**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.

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

**Project Name**: LibraryManagement

**pom.xml**

Add Spring Core dependencies inside the <dependencies> tag:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

</dependencies>

**Step 2: Create applicationContext.xml**

Create this file at:

src/main/resources/applicationContext.xml

**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 Repository Bean -->

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

<!-- Define Service Bean with dependency injection -->

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

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

</bean>

</beans>

**Step 3: Create the Repository Class**

Create the class:

src/main/java/com/library/repository/BookRepository.java

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book saved: " + bookName);

}

}

**Step 4: Create the Service Class**

Create the class:

src/main/java/com/library/service/BookService.java

**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 name) {

bookRepository.saveBook(name);

}

}

**Step 5: Create the Main Class to Run the Application**

Create the class:

src/main/java/com/library/LibraryMain.java

**LibraryMain.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMain {

public static void main(String[] args) {

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

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

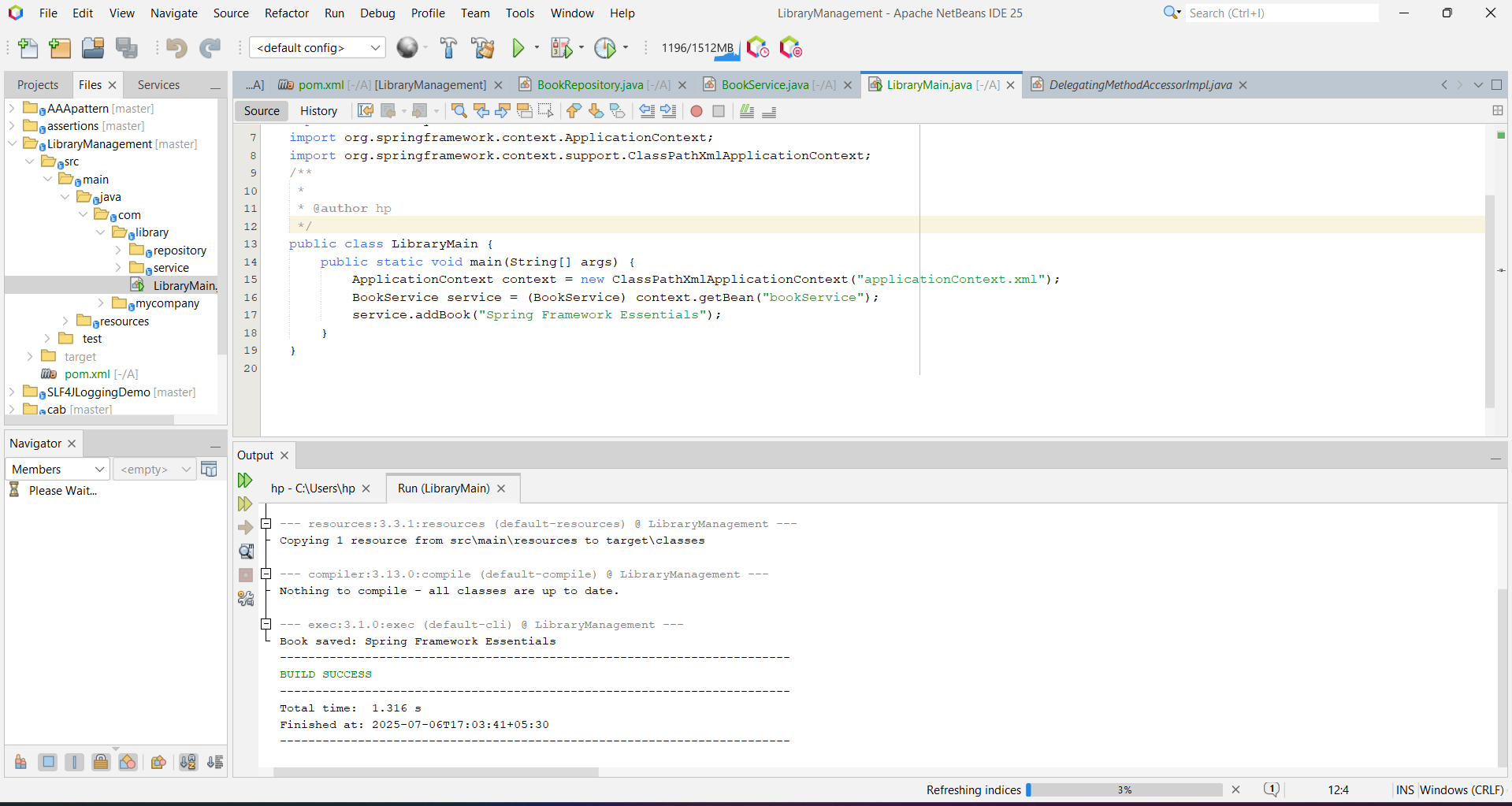
bookService.addBook("Spring Framework Essentials");

}

}

**OUTPUT:**

Book saved: Spring Framework Essentials



**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 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.**

**Step 1: Modify the XML Configuration**

**src/main/resources/applicationContext.xml**

**Ensure this XML file contains:**

<?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 Repository Bean -->

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

<!-- Define Service Bean with Dependency Injection -->

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

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

</bean>

</beans>

The <property> tag uses the setter method setBookRepository() to inject the dependency.

**Step 2: Update the BookService Class**

**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 Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String name) {

bookRepository.saveBook(name);

}

}

The BookService class now uses a setter to accept the BookRepository dependency from Spring.

**Step 3: Test the Configuration**

**src/main/java/com/library/LibraryMain.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMain {

public static void main(String[] args) {

// Load Spring XML Configuration

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

// Retrieve the bookService bean

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

// Use the service

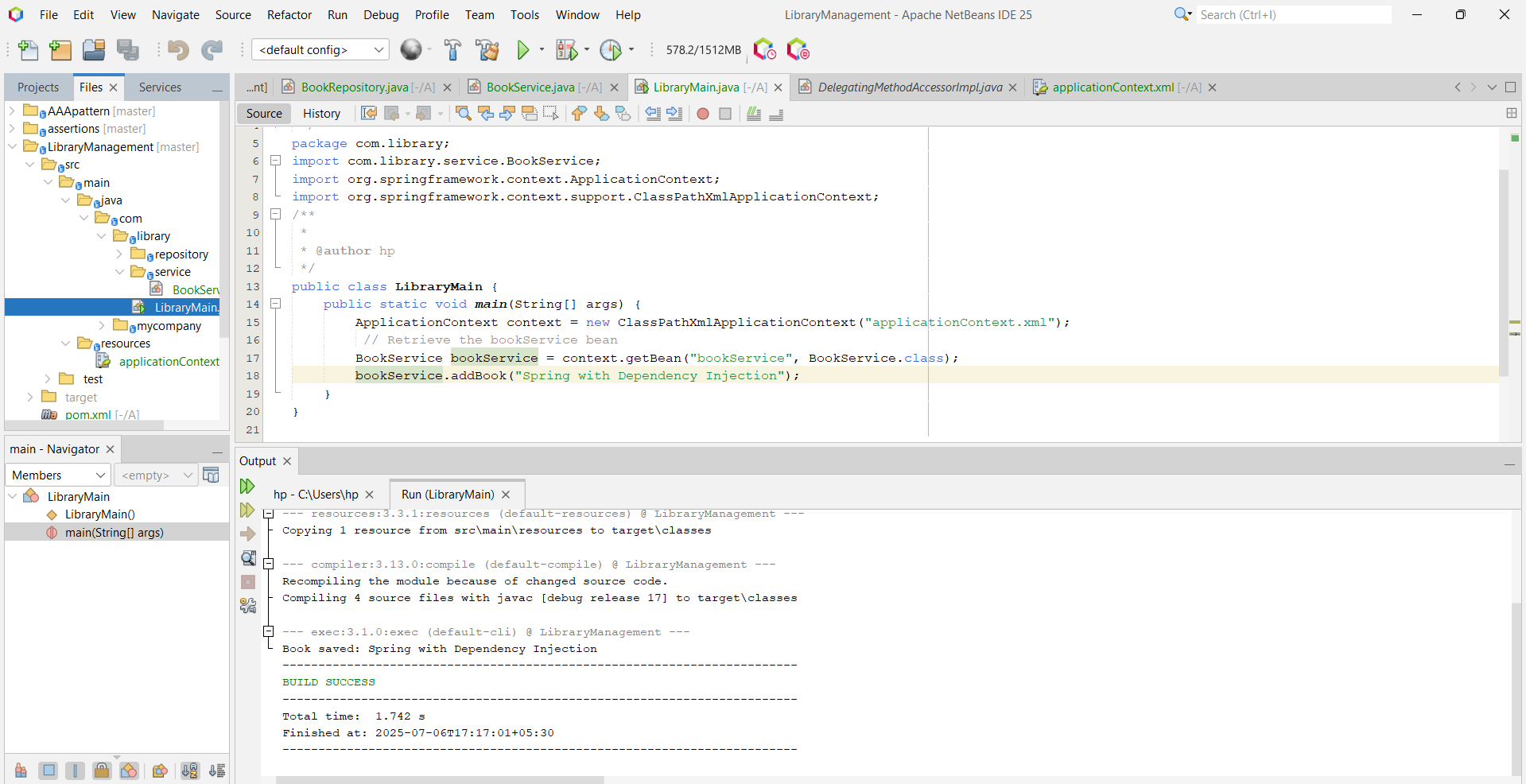
bookService.addBook("Spring with Dependency Injection");

}

}

**Expected Output**

**Book saved: Spring with Dependency Injection**

****

**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.**

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

1. Open NetBeans, IntelliJ, or your preferred IDE.
2. Create a new Maven project named:  
   LibraryManagement
3. Use default packaging type jar.

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

In the root pom.xml, paste the following dependencies inside <dependencies> tag:

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.22</version>

</dependency>

</dependencies>

**Step 3: Configure Maven Compiler Plugin for Java 1.8**

Add the following inside <build> tag in pom.xml:

<build>

<plugins>

<!-- 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>

</plugins>

</build>

**Final pom.xml Template**

<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.mycompany</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

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

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

</properties>

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.22</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- 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>

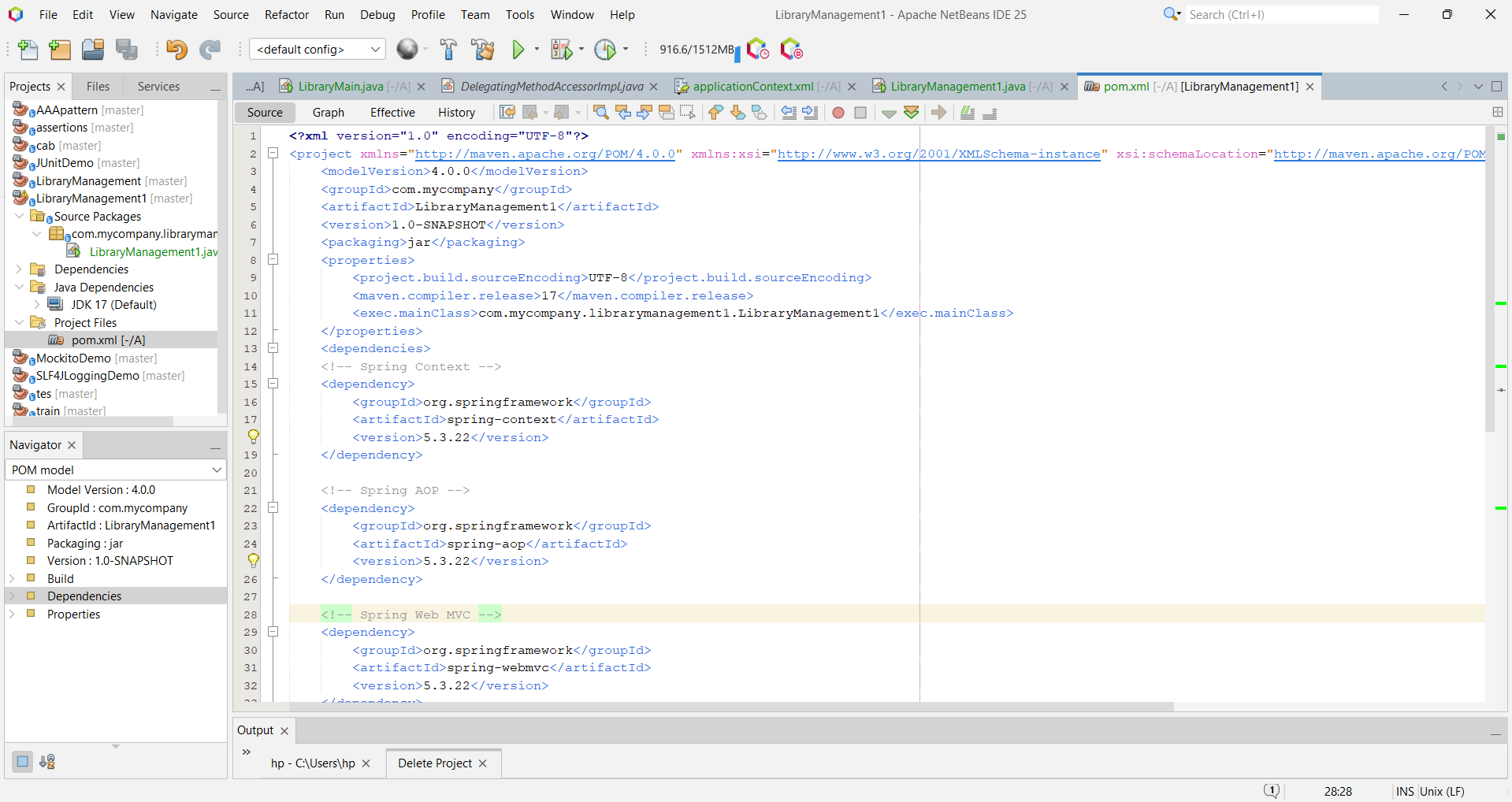
</configuration>

</plugin>

</plugins>

</build>

</project>

****

**Spring Data JPA - Quick Example**

**application.properties**

spring.application.name=Demo project for Spring Data JPA and Hibernate

# Logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# MySQL connection

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=password

# Hibernate config

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

**Country.java (Entity)**

package com.cognizant.orm\_learn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

private String code;

private String name;

// Getters and setters

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// toString

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**CountryRepository.java (Repository)**

package com.cognizant.orm\_learn.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import com.cognizant.orm\_learn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**DemoProjectForSpringDataJpaAndHibernateApplication.java (Main + Test)**

package com.cognizant.orm\_learn;

import java.util.List;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

@SpringBootApplication

public class DemoProjectForSpringDataJpaAndHibernateApplication implements CommandLineRunner {

@Autowired

private CountryRepository countryRepository;

public static void main(String[] args) {

SpringApplication.run(DemoProjectForSpringDataJpaAndHibernateApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

testGetAllCountries();

}

public void testGetAllCountries() {

System.out.println("Start");

List<Country> countries = countryRepository.findAll();

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

System.out.println("End");

}

}

**SQL Schema (optional to include in Word)**

CREATE DATABASE ormlearn;

USE ormlearn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

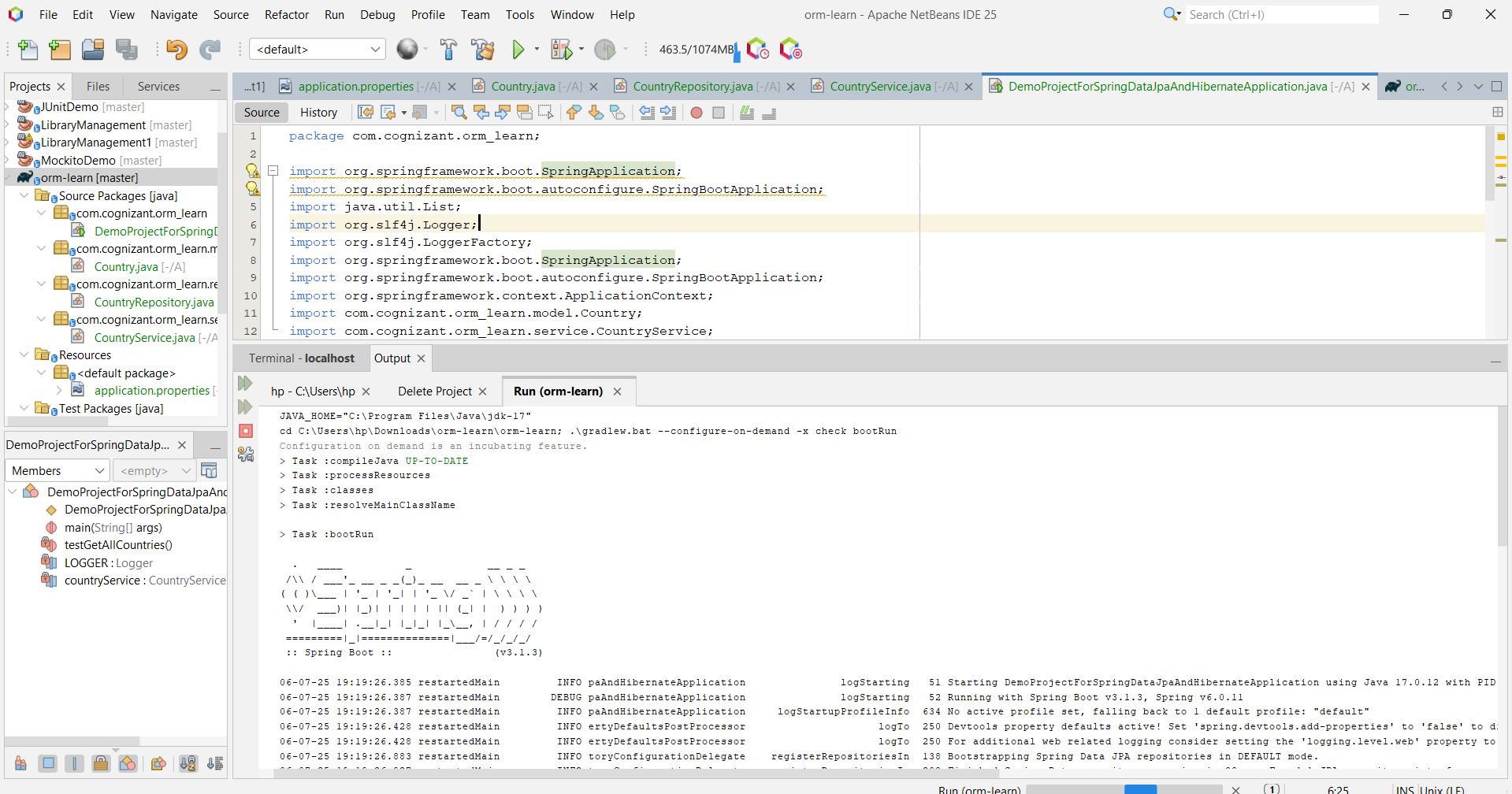
co\_name VARCHAR(100)

);

INSERT INTO country (co\_code, co\_name) VALUES ('IN', 'India');

INSERT INTO country (co\_code, co\_name) VALUES ('US', 'United States');

INSERT INTO country (co\_code, co\_name) VALUES ('JP', 'Japan');



**Difference between JPA, Hibernate and Spring Data JPA**

**Hibernate Code (Manual Session & Transaction Management)**

/\* Method to CREATE an employee in the database \*/

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Spring Data JPA Code (Clean & Abstracted)**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Hibernate Code Output (Manual)**

When you call:

Integer id = addEmployee(new Employee("John Doe", "HR"));

System.out.println("Employee added with ID: " + id);

📤 **Output:**

Hibernate: insert into employee (department, name) values (?, ?)

Employee added with ID: 1

**Spring Data JPA Output**

When you call the service method:

employeeService.addEmployee(new Employee("Jane Smith", "Finance"));

**Output in Console:**

Hibernate: insert into employee (department, name) values (?, ?)

If you want to **see the output on the browser/Postman** via a REST API:

POST /employees

{

"name": "Jane Smith",

"department": "Finance"

}

**Response:**

"Employee saved via Spring Data JPA"