**Exercise 7: Online Bookstore - Introduction to Data Transfer Objects (DTOs)**

**Business Scenario:**

Use DTOs to transfer data between the client and server for books and customers.

**Instructions:**

1. **Create DTOs:**
   * Define BookDTO and CustomerDTO classes.
2. **Mapping Entities to DTOs:**
   * Use a library like **MapStruct** or **ModelMapper** to map entities to DTOs and vice versa.
3. **Custom Serialization/Deserialization:**
   * Customize JSON serialization and deserialization using Jackson annotations.

**Answers: -**

**Exercise 7: Online Bookstore - Introduction to Data Transfer Objects (DTOs)**

**1. Create DTOs**

**Step-by-Step Guide:**

1. **Defining BookDTO and CustomerDTO Classes:**
   * **Purpose of DTOs:**
     + DTOs (Data Transfer Objects) are used to transfer data between the client and server, separating the internal data representation from the API layer. This helps in controlling the data exposed to the client.
   * **Creating BookDTO:**
     + Create the BookDTO class in the src/main/java/com/example/bookstoreapi/dto package.

package com.example.bookstoreapi.dto;

import lombok.Data;

@Data

public class BookDTO {

private int id;

private String title;

private String author;

private double price;

private String isbn;

}

* + **Creating CustomerDTO:**
    - Similarly, create the CustomerDTO class.

package com.example.bookstoreapi.dto;

import lombok.Data;

@Data

public class CustomerDTO {

private int id;

private String firstName;

private String lastName;

private String email;

}

* + **Explanation:**
    - **Lombok’s @Data:** The @Data annotation generates getters, setters, toString(), equals(), and hashCode() methods automatically, simplifying the DTOs.
    - **Exposed Fields:** Only the fields you want to expose to the client are included in the DTOs, leaving out sensitive information like passwords.

**2. Mapping Entities to DTOs**

1. **Use a Library like MapStruct or ModelMapper to Map Entities to DTOs and Vice Versa:**
   * **Using MapStruct (Recommended for simplicity and performance):**
     + **Add Dependency:**
       - In your pom.xml, add the following dependencies:

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>1.5.2.Final</version>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>1.5.2.Final</version>

<scope>provided</scope>

</dependency>

* + - **Create a Mapper Interface:**
      * Define an interface to map between entities and DTOs. This interface will be implemented automatically by MapStruct during the build process.

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.model.Customer;

import org.mapstruct.Mapper;

@Mapper(componentModel = "spring")

public interface BookstoreMapper {

// Mapping methods

BookDTO bookToBookDTO(Book book);

Book bookDTOToBook(BookDTO bookDTO);

CustomerDTO customerToCustomerDTO(Customer customer);

Customer customerDTOToCustomer(CustomerDTO customerDTO);

}

* + - **Using the Mapper:**
      * Inject the mapper into your controllers or services to convert between entities and DTOs.

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookstoreMapper bookstoreMapper;

@PostMapping

public ResponseEntity<BookDTO> addBook(@RequestBody BookDTO bookDTO) {

Book book = bookstoreMapper.bookDTOToBook(bookDTO);

// Save book entity to database or perform logic

BookDTO responseDTO = bookstoreMapper.bookToBookDTO(book);

return new ResponseEntity<>(responseDTO, HttpStatus.CREATED);

}

}

* + **Explanation:**
    - **MapStruct:** MapStruct generates the implementation of the mapping methods at compile time, ensuring high performance and type safety.

**3. Custom Serialization/Deserialization**

1. **Customize JSON Serialization and Deserialization Using Jackson Annotations:**
   * **Customizing BookDTO:**
     + You can customize how the fields of the DTO are serialized and deserialized using Jackson annotations.

import com.fasterxml.jackson.annotation.JsonInclude;

import com.fasterxml.jackson.annotation.JsonProperty;

@Data

public class BookDTO {

private int id;

@JsonProperty("book\_title")

private String title;

@JsonProperty("book\_author")

private String author;

@JsonInclude(JsonInclude.Include.NON\_NULL)

private Double price;

private String isbn;

}

* + **Explanation:**
    - **@JsonProperty("book\_title"):** Changes the name of the field when it is serialized to JSON.
    - **@JsonInclude(JsonInclude.Include.NON\_NULL):** Excludes the price field from the JSON output if it is null.
  + **Example JSON Output:**
    - With the annotations, a BookDTO object with a null price field would be serialized as:

json

Copy code

{

"id": 1,

"book\_title": "Java Programming",

"book\_author": "John Doe",

"isbn": "123-456-789"

}

By using DTOs, mapping libraries like MapStruct, and customizing serialization/deserialization with Jackson, we can ensure that your RESTful service is both secure and efficient in its data exchange between the client and server.