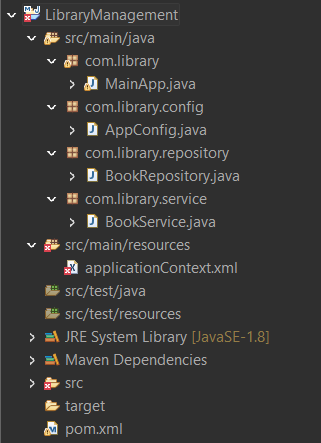
**Exercise 1: Configuring a Basic Spring Application**

**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.

**pom.xml:**

<project >

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0</version>

<dependencies>

<!-- Spring Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

</dependencies>

</project>

1. **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.

**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>

1. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.

**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 addBook() {

bookRepository.saveBook();

System.out.println("Book added via service.");

}

}

* + Create a package **com.library.repository** and add a class **BookRepository**.

**BookRepository.java:**

package com.library.repository;

import org.springframework.stereotype.Repository;

*@Repository*

public class BookRepository {

public void saveBook() {

System.***out***.println("Book saved to repository.");

}

}

1. **Run the Application:**

Create a main class to load the Spring context and test the configuration.

**MainApp.java:**

package com.library;

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");

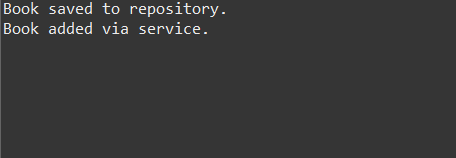
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook();

}

}

**Output:**



**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**.

**In Exercise 1 I have Completed the Step.**

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

System.out.println("Setter Injection: BookRepository has been injected into BookService.");

this.bookRepository = bookRepository;

}

public void addBook() {

bookRepository.saveBook();

System.out.println("Book added via service.");

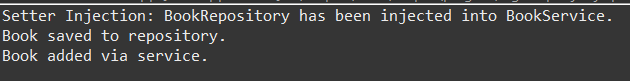
}

}

1. **Test the Configuration:**
   * Run the **Library Management Application** main class to verify the dependency injection.

**Same MainApp.java done in Exercise 1.**

**OUTPUT:**

****

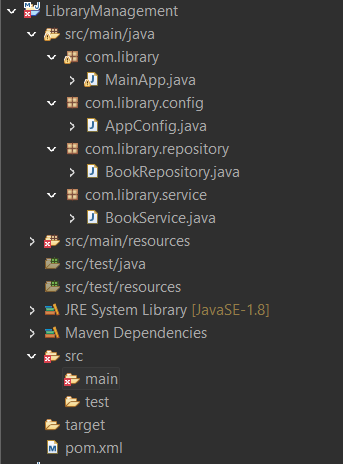
**Exercise 3: Creating and Configuring a Maven Project**

**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**.



1. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
2. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**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</version>

<packaging>jar</packaging>

<properties>

**<!-- Java version -->**

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

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

</properties>

<dependencies>

**<!-- Spring Core and Context -->**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

**<!-- Spring AOP -->**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

**<!-- Spring Web MVC -->**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.20</version>

</dependency>

</dependencies>

<build>

<plugins>

**<!-- Maven Compiler Plugin for Java 1.8 -->**

<plugin>

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Exercise 4 : Spring Data JPA - Quick Example**

**IN Database : MySQL:**

**I Have MySQL 5.5 Command Line Client:**

* + 1. **Start MySQL Command Line Client:**

mysql -u root -p

* + 1. **Create the ormlearn schema:**

create database ormlearn;

use ormlearn;

* + 1. **Create the country table:**

create table country (

code varchar(2) primary key,

name varchar(50)

);

* + 1. **Insert sample data:**

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

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

**IN ECLIPSE IDE:**

**src/main/resources/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

# Console Log Format

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

# DB 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=Vinay

# Hibernate Configuration

spring.jpa.hibernate.ddl-auto=validate

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

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

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "name")

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;

}

@Override

public String toString() {

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

}

}

**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> {

}

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

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import jakarta.transaction.Transactional;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

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

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();

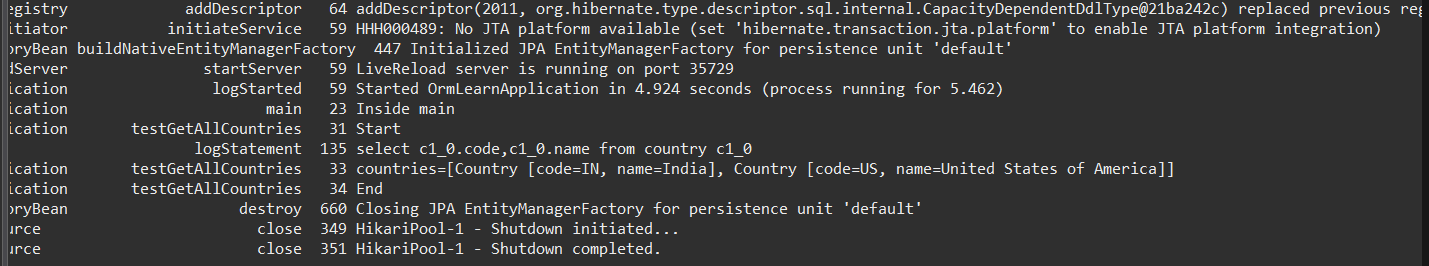
LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**OUTPUT:**

****

**Exercise 5: Difference between JPA, Hibernate and Spring Data JPA** 

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

**What it is**:

* **JPA is a specification** (JSR 338) provided by Java EE (now Jakarta EE).
* It defines **standards** for ORM (Object-Relational Mapping).
* JPA provides interfaces and annotations like @Entity, @Id, EntityManager, etc.
* **Does not have an implementation** — it needs a provider like Hibernate.

**Main Features**:

* Mapping Java objects to database tables.
* CRUD operations using EntityManager.
* Query language (JPQL).

**Example**:

@Entity

public class Employee {

@Id

private Integer id;

private String name;

}

**2. Hibernate**

**What it is**:

* Hibernate is a **popular ORM tool** that **implements the JPA specification**.
* It also offers **extra features beyond JPA**, like caching, batch processing, HQL, etc.

**Responsibilities**:

* Handles session management, transaction control.
* Implements the behavior defined in JPA.

**Traditional Hibernate Example**:

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

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

tx.commit();

session.close();

return employeeId;

}

**Downside**:

* More boilerplate code (open/close session, begin/commit transaction).
* Manual exception handling.

**CODE Snippets Analysis:**

|  |  |
| --- | --- |
| **Aspect** | **Done Manually** |
| Opening session | factory.openSession() |
| Beginning TX | session.beginTransaction() |
| Saving entity | session.save(employee) |
| Commit/Rollback | Explicit |
| Exception handling | Explicit |
| Closing session | Required |

**3. Spring Data JPA**

**What it is**:

* A **Spring-based abstraction** built **on top of JPA (and thus Hibernate)**.
* **It is not a JPA implementation**.
* Automatically implements common data access patterns via interfaces.
* Removes most of the boilerplate code (repositories, queries, etc.)

**Responsibilities**:

* Reduces the need to write DAO classes.
* Provides **JpaRepository, CrudRepository, etc.** to work with entities.
* Integrates with Spring’s @Transactional and dependency injection.

**Example**:

**EmployeeRespository.java:**

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

**EmployeeService.java:**

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**CODE Snippets Analysis:**

|  |  |
| --- | --- |
| **Aspect** | **Who Handles It?** |
| Opening session | Spring + JPA |
| Beginning TX | @Transactional |
| Saving entity | save() from JpaRepository |
| Commit/Rollback | Spring AOP |
| Exception handling | Spring exception translation |
| Closing session | Spring |

**CODE Snippets Comparison Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **Hibernate** | **JPA** | **Spring Data JPA** |
| Code verbosity | High | Moderate | Very Low |
| Session management | Manual (Session) | Manual (EntityManager) | Auto by Spring |
| Transaction control | Manual (Transaction) | Manual (EntityTransaction) | Auto (@Transactional) |
| Save/persist entity | session.save() | em.persist() | repository.save() |
| Exception handling | Manual | Manual | Auto (Spring-managed) |
| Boilerplate code | High | Medium | Minimal |
| Ease of testing | Difficult | Medium | Easy |
| Framework coupling | Tight to Hibernate | Standard JPA API | Abstracted, Spring-centric |

**Overall Comparison Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| Type | Specification | Implementation | Framework/Abstraction over JPA |
| Boilerplate Code | Medium | High | Very Low |
| Implementation Needed | Yes (e.g. Hibernate) | N/A | Uses JPA + implementation (e.g. Hibernate) |
| Transactions | Programmatic or Anno. | Programmatic/Anno. | Handled via Spring annotations |
| Repository Support | No | Manual | Yes (JpaRepository) |
| Ease of Use | Moderate | Complex | Easiest |
| Maintained by | Jakarta EE (spec) | Red Hat | Spring Team |

**Conclusion:**

* JPA is just a set of rules — a specification.
* Hibernate is an implementation of JPA + additional features.
* Spring Data JPA is a tool that makes working with JPA and Hibernate easier by eliminating boilerplate code and managing common CRUD operations with minimal code.

**References Used:**

* [DZone Article on Hibernate vs Spring Data JPA](https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1)
* [JavaWorld on JPA](https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html)