Exercise 1: Online Bookstore - Setting Up RESTful Services

1. Setup Spring Boot Project:

- **Initialize Project:**
    - Use Spring Initializr to create a new Spring Boot project.
    - **Project Name:** BookstoreAPI
    - **Dependencies:**
        - **Spring Web:** For creating RESTful web services.
        - **Spring Boot DevTools:** For faster application development with hot-swapping and live reload.
        - **Lombok:** To reduce boilerplate code in model classes.
    - **Generate Project:** Download and unzip the project.
- **Explore the Project Structure:**
    - **src/main/java**: Contains your Java code.
    - **src/main/resources**: Holds configuration files like application.properties.
    - **src/test/java**: Used for writing and executing test cases.
- **Spring Boot 3 Features:**
    - **Java 17+ Support:** Full support for Java 17 and beyond.
    - **AOT Compilation:** Advanced optimizations for faster startup times.
    - **Native Image Support:** Simplified GraalVM native image builds.
    - **Jakarta EE 9/10:** Migrated to the Jakarta EE namespace.
    - **Enhanced Observability:** Micrometer improvements for better monitoring.

Exercise 2: Online Bookstore - Creating Basic REST Controllers

1. Create Book Controller:

- **BookController Class:**
    - Create a new package com.bookstoreapi.controller.
    - Define BookController class with the @RestController annotation.
    - Map the controller to the /books endpoint using @RequestMapping("/books").

```
package com.bookstoreapi.controller;

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

@RestController

@RequestMapping("/books")
```

```
public class BookController {
    // GET /books
    @GetMapping
    public List<Book> getAllBooks() {
        // Fetch all books from the database
        return new ArrayList<>();
    }
    // POST /books
    @PostMapping
    public Book createBook(@RequestBody Book book) {
        // Save the new book to the database
        return book;
    }
    // PUT /books/{id}
    @PutMapping("/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book book) {
        // Update the book in the database
        return book;
    }
    // DELETE /books/{id}
    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) {
        // Delete the book from the database
    }
}
```

- **Return JSON Responses:**
    - **Book Entity:** Create Book class in com.bookstoreapi.model.

```
package com.bookstoreapi.model;
import lombok.Data;
@Data
public class Book {
```

```
    private Long id;
    private String title;
    private String author;
    private double price;
    private String isbn;
}
```

Exercise 3: Online Bookstore - Handling Path Variables and Query Parameters

1. Path Variables:

- Fetch a book by its ID using @PathVariable.

```
@GetMapping("/{id}")
public Book getBookById(@PathVariable Long id) {
    // Fetch the book by ID from the database
    return new Book(); // Replace with actual database call
}
```

2. Query Parameters:

- Filter books based on title and author using @RequestParam.

```
@GetMapping("/search")
public List<Book> searchBooks(@RequestParam String title, @RequestParam String author) {
    // Search for books by title and author
    return new ArrayList<>(); // Replace with actual database call
}
```

Exercise 4: Online Bookstore - Processing Request Body and Form Data

1. Request Body:

- Create a new customer by accepting a JSON request body using @RequestBody.

```
@PostMapping("/customers")
public Customer createCustomer(@RequestBody Customer customer) {
    // Save the new customer to the database
    return customer;
}
```

2. Form Data:

- Process form data for customer registrations using @ModelAttribute.

```
@PostMapping("/customers/register")
public String registerCustomer(@ModelAttribute Customer customer) {
   // Process registration form data
   return "Registration successful";
}
```

Exercise 5: Online Bookstore - Customizing Response Status and Headers

1. Response Status:

- Customize HTTP status codes using @ResponseStatus.

```
@ResponseStatus(HttpStatus.CREATED)
@PostMapping("/books")
public Book createBook(@RequestBody Book book) {
   // Save the new book to the database
   return book;
}
```

2. Custom Headers:

- Add custom headers to the response using ResponseEntity.

```
@GetMapping("/{id}")
public ResponseEntity<Book> getBookById(@PathVariable Long id) {
   Book book = new Book(); // Fetch book from the database
   return ResponseEntity.ok()
         .header("Custom-Header", "CustomValue")
         .body(book);
}
```

Exercise 6: Online Bookstore - Exception Handling in REST Controllers

1. Global Exception Handler:

- Create a GlobalExceptionHandler class annotated with @ControllerAdvice.

```
package com.bookstoreapi.exception;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import org.springframework.web.bind.annotation.ResponseStatus;


@ControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler(ResourceNotFoundException.class)
   @ResponseStatus(HttpStatus.NOT_FOUND)
   public String handleResourceNotFound(ResourceNotFoundException ex) {
      return ex.getMessage();
   }
}
```

- **Custom Exception:**

```
package com.bookstoreapi.exception;
public class ResourceNotFoundException extends RuntimeException {
   public ResourceNotFoundException(String message) {
      super(message);
   }
}
```

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

1. Create DTOs:

- Define BookDTO and CustomerDTO classes in com.bookstoreapi.dto.

```
package com.bookstoreapi.dto;
import lombok.Data;
@Data
public class BookDTO {
   private Long id;
   private String title;
   private String author;
   private double price;
}
package com.bookstoreapi.dto;
```

```
import lombok.Data;
@Data
public class CustomerDTO {
   private Long id;
   private String name;
   private String email;
}
```

2. Mapping Entities to DTOs:

- Use **MapStruct** to map between entities and DTOs.

```
package com.bookstoreapi.mapper;
import com.bookstoreapi.dto.BookDTO;
import com.bookstoreapi.model.Book;
import org.mapstruct.Mapper;
@Mapper(componentModel = "spring")
public interface BookMapper {
   BookDTO toDTO(Book book);
   Book toEntity(BookDTO bookDTO);
}
```

3. Custom Serialization/Deserialization:

- Customize JSON serialization using Jackson annotations.

```
package com.bookstoreapi.dto;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.Data;
@Data
public class BookDTO {
   private Long id;
   @JsonProperty("book_title")
   private String title;
   private String author;
   @JsonProperty("book_price")
   private double price;}
```