**Digital Nurture 4.0 Deep Skilling - Java FSE**

**Week -3 (Mandatory HandsOn)**

**Spring Core and Maven**

**Exercise 1: Configuring a Basic Spring Application**

**Scenario:**

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**Steps:**

**1. Set Up a Spring Project:**

* Create a Maven project named LibraryManagement.
* Add Spring Core dependencies in the pom.xml file.

**2. Configure the Application Context:**

* Create an XML configuration file named applicationContext.xml in the src/main/resources directory.
* Define beans for BookService and BookRepository in the XML file.

**3. Define Service and Repository Classes:**

* Create a package com.library.service and add a class BookService.
* Create a package com.library.repository and add a class BookRepository.

**4. Run the Application:**

* Create a main class to load the Spring context and test the configuration.

**Program**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

</dependencies>

</project>

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Repository Bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository" />

<!-- Service Bean -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**com.library.repository.BookRepository**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book '" + bookName + "' saved to the repository.");

}

}

**com.library.service.BookService**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}}

**App.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring context from XML

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

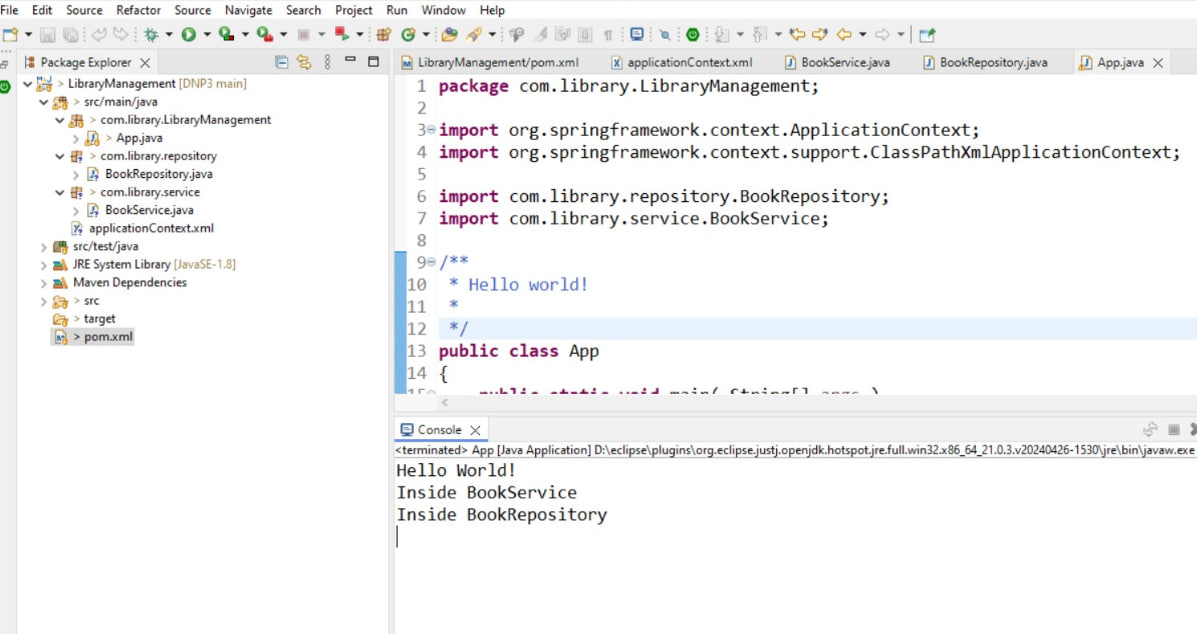
// Get BookService bean

BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring Framework Essentials");

}}

**Output**



**Excerise 2 Implementing Dependency Injection**

**Scenario:**

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**Steps:**

**1. Modify the XML Configuration**:

* Update applicationContext.xml to wire BookRepository into BookService.

**2. Update the BookService Class:**

* Ensure that BookService class has a setter method for BookRepository.

**3. Test the Configuration:**

* Run the LibraryManagementApplication main class to verify the dependency injection.

**Program**

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Define BookRepository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- Define BookService bean and inject BookRepository via setter -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

Updated **BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("BookService: Adding book - " + title);

bookRepository.saveBook(title);

}}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("BookRepository: Saving book - " + title);

}}

**App.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Get BookService bean

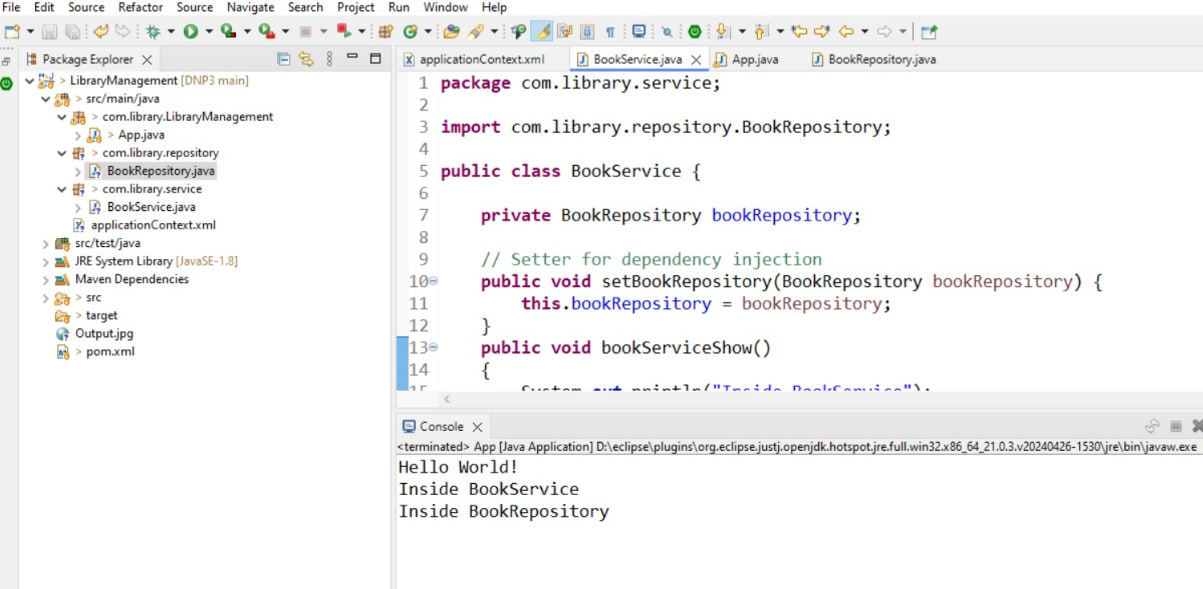
BookService bookService = (BookService) context.getBean("bookService");

// Use the service

bookService.addBook("Spring Dependency Injection Guide");

}}

**Output**



**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:**

You need to set up a new Maven project for the library management application and add Spring dependencies.

**Steps:**

**1. Create a New Maven Project:**

* Create a new Maven project named LibraryManagement.

**2. Add Spring Dependencies in pom.xml:**

* Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

**3. Configure Maven Plugins:**

* Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Program**

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>LibraryManagement</name>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<!-- Spring Core & Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Optional: Servlet API (provided scope for web apps) -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<!-- Step 3: Maven Compiler Plugin for Java 1.8 -->

<build>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

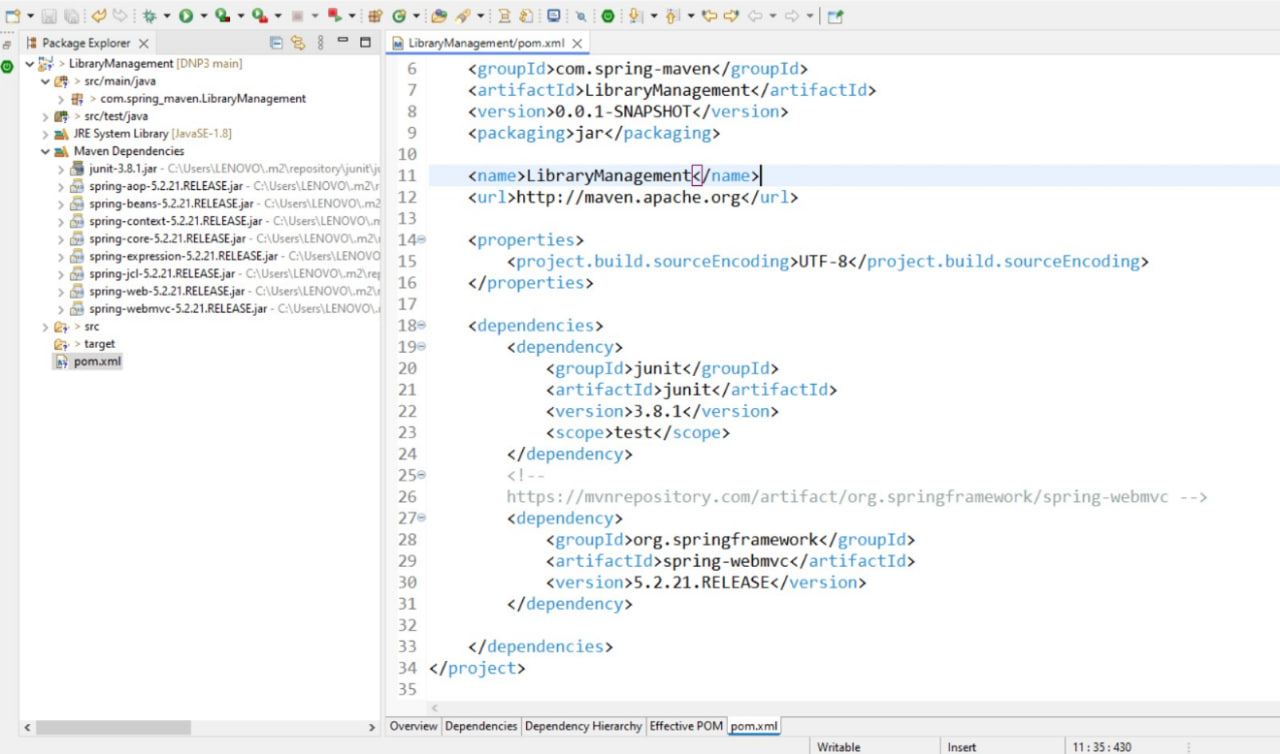
</plugin>

</plugins>

</build>

</project>

**Output**



**Additional HandsOn**

**Exercise 5: Configuring the Spring IoC Container**

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

**1. Create Spring Configuration File:**

* Create an XML configuration file named applicationContext.xml in the src/main/resources directory.
* Define beans for BookService and BookRepository in the XML file.

**2. Update the BookService Class:**

* Ensure that the BookService class has a setter method for BookRepository.

**3. Run the Application:**

* Create a main class to load the Spring context and test the configuration.

**Program**

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Define BookRepository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository" />

<!-- Define BookService bean and inject BookRepository -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("BookRepository: Saving book - " + title);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository; }

public void addBook(String title) {

System.out.println("BookService: Adding book - " + title);

bookRepository.saveBook(title);

}}

**App.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

// Get BookService bean

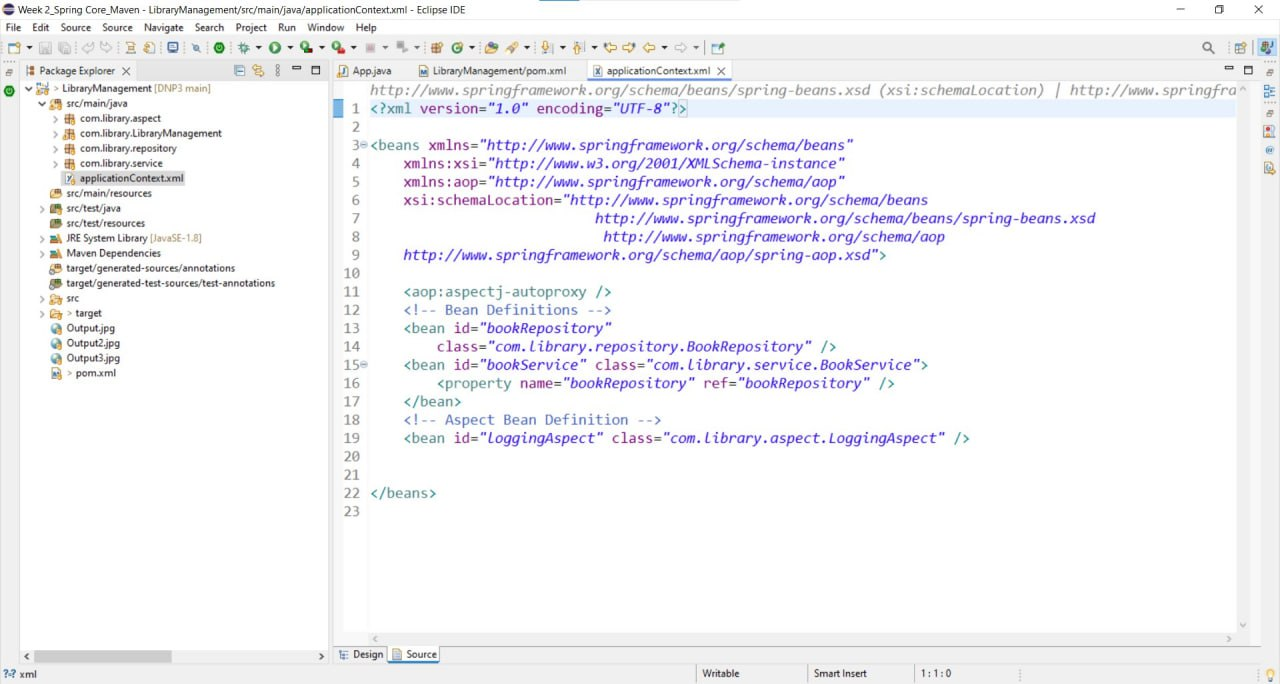
BookService bookService = context.getBean("bookService", BookService.class);

// Use BookService

bookService.addBook("Spring IoC Container Basics");

}}

**Output**



**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

**1. Configure Constructor Injection:**

* Update applicationContext.xml to configure constructor injection for BookService.

**2. Configure Setter Injection:**

* Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.

**3. Test the Injection:**

* Run the LibraryManagementApplication main class to verify both constructor and setter injection.

**Program**

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String serviceName;

private BookRepository bookRepository;

public BookService(String serviceName) {

this.serviceName = serviceName;}

// Setter for BookRepository (setter injection)

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository; }

public void addBook(String title) {

System.out.println(serviceName + ": Adding book - " + title);

bookRepository.saveBook(title);

}}

**BookRespository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("BookRepository: Saving book - " + title);

}}

**application.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- BookRepository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository" />

<!-- BookService bean with constructor and setter injection -->

<bean id="bookService" class="com.library.service.BookService">

<!-- Constructor injection -->

<constructor-arg value="LibraryService"/>

<!-- Setter injection -->

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**App.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

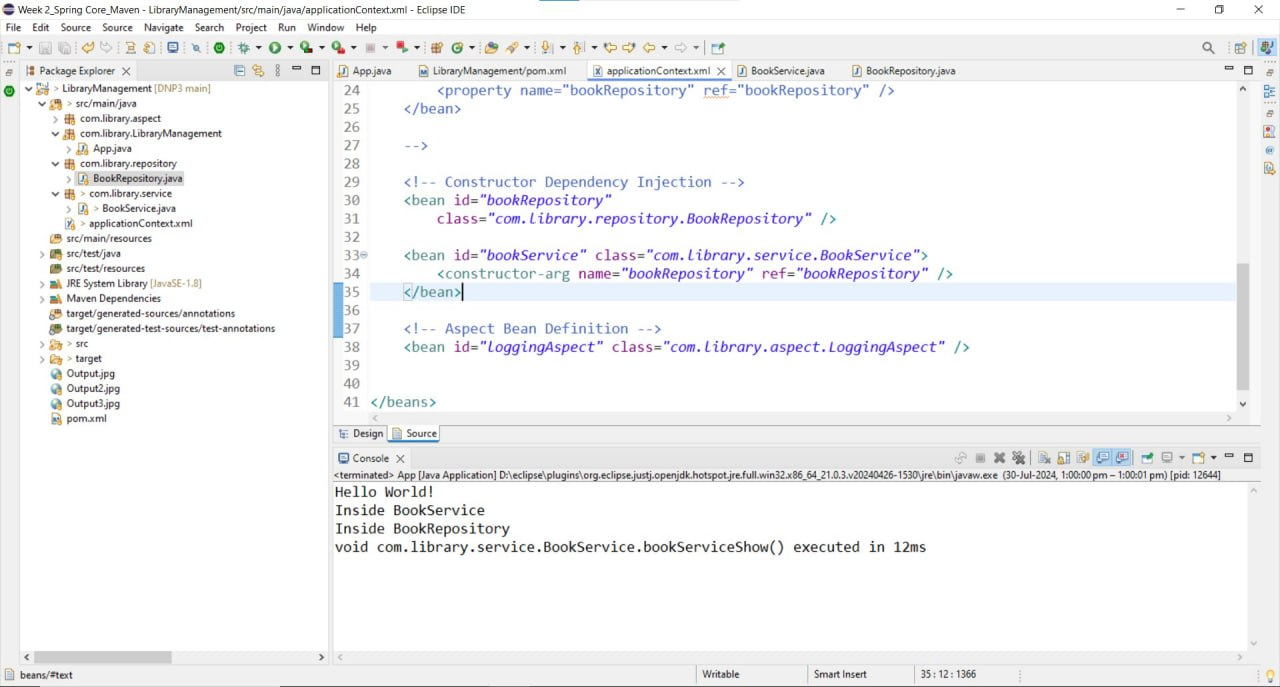
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Constructor and Setter Injection in Spring");

}

}

**Output**



**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

**1. Create a Spring Boot Project:**

* Use Spring Initializr to create a new Spring Boot project named LibraryManagement.

**2. Add Dependencies:**

* Include dependencies for Spring Web, Spring Data JPA, and H2 Database.

**3. Create Application Properties:**

* Configure database connection properties in application.properties.

**4. Define Entities and Repositories:**

* Create Book entity and BookRepository interface.

**5. Create a REST Controller:**

* Create a BookController class to handle CRUD operations.

**6. Run the Application:**

* Run the Spring Boot application and test the REST endpoints.

**Program**

**BookController.java**

package com.example.LibraryManagement.controller;

import com.example.LibraryManagement.entity.Book;

import com.example.LibraryManagement.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

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

import java.util.HashMap;

import java.util.List;

import java.util.Map;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll(); }

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable(value = "id") Long bookId) {

Book book = bookRepository.findById(bookId)

.orElseThrow();

return ResponseEntity.ok().body(book);}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book); }

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable(value = "id") Long bookId, @RequestBody Book bookDetails) {

Book book = bookRepository.findById(bookId)

.orElseThrow();

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setIsbn(bookDetails.getIsbn());

final Book updatedBook = bookRepository.save(book);

return ResponseEntity.ok(updatedBook);}

@DeleteMapping("/{id}")

public Map<String, Boolean> deleteBook(@PathVariable(value = "id") Long bookId) {

Book book = bookRepository.findById(bookId)

.orElseThrow();

bookRepository.delete(book);

Map<String, Boolean> response = new HashMap<>();

response.put("deleted", Boolean.TRUE);

return response;

}}

**Book.java**

package com.example.LibraryManagement.entity;

import javax.annotation.processing.Generated;

import jakarta.persistence.Entity;

@Entity

public class Book {

private Long id;

private String title;

private String author;

private String isbn;

// Getters and Setters

public Long getId() {

return id; }

public void setId(Long id) {

this.id = id;}

public String getTitle() {

return title;}

public void setTitle(String title) {

this.title = title; }

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;}

public String getIsbn() {

return isbn; }

public void setIsbn(String isbn) {

this.isbn = isbn;

}}

**Output**

