QUESTION 1:

Overview:

Create a Spring Boot application with two entities: "Portfolio" and "Investment". A portfolio can have multiple investments, and an investment belongs to only one portfolio. Implement a one-to-many bidirectional mapping between these entities using Spring JPA. Utilize JPQL for retrieving details and incorporate handling for DuplicatePortfolioException.

Functional Requirements:

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

Folder Structure:

WORKSPACE/springapp/

```
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── examly/
│   │   │           └── springapp/
│   │   │               ├── controller/
│   │   │               │   ├── PortfolioController.java
│   │   │               │   ├── InvestmentController.java
│   │   │               ├── model/
│   │   │               │   ├── Portfolio.java
│   │   │               │   ├── Investment.java
│   │   │               ├── repository/
│   │   │               │   ├── PortfolioRepo.java
```

```
│   │   │       │   ├── InvestmentRepo.java
│   │   │       ├── service/
│   │   │       │   ├── PortfolioService.java
│   │   │       │   ├── InvestmentService.java
│   │   │       │   ├── PortfolioServiceImpl.java
│   │   │       │   ├── InvestmentServiceImpl.java
│   │   │       ├── exception/
│   │   │       │   ├── DuplicatePortfolioException.java
│   │   │       └── SpringappApplication.java
│   ├── resources/
│   │   ├── application.properties
│   │   └── data.sql
└── pom.xml
```

Inside the Folders:

controller/

1. **PortfolioController**
2. **InvestmentController**

model/

1. **Portfolio**:
    - portfolioId - int (auto-generated primary key)
    - portfolioName - String
    - owner - String
    - creationDate - LocalDate
    - investments - List<Investment> (OneToMany, mappedBy = "portfolio", @JsonManagedReference)
2. **Investment**:
    - investmentId - int (auto-generated primary key)

- investmentName - String
- amount - double
- type - String
- portfolio - Portfolio (ManyToOne, @JsonBackReference)

repository/

1. **PortfolioRepo**
2. **InvestmentRepo**

service/

1. **PortfolioService**
2. **InvestmentService**
3. **PortfolioServiceImpl**
4. **InvestmentServiceImpl**

exception/

1. **DuplicatePortfolioException**

API Endpoints:

1. **POST - "/portfolio"**
    - Returns response status **201** with the portfolio object on successful creation.
    - In case of a DuplicatePortfolioException, returns a status of **409** with the message:
      "Portfolio with name {portfolioName} already exists."
2. **POST - "/investment/{portfolioId}/portfolio"**
    - Returns response status **201** with the investment object mapped to the portfolio.
    - If the portfolio ID is not found, returns **404** with an error message.
    - For other errors, returns **500 (INTERNAL SERVER ERROR)**.
3. **GET - "/portfolio/{portfolioId}"**

- Returns response status **200** with the portfolio object, including investment details.
- If not found, returns **404**.

4. **GET - "/investment"**
    - Returns response status **200** with a list of investment objects on successful retrieval.
    - If no investments are found, returns **404**.
5. **GET - "/portfolio/bydate"**
    - Returns response status **200** with a list of portfolios sorted by creation date in descending order.
    - If no portfolios are found, returns **404**.
    - Use the @Query annotation in the repository for this query.
6. **DELETE - "/investment/{investmentId}"**
    - Returns response status **200** with the message "Investment deleted successfully" on deletion.
    - If not found, returns **404** with the message "No investment found with the given ID."

QUESTION 2:

Overview:

Create a Spring Boot application with two entities: "Account" and "Transaction". An account can have multiple transactions, and a transaction can belong to only one account. Implement a one-to-many bidirectional mapping between these entities using Spring JPA. Utilize JPQL for retrieving details and incorporate handling for DuplicateAccountException.

Functional Requirements:

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

Folder Structure:

```
WORKSPACE/springapp/
 ├── src/
 │   ├── main/
 │   │   ├── java/
 │   │   │   └── com/
 │   │   │       └── examly/
 │   │   │           └── springapp/
 │   │   │               ├── controller/
 │   │   │               │   ├── AccountController.java
 │   │   │               │   ├── TransactionController.java
 │   │   │               ├── model/
 │   │   │               │   ├── Account.java
 │   │   │               │   ├── Transaction.java
 │   │   │               ├── repository/
 │   │   │               │   ├── AccountRepo.java
 │   │   │               │   ├── TransactionRepo.java
 │   │   │               ├── service/
 │   │   │               │   ├── AccountService.java
 │   │   │               │   ├── TransactionService.java
 │   │   │               │   ├── AccountServiceImpl.java
 │   │   │               │   ├── TransactionServiceImpl.java
 │   │   │               ├── exception/
 │   │   │               │   ├── DuplicateAccountException.java
 │   │   │               └── SpringappApplication.java
 │   ├── resources/
 │   │   ├── application.properties
 │   │   └── data.sql
```

└── pom.xml

Inside the Folders:

controller/

1. **AccountController**
2. **TransactionController**

model/

1. **Account**:
    - accountId - int (auto-generated primary key)
    - accountName - String
    - accountType - String
    - balance - double
    - transactions - List<Transaction> (OneToMany, mappedBy = "account", @JsonManagedReference)
2. **Transaction**:
    - transactionId - int (auto-generated primary key)
    - transactionType - String
    - amount - double
    - timestamp - LocalDateTime
    - account - Account (ManyToOne, @JsonBackReference)

repository/

1. **AccountRepo**
2. **TransactionRepo**

service/

1. **AccountService**
2. **TransactionService**
3. **AccountServiceImpl**
4. **TransactionServiceImpl**

exception/

1. **DuplicateAccountException**

API Endpoints:

1. **POST - "/account"**
    - Returns response status **201** with the account object on successful creation.
    - In case of a DuplicateAccountException, returns a status of **409** with the message:
      "Account with name {accountName} already exists."
2. **POST - "/transaction/{accountId}/account"**
    - Returns response status **201** with the transaction object mapped to the account.
    - If the account ID is not found, returns **404** with an error message.
    - For other errors, returns **500 (INTERNAL SERVER ERROR)**.
3. **GET - "/account/{accountId}"**
    - Returns response status **200** with the account object, including transaction details.
    - If not found, returns **404**.
4. **GET - "/transaction"**
    - Returns response status **200** with a list of transaction objects on successful retrieval.
    - If no transactions are found, returns **404**.
5. **GET - "/account/bytype"**
    - Returns response status **200** with a list of accounts sorted by account type in ascending order.
    - If no accounts are found, returns **404**.
    - Use the @Query annotation in the repository for this query.
6. **DELETE - "/transaction/{transactionId}"**

- Returns response status **200** with the message "Transaction deleted successfully" on deletion.
- If not found, returns **404** with the message "No transaction found with the given ID."

QUESTION 3:
Overview:

Create a Spring Boot application with two entities: **"Genre"** and **"Movie"**. A genre can have multiple movies, and a movie can belong to only one genre. Implement a one-to-many bidirectional mapping between these entities using Spring JPA. Utilize JPQL for retrieving details and incorporate handling for DuplicateGenreException.

Functional Requirements:

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

Folder Structure:

```
WORKSPACE/springapp/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── examly/
│   │   │           └── springapp/
│   │   │               ├── controller/
│   │   │               │   ├── GenreController.java
│   │   │               │   ├── MovieController.java
│   │   │               ├── model/
│   │   │               │   ├── Genre.java
│   │   │               │   ├── Movie.java
│   │   │               ├── repository/
```

```
│ │ │          │  ├── GenreRepo.java
│ │ │          │  ├── MovieRepo.java
│ │ │          ├── service/
│ │ │          │  ├── GenreService.java
│ │ │          │  ├── MovieService.java
│ │ │          │  ├── GenreServiceImpl.java
│ │ │          │  ├── MovieServiceImpl.java
│ │ │          ├── exception/
│ │ │          │  ├── DuplicateGenreException.java
│ │ │          └── SpringappApplication.java
│ ├── resources/
│ │  ├── application.properties
│ │  └── data.sql
└── pom.xml
```

Inside the Folders:

controller/

1. **GenreController**
2. **MovieController**

model/

1. **Genre**:
    - genreId - int (auto-generated primary key)
    - genreName - String
    - description - String
    - movies - List<Movie> (OneToMany, mappedBy = "genre", @JsonManagedReference)
2. **Movie**:
    - movieId - int (auto-generated primary key)

- title - String
- director - String
- boxOffice - double
- genre - Genre (ManyToOne, @JsonBackReference)

repository/

1. **GenreRepo**
2. **MovieRepo**

exception/

1. **DuplicateGenreException**

service/

1. **GenreService**
2. **MovieService**
3. **GenreServiceImpl**
4. **MovieServiceImpl**

API Endpoints:

1. **POST - "/genre"**
    - Returns response status **201** with the genre object on successful creation.
    - In case of a DuplicateGenreException, returns a status of **500** with the message:
      "Genre with name {genreName} already exists!"
2. **POST - "/movie/{genreId}/genre"**
    - Returns response status **201** with the movie object mapped to the genre.
    - If the genre ID is not found, returns **500** with an appropriate error message.
3. **GET - "/movie"**
    - Returns response status **200** with a list of movie objects on successful retrieval.

- If no movies are found, returns **404**.

4. **GET - "/genre/{genreId}"**
    - Returns response status **200** with the genre object, including movie details.
    - If not found, returns **404**.
5. **GET - "/movie/byRevenue/{revenue}"**
    - Returns a **200 OK** response with a list of movie objects where the box office revenue is greater than the specified amount.
    - If no movies match the condition, returns **404**.
    - Use the @Query annotation in the repository for this query.
6. **DELETE - "/movie/{movieId}"**
    - Returns response status **200** with the message "Movie {movieId} deleted successfully" on successful deletion.
    - If not found, returns **404** with the message "Movie not found with ID: {movieId}."
7. **Exceptional Handling** :
    - Implement an additional exception class InvalidMovieTitleException.
    - Trigger this exception if the title attribute of a movie is less than 3 characters long during creation. Return a **400 BAD REQUEST** response with the message:
      "Movie title must be at least 3 characters long."