**Exercise 2: Online Bookstore - Creating Basic REST Controllers**

**Business Scenario:**

Implement RESTful endpoints to manage books.

**Instructions:**

1. **Create Book Controller:**
   * Define a **BookController** class with request mappings for /books.
2. **Handle HTTP Methods:**
   * Implement methods to handle **GET**, **POST**, **PUT**, and **DELETE** requests.
3. **Return JSON Responses:**
   * Ensure the controller returns JSON responses.

Define the Book entity with attributes like **id, title, author, price**, and **isbn**.

**Answers: -**

**Exercise 2: Online Bookstore - Creating Basic REST Controllers**

**1. Creating Book Controller**

**Step-by-Step Guide:**

1. **Defining the BookController class:**
   * In the src/main/java/com/example/bookstoreapi/controller package (create this package if it doesn’t exist), create a new Java class named BookController.
   * Annotate the class with @RestController to define it as a REST controller.
   * Use the @RequestMapping("/books") annotation to map the base URL for all endpoints in this controller.

package com.example.bookstoreapi.controller;

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

@RestController

@RequestMapping("/books")

public class BookController {

// Endpoint methods will be added here

}

**2. Handling HTTP Methods**

1. **Implementing methods to handle GET, POST, PUT, and DELETE requests:**
   * **GET Request**: Retrieve a list of books or a specific book by ID.
   * **POST Request**: Add a new book to the store.
   * **PUT Request**: Update an existing book's details.
   * **DELETE Request**: Remove a book from the store.

Here’s how you can implement these methods:

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

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

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

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

// GET: /books - Retrieve all books

@GetMapping

public List<Book> getAllBooks() {

return books;

}

// GET: /books/{id} - Retrieve a book by its ID

@GetMapping("/{id}")

public Book getBookById(@PathVariable int id) {

return books.stream()

.filter(book -> book.getId() == id)

.findFirst()

.orElse(null);

}

// POST: /books - Add a new book

@PostMapping

public Book addBook(@RequestBody Book book) {

books.add(book);

return book;

}

// PUT: /books/{id} - Update an existing book

@PutMapping("/{id}")

public Book updateBook(@PathVariable int id, @RequestBody Book updatedBook) {

for (int i = 0; i < books.size(); i++) {

Book book = books.get(i);

if (book.getId() == id) {

books.set(i, updatedBook);

return updatedBook;

}

}

return null;

}

// DELETE: /books/{id} - Remove a book

@DeleteMapping("/{id}")

public String deleteBook(@PathVariable int id) {

books.removeIf(book -> book.getId() == id);

return "Book with ID " + id + " has been deleted.";

}

}

**3. Returning JSON Responses**

1. **Ensuring the controller returns JSON responses:**
   * The @RestController annotation ensures that all methods return JSON responses by default.
2. **Defining the Book entity:**
   * Create a new class named Book in the src/main/java/com/example/bookstoreapi/model package.

package com.example.bookstoreapi.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Book {

private int id;

private String title;

private String author;

private double price;

private String isbn;

}

* + This class uses Lombok annotations:
    - @Data: Generates getters, setters, toString, equals, and hashCode methods.
    - @AllArgsConstructor: Generates a constructor with all fields.
    - @NoArgsConstructor: Generates a no-argument constructor.

By following these steps, we’ll have a basic RESTful service for managing books in your online bookstore.