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

**1. Creating Data Transfer Objects (DTOs)**

DTOs are used to transfer data between the client and server, separating internal entity models from the data exposed to the client. Here, we will create BookDTO and CustomerDTO to represent data for books and customers.

**Define BookDTO**

package com.example.bookstoreapi.dto;

import lombok.Data;

@Data

public class BookDTO {

private Long id;

private String title;

private String author;

private double price;

private String isbn;

}

Define CustomerDTO:

package com.example.bookstoreapi.dto;

import lombok.Data;

@Data

public class CustomerDTO {

private Long id;

private String name;

private String email;

}

In the CustomerDTO, we're excluding the phone field, for example, to expose only relevant data to the client. We can customize these as needed.

**2. Mapping Entities to DTOs**

We can use libraries like **MapStruct** or **ModelMapper** to map entities to DTOs and vice versa. For simplicity, we'll use **MapStruct** in this example.

**Step 1: Add MapStruct Dependency**

In pom.xml, add the MapStruct dependency:

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct</artifactId>

<version>1.5.0.Final</version>

</dependency>

<dependency>

<groupId>org.mapstruct</groupId>

<artifactId>mapstruct-processor</artifactId>

<version>1.5.0.Final</version>

<scope>provided</scope>

</dependency>

**Step 2: Create a Mapper Interface**

MapStruct automatically generates the mapping code. You define an interface with the required mappings.

**BookMapper**

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.model.Book;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface BookMapper {

BookMapper INSTANCE = Mappers.getMapper(BookMapper.class);

BookDTO bookToBookDTO(Book book);

Book bookDTOToBook(BookDTO bookDTO);

}

CustomerMapper:

package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.CustomerDTO;

import com.example.bookstoreapi.model.Customer;

import org.mapstruct.Mapper;

import org.mapstruct.factory.Mappers;

@Mapper

public interface CustomerMapper {

CustomerMapper INSTANCE = Mappers.getMapper(CustomerMapper.class);

CustomerDTO customerToCustomerDTO(Customer customer);

Customer customerDTOToCustomer(CustomerDTO customerDTO);

}

* BookMapper and CustomerMapper will generate methods to convert between entity objects (Book, Customer) and DTOs (BookDTO, CustomerDTO).

**Step 3: Use Mappers in Controllers**

We can now use these mappers in our controllers to convert entities to DTOs and vice versa.

**Example in BookController**

import com.example.bookstoreapi.dto.BookDTO;

import com.example.bookstoreapi.mapper.BookMapper;

import com.example.bookstoreapi.model.Book;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.stream.Collectors;

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> bookList = new ArrayList<>();

@GetMapping

public List<BookDTO> getAllBooks() {

return bookList.stream()

.map(BookMapper.INSTANCE::bookToBookDTO)

.collect(Collectors.toList());

}

@PostMapping

public BookDTO createBook(@RequestBody BookDTO bookDTO) {

Book book = BookMapper.INSTANCE.bookDTOToBook(bookDTO);

bookList.add(book);

return BookMapper.INSTANCE.bookToBookDTO(book);

}

}

In this example:

* The GET method maps the list of Book entities to a list of BookDTO before sending the response.
* The POST method accepts a BookDTO, converts it to a Book entity, and adds it to the list, returning the created BookDTO.

**3. Custom Serialization/Deserialization with Jackson**

Jackson annotations allow us to customize how JSON data is serialized and deserialized. Some common annotations include @JsonIgnore, @JsonProperty, and @JsonFormat.

**Example: Custom Serialization with Jackson**

We can use Jackson annotations in the DTO classes to customize how the fields are handled.

**Example in CustomerDTO**

package com.example.bookstoreapi.dto;

import com.fasterxml.jackson.annotation.JsonIgnore;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.Data;

@Data

public class CustomerDTO {

private Long id;

@JsonProperty("full\_name") // Renames 'name' to 'full\_name' in the JSON output

private String name;

@JsonIgnore // Hides the email field from the JSON output

private String email;

}

* **@JsonProperty:** Renames the name field to full\_name in the serialized JSON.
* **@JsonIgnore:** Excludes the email field from the JSON response.

**Example: Date Formatting with @JsonFormat**

import com.fasterxml.jackson.annotation.JsonFormat;

import lombok.Data;

import java.time.LocalDate;

@Data

public class BookDTO {

private Long id;

private String title;

@JsonFormat(pattern = "yyyy-MM-dd")

private LocalDate publishedDate;

}