Exercise 1: Setting Up RESTful Services

Business Scenario:

You are tasked with developing a RESTful service for an online bookstore that will manage books, authors, and customers.

Instructions:

mvn archetype:generate -DgroupId=com.example.bookstoreapi -DartifactId=BookstoreAPI -Dversion=1.0.0 -DinteractiveMode=false

Dependencies to add in pom.xml:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
  </dependency>
  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

BookstoreApiApplication.java

```
package com.example.bookstoreapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BookstoreApiApplication {
```

```
    public static void main(String[] args) {
        SpringApplication.run(BookstoreApiApplication.class, args);
    }
}
```

Exercise 2: Creating Basic REST Controllers

Business Scenario:

Implement RESTful endpoints to manage books in the online bookstore.

BookController.java

```
package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.Book;
import com.example.bookstoreapi.service.BookService;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    @GetMapping
    public List<Book> getAllBooks() {
        return bookService.getAllBooks();
```

```
    }

    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookService.addBook(book);
    }

    @PutMapping("/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book book) {
        return bookService.updateBook(id, book);
    }

    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookService.deleteBook(id);
    }
}
```

Book.java

```
package com.example.bookstoreapi.entity;
import lombok.Data;
import javax.persistence.*;
@Data
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    private String author;
    private Double price;
    private String isbn;
```

```
}
```

Exercise 3: Handling Path Variables and Query Parameters

Business Scenario:

Enhance the book management endpoints to handle dynamic URLs and query parameters.

BookController.java

```
@GetMapping("/{id}")
public Book getBookById(@PathVariable Long id) {
    return bookService.getBookById(id);
}
@GetMapping("/search")
public List<Book> searchBooks(@RequestParam(required = false) String title,
                              @RequestParam(required = false) String author) {
    return bookService.searchBooks(title, author);
}
```

Exercise 4: Processing Request Body and Form Data

Business Scenario:

Create endpoints to accept and process JSON request bodies and form data for customer registrations.

CustomerController.java

```
package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.entity.Customer;
import com.example.bookstoreapi.service.CustomerService;
import org.springframework.web.bind.annotation.*;

@RestController
```

```
@RequestMapping("/customers")
public class CustomerController {

    private final CustomerService customerService;

    public CustomerController(CustomerService customerService) {
        this.customerService = customerService;
    }

    @PostMapping
    public Customer createCustomer(@RequestBody Customer customer) {
        return customerService.createCustomer(customer);
    }

    @PostMapping("/register")
    public String registerCustomer(@RequestParam String name, @RequestParam String email) {
        return customerService.registerCustomer(name, email);
    }
}
```

Exercise 5: Customizing Response Status and Headers

Business Scenario:

Customize the HTTP response status and headers for the book management endpoints.

BookController.java

```
@ResponseStatus(HttpStatus.CREATED)
@PostMapping("/books")
public ResponseEntity<Book> createBook(@RequestBody Book book) {
    HttpHeaders headers = new HttpHeaders();
    headers.add("Custom-Header", "foo");
    return new ResponseEntity<>(book, headers, HttpStatus.CREATED);
```

```
}
HttpHeaders headers = new HttpHeaders();
headers.add("Custom-Header", "foo");
```

Exercise 6: Exception Handling in REST Controllers

Business Scenario:

Implement a global exception handling mechanism for the bookstore RESTful services.

GlobalExceptionHandler.java

```
package com.example.bookstoreapi.exception;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
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 ResponseEntity<String> handleNotFound(ResourceNotFoundException ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(IllegalArgumentException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    public ResponseEntity<String> handleBadRequest(IllegalArgumentException ex) {
        return new ResponseEntity<>(ex.getMessage(), HttpStatus.BAD_REQUEST);
```

```
    }
}
```

ResourceNotFoundException.java

```
package com.example.bookstoreapi.exception;


public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException(String message) {
        super(message);
    }
}
```

Exercise 7: Introduction to Data Transfer Objects (DTOs)

Business Scenario:

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

BookDTO.java

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

CustomerDTO.java

```
package com.example.bookstoreapi.dto;
import lombok.Data;
@Data
public class CustomerDTO {
    private Long id;
```

```
    private String name;
    private String email;
}
```

DTOMapper.java

```
package com.example.bookstoreapi.mapper;

import com.example.bookstoreapi.dto.BookDTO;
import com.example.bookstoreapi.entity.Book;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Component;

@Component
public class DTOMapper {

    private final ModelMapper modelMapper = new ModelMapper();

    public BookDTO convertToDTO(Book book) {
        return modelMapper.map(book, BookDTO.class);
    }

    public Book convertToEntity(BookDTO bookDTO) {
        return modelMapper.map(bookDTO, Book.class);
    }
}
```