not found with ID: " + orderId.
- **GET** - "/customer" - Returns response status 200 with List<Customer> object on successful retrieval of all customers or else 404.
- **GET** - "/order/customer/{customerId}" - Returns response status 200 with List<Order> object on successful retrieval of orders for a specific customer or else 404.
- **GET** - "/order/count/{customerId}" - Returns response status 200 with the total number of orders for a specific customer or else 404.