**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.*

**Step 1: Set Up a Maven Project**

**Project Name: LibraryManagement**

<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>

**Step 2: Configure applicationContext.xml**

**Path:** src/main/resources/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">

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

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

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

</bean>

</beans>

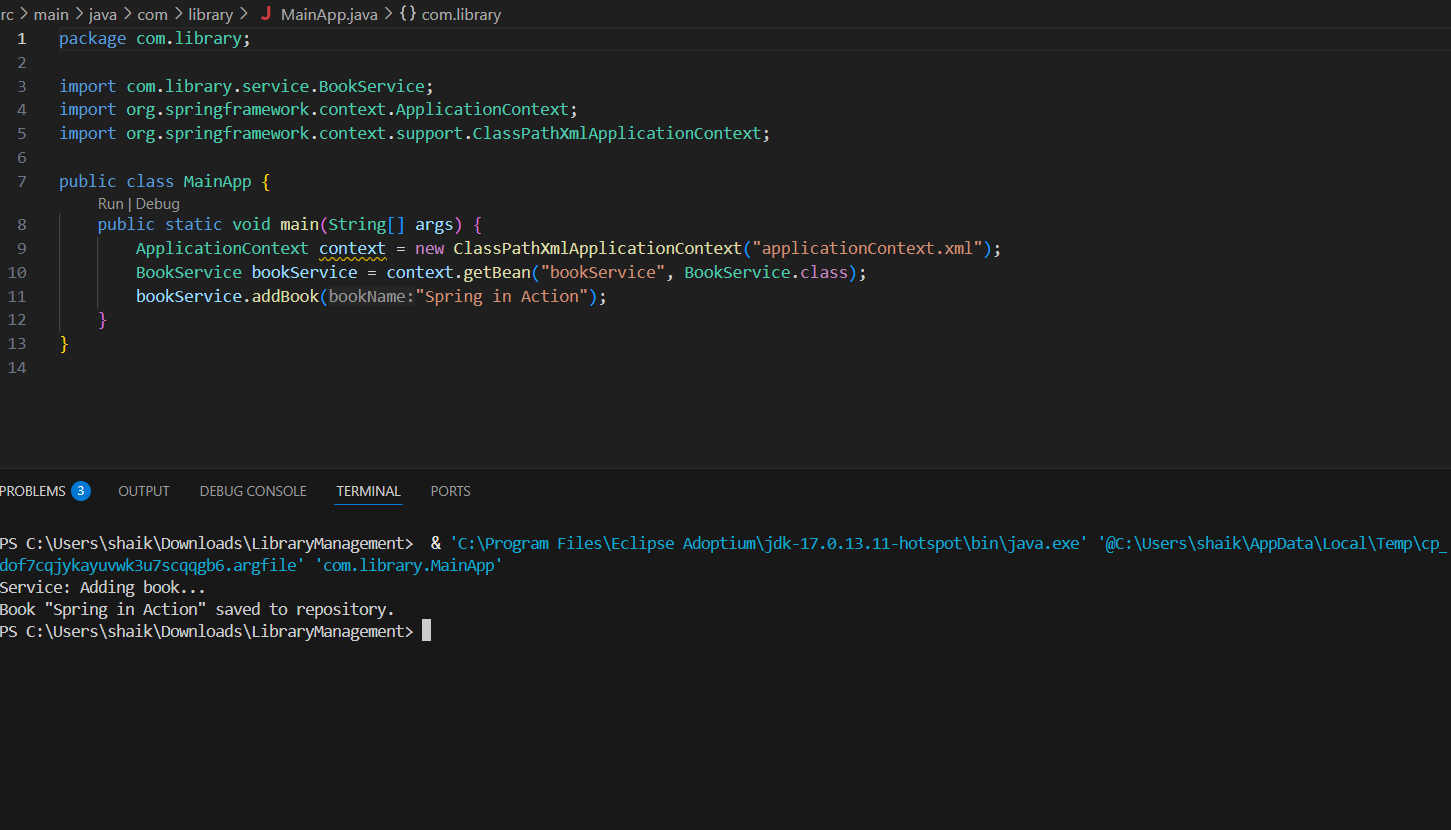
**Step 3: Define Java Classes**

**Package: com.library.repository**

BookRepository.java

**Step 4: Run the Application**

**Output Screenshot / Terminal Result:**

****

**Exercise 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 container and Dependency Injection.*

**Step 1: Modify the XML Configuration**

**File:** src/main/resources/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">

<!-- BookRepository bean -->

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

<!-- BookService bean with DI via setter -->

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

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

</bean>

</beans>

**Step 2: Update the BookService Class**

**File:** src/main/java/com/library/service/BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("Service: Adding book...");

bookRepository.saveBook(bookName);

}

}

**Step 3: Test the Configuration**

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) {

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

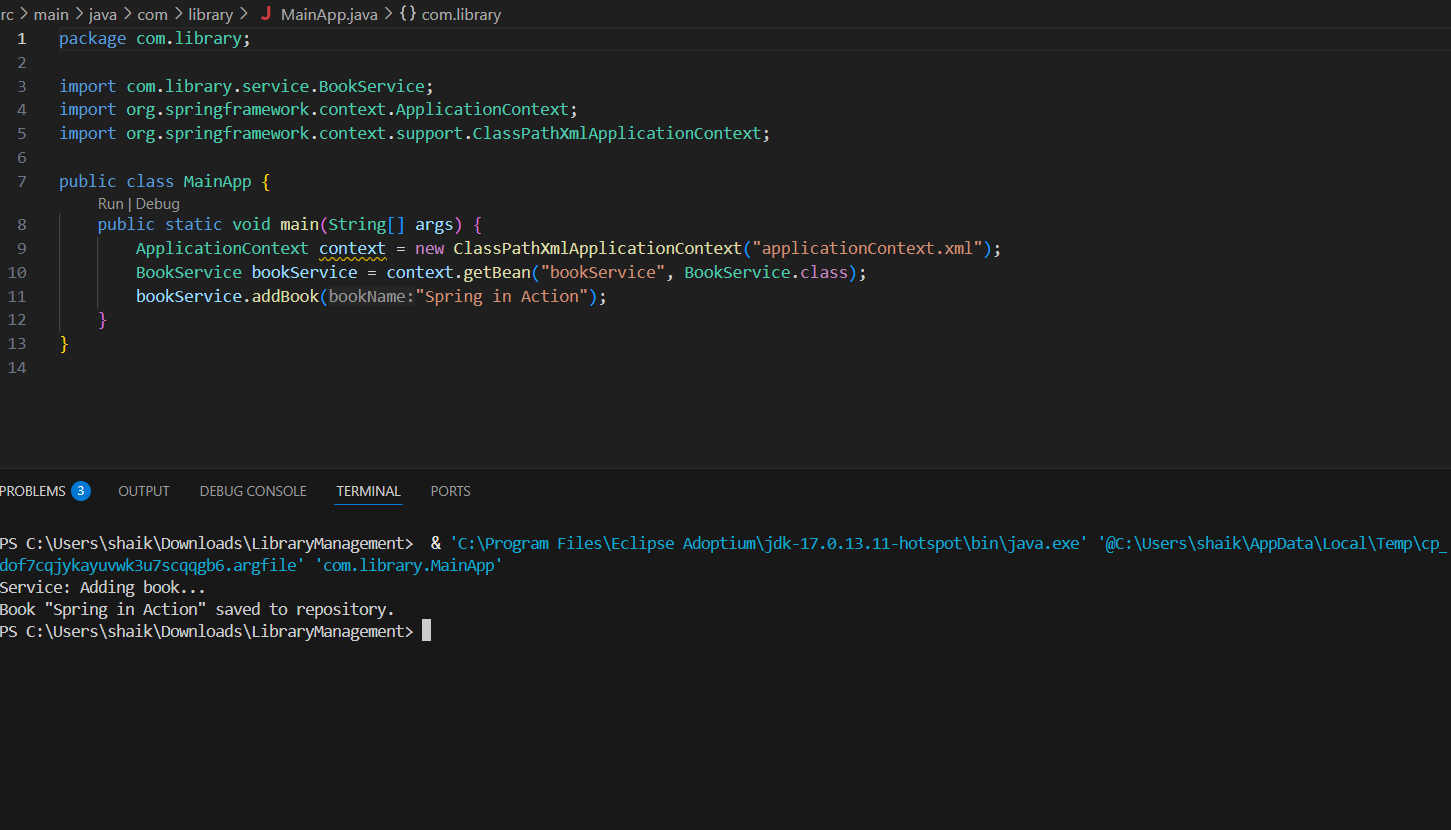
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring in Action");

}

}

**Output/Terminal Result**



**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 for core, AOP, and web functionality.*

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

Create a Maven project named LibraryManagement.

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

**📄 pom.xml Configuration**

<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 Container -->

<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>

</dependencies>

**Step 3: Configure Maven Compiler Plugin**

<build>

<plugins>

<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>

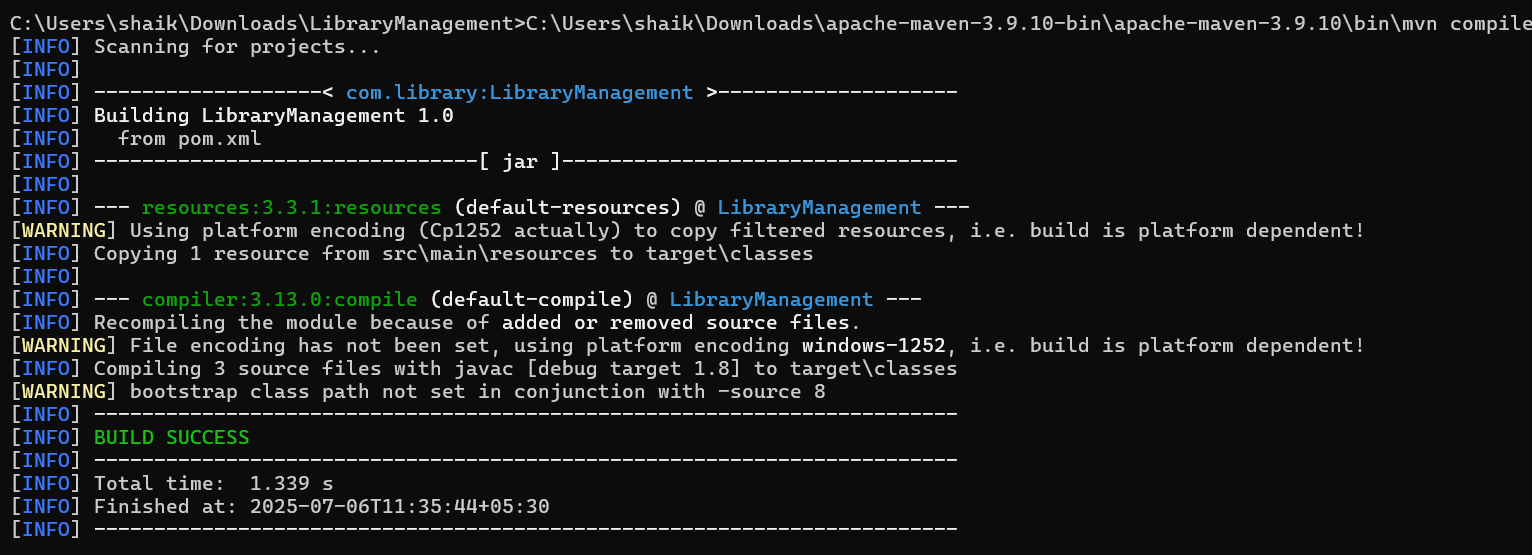
</plugin>

</plugins>

</build>

</project>

**Output Screenshot / Terminal Result:**



**Exercise 4 – Country Entity Implementation**

**Objective:**

To implement the Country entity, define the repository, service layer, and test the JPA functionality by retrieving a list of countries from the database using Spring Data JPA.

**Steps Performed:**

**1. Created Country Entity Class**

* Package: com.cognizant.orm\_learn
* Annotated the class with @Entity and mapped to the table country using @Table(name = "country").
* Defined two fields:
  + code (mapped to co\_code)
  + name (mapped to co\_name)
* Used @Id and @Column annotations to map to table columns.

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters and setters

}

**2. Created CountryRepository Interface**

* Extended JpaRepository<Country, String> to provide CRUD methods.
* No custom implementation required.

public interface CountryRepository extends JpaRepository<Country, String> {

}

**3.Created CountryService Class**

* Autowired CountryRepository.
* Created method getAllCountries() to fetch all records from the database.
* Annotated the method with @Transactional

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**4. Tested the Implementation**

* In the OrmLearnApplication class, called the getAllCountries() method and printed the result.
* Output was successfully printed to the console, confirming database interaction via JPA.

public class OrmLearnApplication {

@Autowired

private static CountryService countryService;

public static void main(String[] args) {

ConfigurableApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

testGetAllCountries();

context.close();

}

private static void testGetAllCountries() {

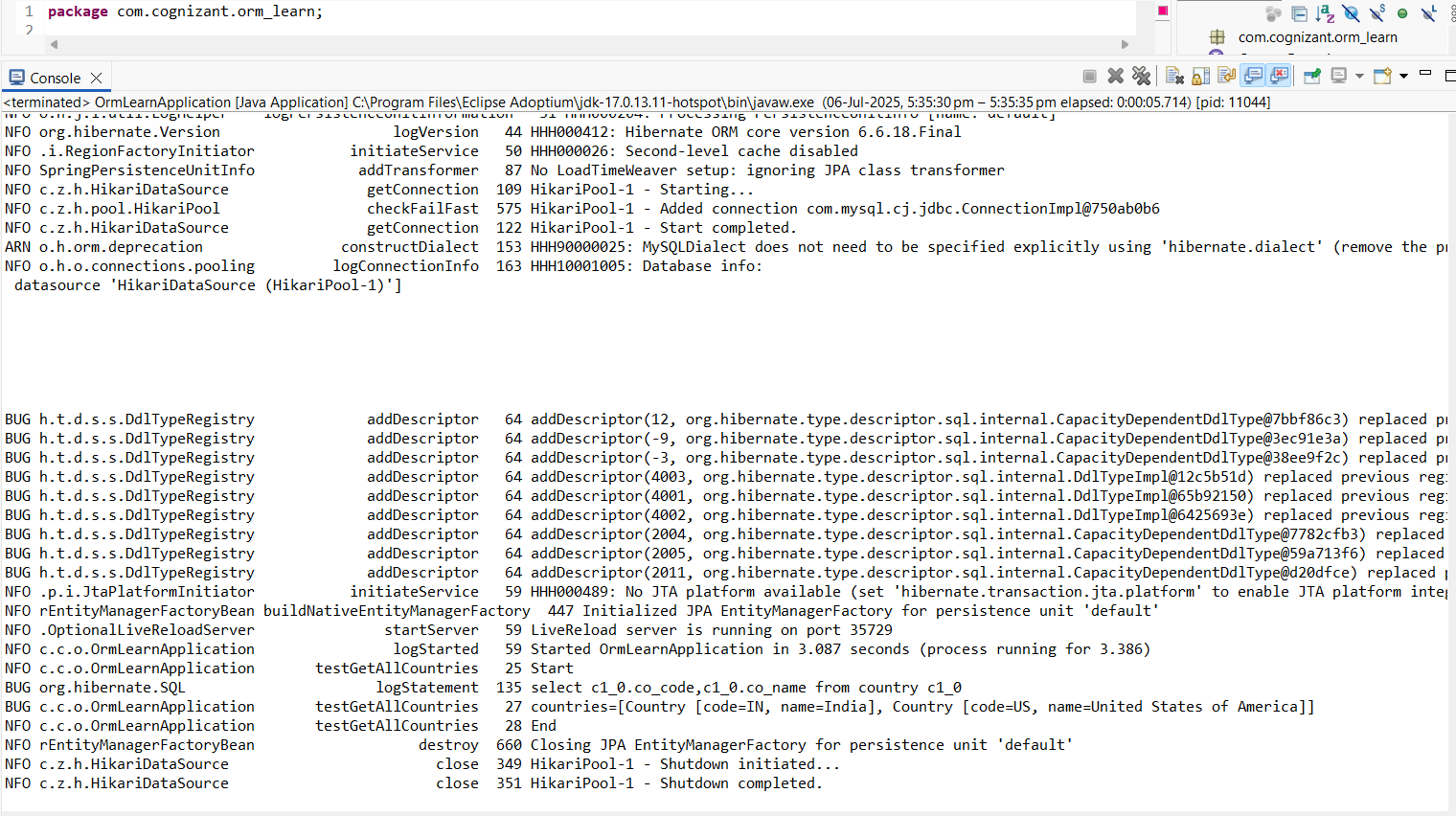
List<Country> countries = countryService.getAllCountries();

System.out.println("countries=" + countries);

}

}

**Console Output:**

****