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

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

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

}

}

**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(String bookName) {

bookRepository.save(bookName);

}

}

**MainApp.java**

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

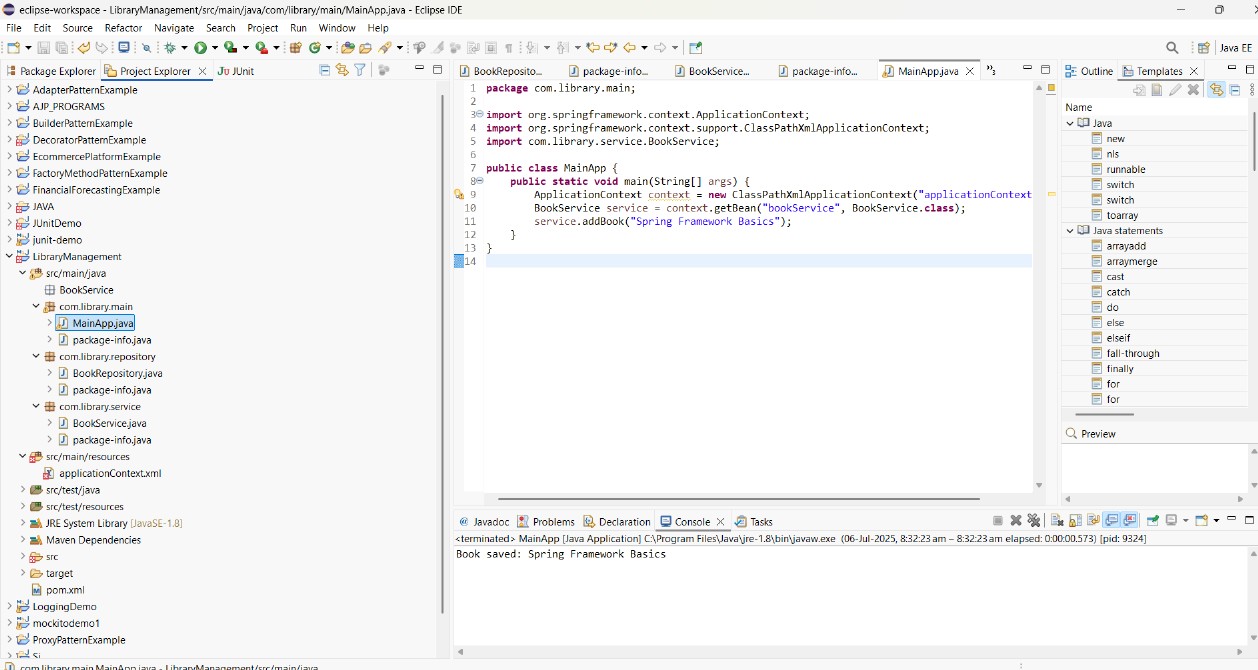
public static void main(String[] args) {

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

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

service.addBook("Spring Framework Basics");

}

}

**Exercise 2: Implementing Dependency Injection**

**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(String bookName) {

System.out.println("BookService: Adding book - " + bookName);

bookRepository.save(bookName);

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

System.out.println("BookRepository: Saving book - " + bookName);

}

}

**MainApp.java**

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) {

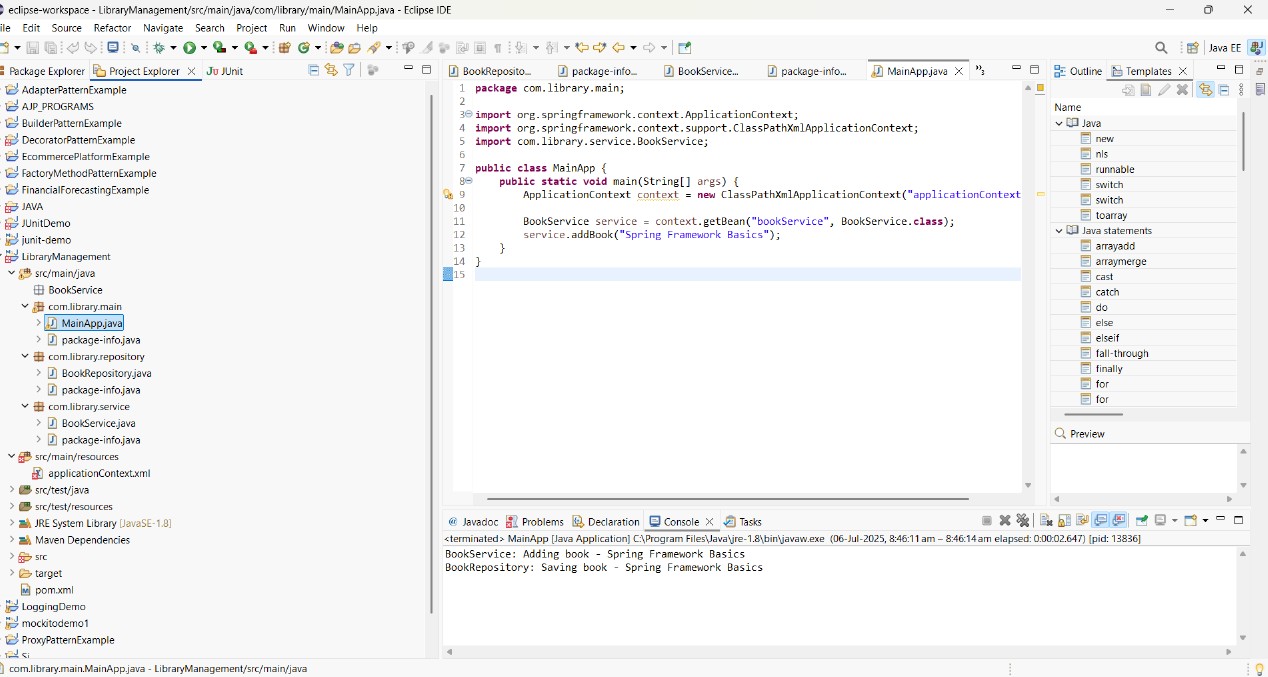
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

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

service.addBook("Spring Framework Basics");

}

**}**



**Exercise 4: Creating and Configuring a Maven Project**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("Book saved to the database.");

}

}

**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 save() {

bookRepository.saveBook();

}

}

**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>

**LibraryManagementApplication.java**

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) {

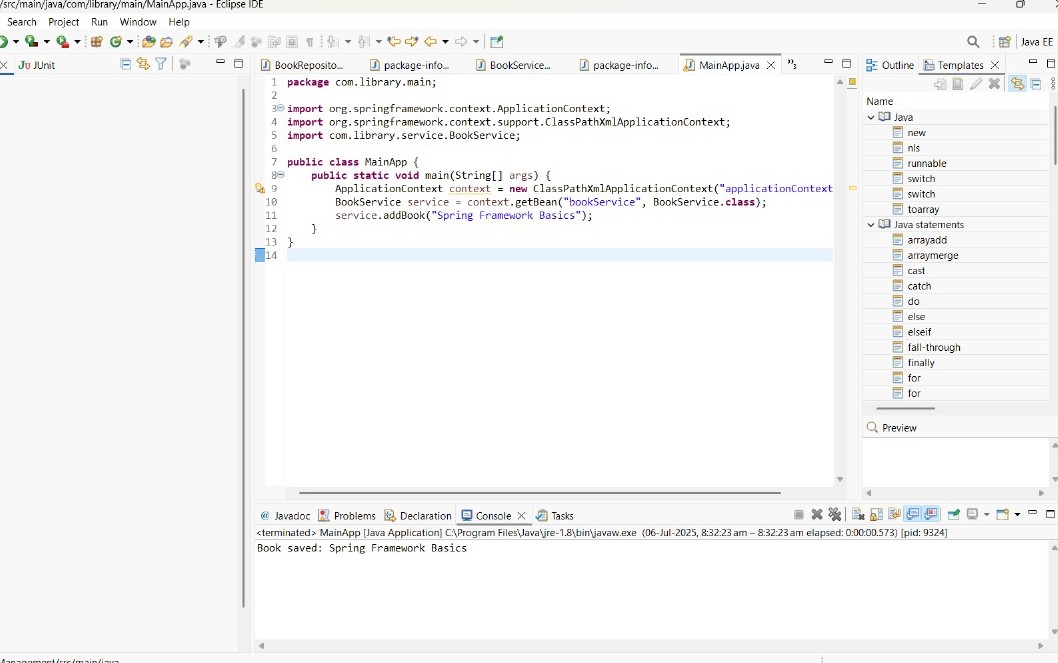
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.save();

}

}



**Hands on 1**

**Spring Data JPA - Quick Example**

**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 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

**Country.java**

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**

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> {

}

**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();

}

}

**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);

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");

}

}

**MySQL**

CREATE DATABASE ormlearn;

USE ormlearn;

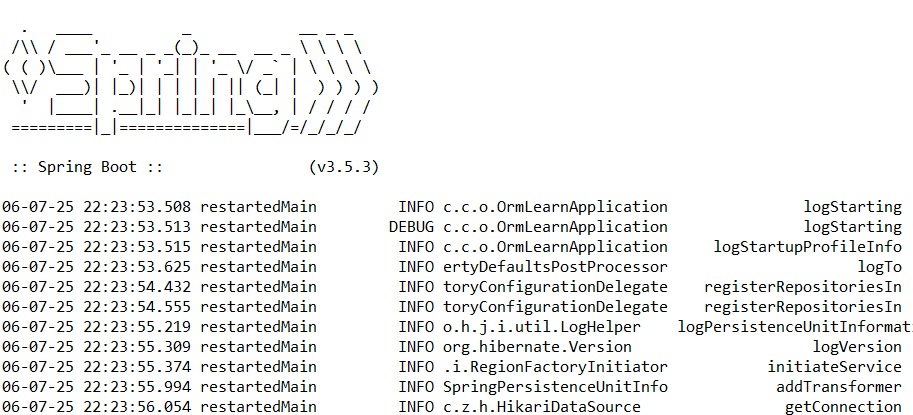
CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India'), ('US', 'United States of America');



**Hands on 4**

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

**Java Persistence API (JPA)**

* JPA stands for Java Persistence API.
* It is a specification (JSR 338) provided by Java EE to define a standard for persisting Java objects in relational databases.
* JPA itself does not provide any implementation — it only defines:
* Interfaces like EntityManager, Entity, Query, etc.
* Vendor-independent – can be implemented by ORM tools like Hibernate, EclipseLink, etc.

**Hibernate**

* Hibernate is an Object-Relational Mapping (ORM) tool for Java.
* It is one of the most popular implementations of JPA.
* Adds more features beyond the JPA specification:
  + HQL (Hibernate Query Language)
  + First- and second-level caching
  + Dirty checking
  + Lazy loading
* Can work with:
  + JPA-based APIs (via JPA annotations and interfaces)
  + Native Hibernate APIs

**Spring Data JPA**

* A module of the Spring Framework built on top of JPA.
* Abstraction layer over JPA to reduce boilerplate code.
* Commonly uses Hibernate as the default JPA provider.
* Main Features:
  + Auto-generates repository implementations (e.g., JpaRepository)
  + Supports method name-based query derivation (e.g., findByUsername())
  + Seamlessly integrates with Spring Boot

**Summary of Differences**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification | ORM Implementation | Spring Abstraction over JPA |
| **Provided By** | Java EE | Hibernate Project | Spring Framework |
| **Boilerplate Reduction** | No | Partial | Yes |
| **Implementation** | None | Yes (implements JPA) | Uses Hibernate or other JPA providers |
| **Main Role** | API Definition for Persistence | ORM framework | Simplifies data access using JPA |
| **Extra Features** | None | Caching, HQL, Lazy loading | Auto-repo generation, Query derivation |

**Use Cases**

| **Scenario** | **Recommended Option** |
| --- | --- |
| You want a standardized API for persistence | JPA |
| You need fine control and advanced ORM features | Hibernate |
| You prefer minimal code with Spring integration | Spring Data JPA |

Great! Here's a cleaned-up and well-structured walkthrough for building a Spring Data JPA application using Spring Boot, Hibernate, H2, and a simple REST API to manage books.