**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**

* **Project Name:** LibraryManagement
* **Type:** Maven Project

<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 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

</project>

**2. Configure the Application Context**

* **File Name:** applicationContext.xml
* **Location:** 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

https://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>

**3. Define Service and Repository Classes**

**📄 BookRepository.java**

* **Package:** com.library.repository

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

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

}

}

**BookService.java**

* **Package:** com.library.service

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);

}

}

**4. Run the Application**

**📄 LibraryApp.java**

* **Package:** com.library

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

// Load Spring context

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

// Get BookService bean

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

// Test by adding a book

bookService.addBook("Effective Java");

}

}

**Conclusion**

In this exercise, we have successfully:

* Set up a Spring Maven project.
* Configured beans using XML-based configuration.
* Created service and repository classes.
* Verified the configuration by running a test main class.

Output Example:

Adding book: Effective Java

Book 'Effective Java' saved to the repository.

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 Inversion of Control (IoC) and Dependency Injection (DI).

**Steps**

**🔹 1. Modify the XML Configuration**

* **File:** applicationContext.xml
* **Location:** src/main/resources

**Updated 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

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

<!-- Define BookRepository bean -->

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

<!-- Define BookService bean with dependency injection -->

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

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

</bean>

</beans>

**Note:** The property tag inside the bookService bean definition ensures that Spring injects the BookRepository object into BookService.

**2. Update the BookService Class**

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

**Updated BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method 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);

}

}

**Note:** The setter method setBookRepository() enables Spring to inject the BookRepository dependency.

**3. Test the Configuration**

* **File:** LibraryApp.java
* **Package:** com.library

**LibraryApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

// Load Spring application context

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

// Get BookService bean from context

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

// Test the service

bookService.addBook("Clean Code");

}

}

**Conclusion**

In this exercise, we successfully:

* Configured Spring XML to wire BookRepository into BookService using setter injection.
* Updated BookService to include a setter for dependency injection.
* Verified the configuration by running the application.

Output Example

Adding book: Clean Code

Book 'Clean Code' saved to the repository.

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**

* **Project Name:** LibraryManagement

2.Add Spring Dependencies in pom.xml.

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 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>

<!-- Optional: Servlet API for web projects -->

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

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

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

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Explanation of dependencies:**

* **spring-context**: For core Spring features and dependency injection.
* **spring-aop**: For aspect-oriented programming support.
* **spring-webmvc**: For building web applications using Spring MVC.
* **javax.servlet-api**: Needed if you build a web application; scope is provided as the server provides it.

**3. Configure Maven Compiler Plugin**

* Already included in the <build> section of pom.xml above.
* **Configuration:**
  + source: 1.8
  + target: 1.8

This ensures that your project uses Java 1.8 for both source code and compiled classes.

**Conclusion**

In this exercise, we successfully:

* Created a new Maven project named **LibraryManagement**.
* Added required Spring dependencies (Spring Context, Spring AOP, Spring Web MVC).
* Configured the Maven Compiler Plugin to use Java version 1.8.

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**

Project Setup

**🔹 Create Eclipse Project using Spring Initializr**

1️⃣ Go to <https://start.spring.io/>

2️⃣ Configure:

* **Group:** com.cognizant
* **Artifact:** orm-learn
* **Description:** Demo project for Spring Data JPA and Hibernate
* **Dependencies:**
  + Spring Boot DevTools
  + Spring Data JPA
  + MySQL Driver

3️⃣ Click **Generate**, download the ZIP file.

4️⃣ Extract ZIP to your Eclipse workspace root folder.

5️⃣ Import into Eclipse:

File > Import > Maven > Existing Maven Projects > Browse extracted folder > Finish

Create Schema in MySQL

mysql -u root -p

mysql> create schema ormlearn;

**Configuration**

**🔹 application.properties**

**Location:** src/main/resources/application.propertie

# Spring and application logs

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs

logging.level.org.hibernate.SQL=trace

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

# Console log pattern

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 configuration

spring.jpa.hibernate.ddl-auto=validate

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

**Build the Project**

Execute this Maven command:

mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

**Main Class Setup**

**🔹 Logging**

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

Main Method

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

}

**Project Structure Walkthrough**

**1️⃣ src/main/java**

* Contains application source code.

**2️⃣ src/main/resources**

* Contains configuration files like application.properties.

**3️⃣ src/test/java**

* Contains unit and integration test code.

**4️⃣ OrmLearnApplication.java**

* Entry point of Spring Boot application. Contains main() method.

**5️⃣ @SpringBootApplication**

* Combination of:
  + @Configuration: Marks class as source of bean definitions.
  + @EnableAutoConfiguration: Enables Spring Boot auto-configuration.
  + @ComponentScan: Scans for components in package.

**6️⃣ pom.xml**

* Declares project dependencies, plugins, and build configurations.
* You can view all configuration in XML and inspect **Dependency Hierarchy** to see all transitive dependencies.
* **Country Table Creation**
* **SQL**
* sql
* Copy code

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');

**Persistence Class**

**📄 Country.java**

**Package:** com.cognizant.ormlearn.model

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="co\_code")

private String code;

@Column(name="co\_name")

private String name;

// Getters and Setters

@Override

public String toString() {

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

}

}

**Notes:**

* @Entity: Declares it as a JPA entity.
* @Table: Specifies table mapping.
* @Id: Marks primary key.
* @Column: Maps Java fields to table columns.
* **Repository Interface**
* **📄 CountryRepository.java**
* **Package:** com.cognizant.ormlearn.repository

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> {

}

**Service Class**

**📄 CountryService.java**

**Package:** com.cognizant.ormlearn.service

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();

}

}

**Testing in Main Class**

**📄 OrmLearnApplication.java**

java

Copy code

package com.cognizant.ormlearn;

import java.util.List;

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;

@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);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

}

Expected Log Output

Inside main

Start

countries=[Country [code=IN, name=India], Country [code=US, name=United States of America]]

End

**Conclusion**

In this exercise, we have successfully:

* Created a Spring Boot project using Spring Initializr.
* Configured MySQL and Spring Data JPA.
* Defined Country entity, repository, and service classes.
* Retrieved data from MySQL using Spring Data JPA and verified it via logs.