**R TEJASHREE**

**Superset ID : 6419704**

**WEEK 3**

**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.

**Step1 : Set Up a Spring Project:**

* + Create a Maven project named **LibraryManagement**.

**Open Eclipse** → File > New > Maven Project

Select:

* + **Create a simple project (skip)**: (tick this)
  + Click **Next**

Fill in:

* + **Group Id**: com.library
  + **Artifact Id**: LibraryManagement
  + Click **Finish**

A new Maven project named LibraryManagement will be created.

**Step2 : Add Spring Core dependencies in the pom.xml file.**

Open the pom.xml file and add the following inside:

<dependencies> tag:

xml

Copy code

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

After adding, right-click the project > Maven > Update Project to download dependencies.

**Step3 : Create applicationContext.xml Configuration File**

1. **Right-click** on src/main/resources > New > Folder > name it resources (if not already there).
2. Right-click resources > New > File > name it 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>

**Step 4: Create Service and Repository Classes**

**➤ BookRepository**

1. Right-click src/main/java > New > Package > name it com.library.repository.
2. Inside it, create a class named BookRepository.

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

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

}

}

**➤ BookService**

1. Right-click src/main/java > New > Package > name it com.library.service.
2. Inside it, create a class named 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 title) {

bookRepository.saveBook(title);

}

}

**Step 5: Create Main Class to Load Spring Context**

1. Right-click src/main/java > New > Package > name it com.library.main
2. Create a class MainApp inside this package:

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

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

bookService.addBook("The Spring Handbook");

((ClassPathXmlApplicationContext) context).close();

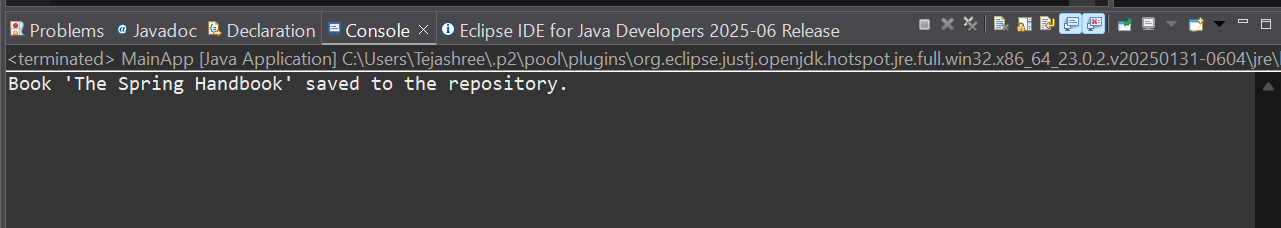
}

}

**Step 6: Run the Application**

* **Right-click MainApp.java** > Run As > Java Application

**OUTPUT 1 :**

****

**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.

**Step 1:Modifying pom.xml Configuration**

* Use the following full pom.xml to add Spring dependency:

<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

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

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryDIExample</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

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

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

</properties>

<dependencies>

<!-- Spring Core/Context Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

</project>

Save and **Maven > Update Project** after adding dependencies.

**Step 2: Updating the Repository Class**

* com.library.repository.BookRepository.java

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("Returning all books from repository.");

}

}

**Step 3: Create the Service Class with a Setter Method**

* 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 getBooks() {

System.out.println("Fetching books...");

bookRepository.displayBooks();

}

}

**Step 4: Configure Dependency Injection in applicationContext.xml**

* 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">

<!-- Define repository bean -->

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

<!-- Define service bean and inject repository using setter -->

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

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

</bean>

</beans>

* The property tag matches the setter setBookRepository() in BookService.

**Step 5: Create the Main Class to Test the Injection**

* com.library.main.LibraryManagementApplication.java

package com.library.main;

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) {

// Load Spring context from XML

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

// Get the bookService bean

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

// Use the bean

bookService.getBooks();

// Close the context

((ClassPathXmlApplicationContext) context).close();

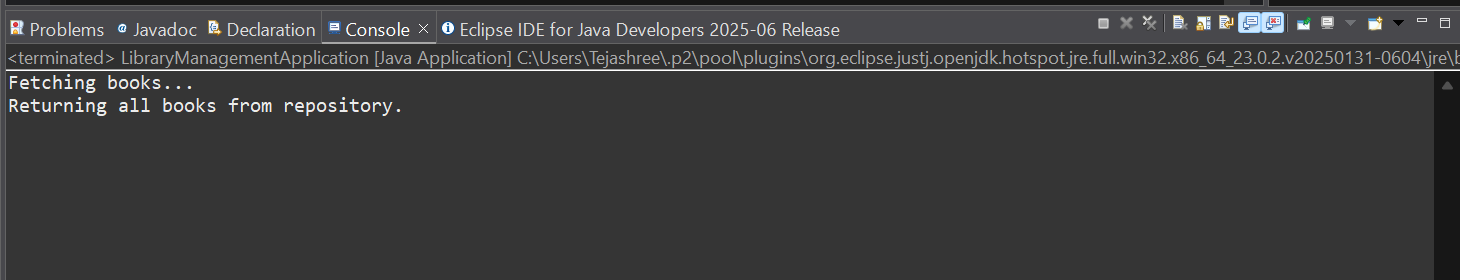
}

}

**Step 6: Run the Application**

1. Right-click LibraryManagementApplication.java
2. Choose: Run As → Java Application

**OUTPUT 2 :**

****

**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.

**Step 1: Update the pom.xml to Add Spring Dependencies**

1. Open the pom.xml file
2. Replace the content with the following full working config:

<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

https://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>

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

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

</properties>

<dependencies>

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

<!-- (Optional) Servlet API for WebMVC -->

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<!-- Maven Plugin Configuration -->

<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>

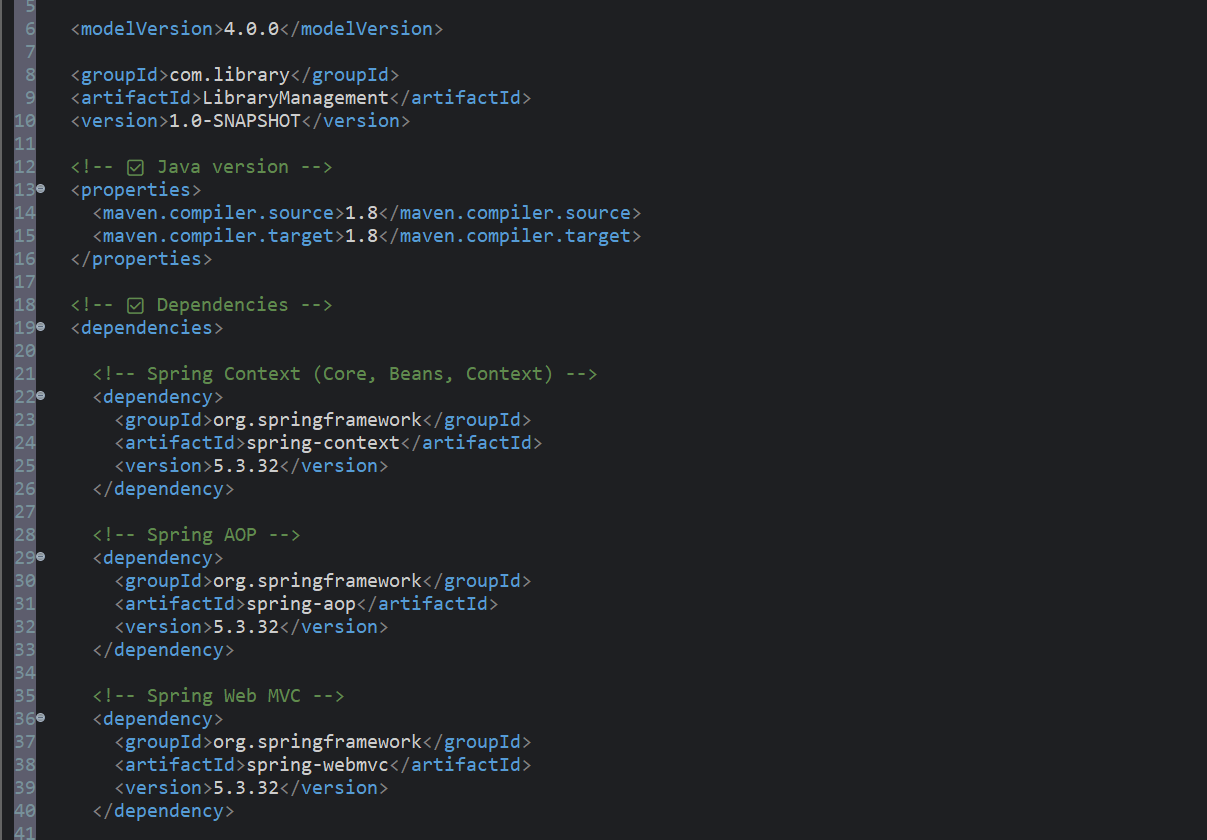
</build>

</project>

**Step 2: Update Maven**

1. Save the pom.xml
2. Right-click the project → Maven → Update Project…
3. Tick your project → Click **OK**
4. Wait for dependencies to download under **Maven Dependencies**

**OUTPUT 4 :**



**SPRING DATA JPA HANDSON**

**EX 1 : Difference between JPA, Hibernate and Spring Data JPA**

**🔹 Java Persistence API (JPA)**

* JPA is a **specification** (JSR 338) for object-relational mapping (ORM).
* It defines the **standard interface** for persisting, retrieving, and managing Java objects in a relational database.
* JPA **does not provide an implementation**—instead, vendors like Hibernate implement the JPA standard.
* It defines concepts like Entity, EntityManager, and annotations such as @Entity, @Id, etc.

**🔹 Hibernate**

* Hibernate is a **popular ORM framework** and a **JPA implementation**.
* It allows developers to map Java classes to database tables.
* Developers need to handle **session management** and **transactions** manually (unless integrated with Spring).
* It includes both JPA-compliant features and additional non-standard features (like caching and performance optimizations).

**Example using Hibernate:**

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**

* Spring Data JPA is **not a JPA implementation**.
* Spring Data JPA is a part of the Spring Framework that builds on top of JPA.
* It provides a **higher abstraction** on top of JPA, reducing boilerplate code.
* It **auto-generates repository methods**, manages transactions, and integrates tightly with Spring Boot.

**Example using Spring Data JPA:**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**EX2 : Hands on 1**

**Spring Data JPA - Quick Example**   
  
**Software Pre-requisites**

* MySQL Server 8.0
* MySQL Workbench 8
* Eclipse IDE for Enterprise Java Developers 2019-03 R
* Maven 3.6.2

**Create a Eclipse Project using Spring Initializr**

* Go to <https://start.spring.io/>
* Change Group as “com.cognizant”
* Change Artifact Id as “orm-learn”
* In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
* Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
* Click Generate and download the project as zip
* Extract the zip in root folder to Eclipse Workspace
* Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
* Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

**Step 1: Create Project from Spring Initializr**

1. Go to: <https://start.spring.io>
2. Fill the form:
   * **Group**: com.cognizant
   * **Artifact**: orm-learn
   * **Description**: Demo project for Spring Data JPA and Hibernate
3. Click **"Add Dependencies"**:
   * Spring Boot DevTools
   * Spring Data JPA
   * MySQL Driver
4. Click **Generate** → it downloads orm-learn.zip

**Step 2: Import into Eclipse**

1. Extract the orm-learn.zip to your workspace.
2. In Eclipse:  
   Go to File → Import → Maven → Existing Maven Projects
3. Browse to the extracted folder → Click **Finish**

Your project is now imported into Eclipse.

**Step 3: Create MySQL Schema**

1. Open terminal/command prompt and run:

mysql -u root -p

1. In MySQL client:

create schema ormlearn;

**Step 4: Configure application.properties**

File: src/main/resources/application.properties

Replace contents with:

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

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

# Database

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

Change username and password as per your MySQL setup if needed.

**Step 5: Add Logging in OrmLearnApplication.java**

File: src/main/java/com/cognizant/ormlearn/OrmLearnApplication.java

Update:

package com.cognizant.ormlearn;

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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

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();

LOGGER.info("Inside main");

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**Step 6: Create country Table in MySQL**

use ormlearn;

create table country (

code varchar(2) primary key,

name varchar(50)

);

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

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

**Step 7: Create the Model Class**

* src/main/java/com/cognizant/ormlearn/model/Country.java

package com.cognizant.ormlearn.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "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 + "]";

}

}

**Step 8: Create Repository Interface**

* src/main/java/com/cognizant/ormlearn/repository/CountryRepository.java

package com.cognizant.ormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**Step 9: Create Service Class**

* src/main/java/com/cognizant/ormlearn/service/CountryService.java

package com.cognizant.ormlearn.service;

import java.util.List;

import javax.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Step 10: Build & Run the Application**

1. Right-click the project > Run As > Maven Clean
2. Then: Run As → Maven Build  
   Set goals to:

clean package

(proxy options are not needed unless you're in a corporate network)

1. Finally: Right-click OrmLearnApplication.java → Run As → Java Application