Spring & Spring Boot Tasks - Library Management

Prepared by Anusha Kurlem

This document captures practical implementation of configuring Spring Framework and Spring Boot in a Library Management application.

# Exercise 1: Configuring a Basic Spring Application

1. Created a Maven project named LibraryManagement.

2. Added Spring Core dependency in pom.xml:

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.9</version>  
</dependency>

3. Defined beans in applicationContext.xml:

<bean id="bookRepository" class="com.library.repository.BookRepository"/>  
<bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
</bean>

4. Created BookRepository and BookService classes.

public class BookRepository {  
 public void displayBooks() {  
 System.out.println("Displaying books from the repository.");  
 }  
}

public class BookService {  
 private BookRepository bookRepository;  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
 public void listBooks() {  
 bookRepository.displayBooks();  
 }  
}

5. Ran the application using LibraryManagementApplication.java:

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
BookService bookService = context.getBean("bookService", BookService.class);  
bookService.listBooks();

Output:

Displaying books from the repository.

# Exercise 2: Implementing Dependency Injection

Used Spring's IoC to inject BookRepository into BookService using setter method.

Updated BookService with a setter:

public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
}

Verified wiring in applicationContext.xml.

Output:

Dependency injection successful. Output verified.

# Exercise 4: Creating and Configuring a Maven Project

Created a Maven project named LibraryManagement.

Added Spring dependencies in pom.xml:

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.9</version>  
</dependency>  
<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.9</version>  
</dependency>

Configured Maven compiler plugin:

<plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
</plugin>

# Exercise 5: Configuring the Spring IoC Container

Central configuration file created: applicationContext.xml

Defined BookService and BookRepository beans.

BookService has a setter method for BookRepository.

Output:

Spring IoC container loaded successfully. Beans initialized.

# Exercise 7: Implementing Constructor and Setter Injection

BookService class modified to support constructor injection:

public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
}

Setter method also retained for flexibility.

applicationContext.xml updated with both constructor and setter injection:

<bean id="bookService" class="com.library.service.BookService">  
 <constructor-arg ref="bookRepository"/>  
 <property name="bookRepository" ref="bookRepository"/>  
</bean>

Output:

Constructor and setter injections worked successfully.

# Exercise 9: Creating a Spring Boot Application

Created Spring Boot project using Spring Initializr named LibraryManagement.

Included dependencies: Spring Web, Spring Data JPA, H2 Database.

Configured database in application.properties:

spring.datasource.url=jdbc:h2:mem:librarydb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

Created entity and repository:

@Entity  
public class Book {  
 @Id  
 private int id;  
 private String title;  
}

public interface BookRepository extends JpaRepository<Book, Integer> {}

Developed BookController for RESTful CRUD operations:

@RestController  
@RequestMapping("/books")  
public class BookController {  
 @Autowired  
 private BookRepository repository;  
  
 @PostMapping  
 public Book save(@RequestBody Book book) {  
 return repository.save(book);  
 }  
  
 @GetMapping  
 public List<Book> getAll() {  
 return repository.findAll();  
 }  
}

Output:

Spring Boot application ran successfully. REST endpoints tested with Postman.