# WEEK-3: SPRING CORE, MAVEN & SPRING DATA JPA

SPRING CORE & MAVEN

### ****Exercise 1 & 2: Configuring a Basic Spring Application with Dependency Injection****

### ****Scenario:****

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle backend operations.  
Initially, you configure a basic Spring application. Then, you implement **Dependency Injection (DI)** using Spring's IoC container.

### ****Steps:****

#### ****1. Create Maven Project****

* Project Name: LibraryManagement
* Add Spring Core dependency in pom.xml:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version></dependency>

**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>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies><build>

<plugins>

<!-- Maven Exec Plugin for running Java classes -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<!-- Replace with your actual main class name -->

<mainClass>com.library.LibraryManagementApplication</mainClass>

</configuration>

</plugin></plugins></build></project>

#### ****2. Configure the Application Context****

Create an XML file named applicationContext.xml inside src/main/resources.

Define beans for BookRepository and BookService.

**applicationContext.xml**

<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>

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

****Package:** com.library.repository  
**BookRepository.java****

package com.library.repository;

public class BookRepository {

public String getBookTitle() {

return "Spring in Action";

}

}

**Package:** com.library.service  
 **BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository; }

public void printBook() {

System.out.println("Book title: " + bookRepository.getBookTitle());

} }

#### ****Run the Application****

Package:com.library

**Main Class:** LibraryManagementApplication.java

package com.library;

import org.springframework.context.ApplicationContext;import org.springframework.context.support.ClassPathXmlApplicationContext;import com.library.service.BookService;

public class LibraryManagementApplication {

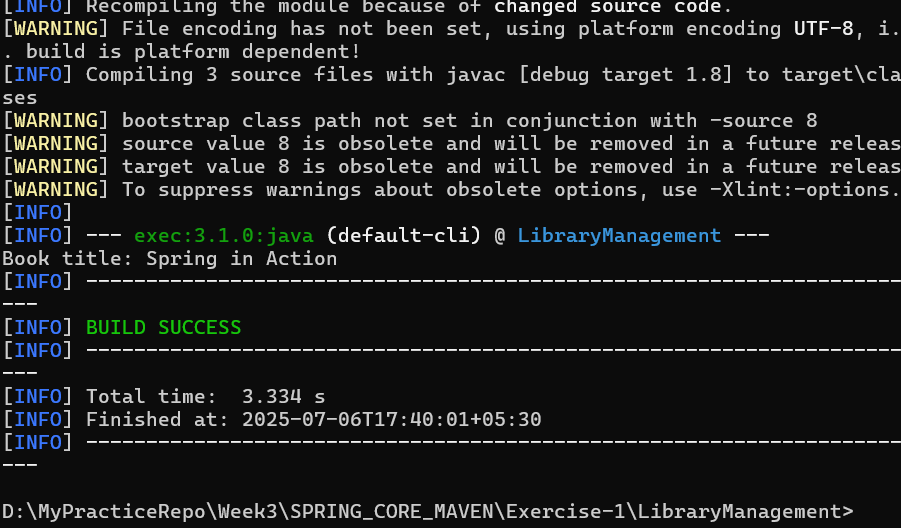
public static void main(String[] args) {

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

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

bookService.printBook(); } }

### ****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 required **Spring dependencies** and **Maven plugin configurations**.

### ****Steps:****

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

Create a Maven project named: LibraryManagement

#### ****Add Spring Dependencies in**** pom.xml

Include the following Spring dependencies:

spring-context (Core container & DI)

spring-aop (Aspect Oriented Programming support)

spring-webmvc (MVC framework)

**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>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<!-- Spring Core (Context) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

#### ****3. Configure Maven Compiler Plugin****

This ensures your code is compiled with **Java 1.8** compatibility.

(continued in pom.xml):

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

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

<version>3.8.0</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

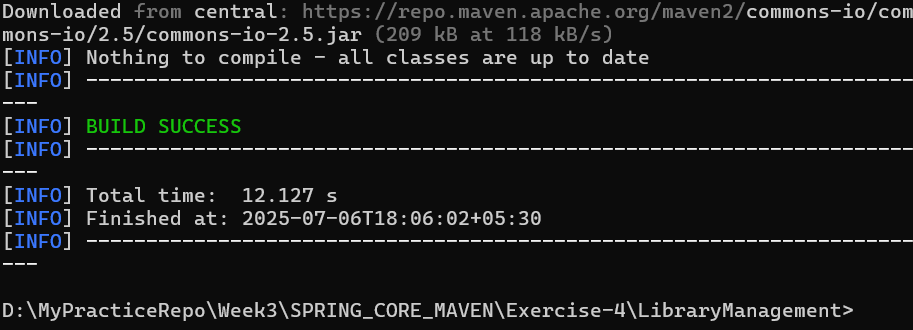
</configuration>

</plugin>

</plugins>

</build></project>

### ****Output:****



# Spring Data JPA with Spring Boot, Hibernate

### ****Hands-on 1: Spring Data JPA – Quick Example****

### ****Scenario:****

You are creating a demo Spring Boot project to explore how Spring Data JPA works with Hibernate and MySQL. The goal is to retrieve data from a table using a service and repository layer.

### ****Steps:****

1. **Create a Spring Boot project** from [https://start.spring.io/](https://start.spring.io/" \t "_new)

Group: com.cognizant

Artifact: orm-learn

Dependencies: Spring Boot DevTools, Spring Data JPA, MySQL Driver

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.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>orm-learn</name>

<description>Demo project for Spring Data JPA and Hibernate</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.18</version>

<relativePath/>

</parent>

<properties>

<java.version>11</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-logging</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

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

<version>3.8.0</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

<version>2.7.18</version>

<configuration>

<mainClass>com.cognizant.ormlearn.OrmLearnApplication</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

1. **Create MySQL schema:**

create schema ormlearn;

1. **Create table and insert data:**

**create table country(co\_code varchar(2) primary key, co\_name varchar(50));**

**insert into country values ('IN', 'India');**

**insert into country values ('US', 'United States of America');**

1. **Configure application.properties**

# 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

# DB 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=root

# Hibernate spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

1. **Code Implementation:**

**OrmLearnApplication.java**

package com.cognizant.ormlearn;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.model.Country;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static CountryService countryService;

public static void main(String[] args) {

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

countryService = context.getBean(CountryService.class);

testGetAllCountries(); } private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

**Country.java (Entity)**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

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; }

@Override

public String toString() {

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

**CountryRepository.java (Repository Interface)**

package com.cognizant.ormlearn.repository;

import com.cognizant.ormlearn.model.Country;

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> { }

**CountryService.java (Service Layer)**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll(); } }

**application.properties**

# Logging configuration

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

# Database configuration

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

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

spring.datasource.username=root

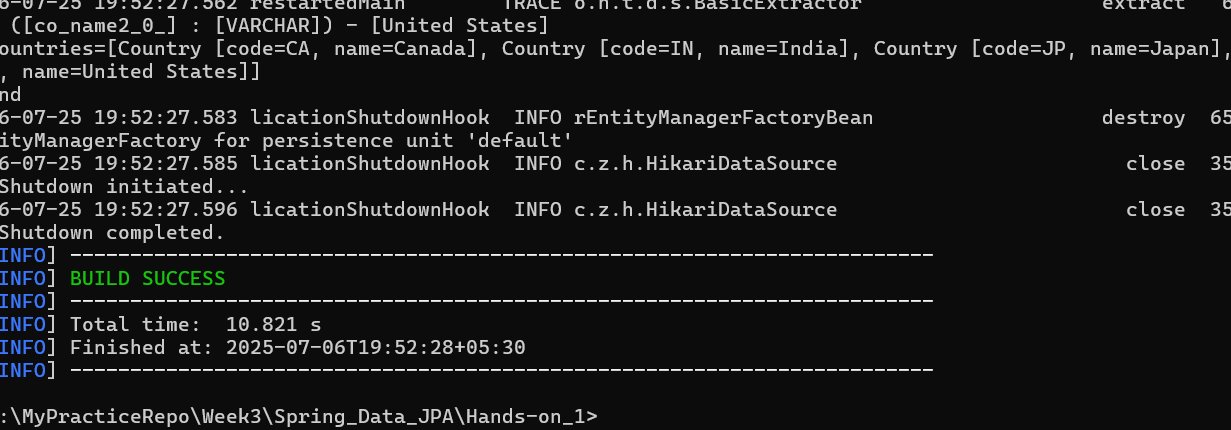
spring.datasource.password=root

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

### ****Output:****



## ****Hands-on 4: JPA vs Hibernate vs Spring Data JPA****

### ****Scenario:****

This hands-on explains the conceptual and implementation differences between **Java Persistence API (JPA)**, **Hibernate**, and **Spring Data JPA** with code examples.

### ****Differences:****

**1**. **Java Persistence API (JPA):**

JSR 338 Specification for persisting, reading, and managing data from Java objects

It is only a specification, not an implementation

Hibernate is a popular implementation of JPA

**2**. **Hibernate:**

An Object-Relational Mapping (ORM) tool

Provides an implementation of JPA

**3**. **Spring Data JPA:**

Not a JPA implementation itself, but builds on top of it

Provides abstraction and reduces boilerplate code

Handles most of the common JPA operations automatically

### ****Code Comparison:****

**Hibernate Example:**

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 Example:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}

@Servicepublic class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee); }

}

**Output:**  
Understanding of JPA vs Hibernate vs Spring Data JPA with code comparison and abstraction benefits using Spring Data JPA.  
No actual application output; focus is on conceptual clarity and simplified persistence logic.