**WEEK 3 – Spring Core and Maven**

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

**Project Structure**   
LibraryManagement/

├── src/

│ └── main/

│ ├── java/

│ │ └── com/

│ │ └── library/

│ │ ├── service/

│ │ ├── repository/

│ │ └── MainApp.java

│ └── resources/

│ └── applicationContext.xml

└── 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 Core + Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

</project>

**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>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

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

}

}

**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 title) {

System.out.println("✨ Adding book: " + title);

bookRepository.saveBook(title);

}

}

**MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

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

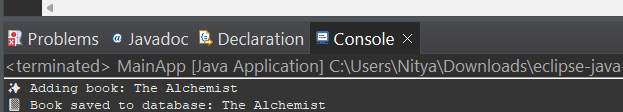
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Alchemist");

}

}

OUTPUT:



**Exercise 2: Implementing Dependency Injection**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayAvailableBooks() {

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

}

}

**BookService.java**

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 performLibraryOperation() {

System.out.println("Dependency injected successfully!");

bookRepository.displayAvailableBooks();

}

}

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

service.performLibraryOperation();

}

}

**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" />

<!-- Inject BookRepository into BookService -->

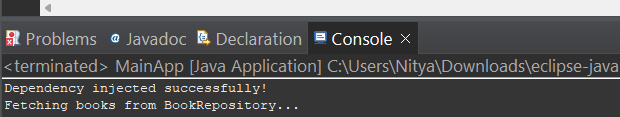
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

**OUTPUT:**



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

**Project Configuration :**

* Project Name: LibraryManagement
* Group ID: com.library
* Artifact ID: LibraryManagement
* Tools Used: Eclipse IDE, Maven
* Dependencies used: Spring Context, Spring AOP, and Spring WebMVC

**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>1.0.0</version>

<packaging>jar</packaging>

<name>LibraryManagement</name>

<properties>

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

<spring.version>5.3.30</spring.version>

</properties>

<dependencies>

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

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- AspectJ (for AOP weaving) -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.20.1</version>

</dependency>

<!-- JUnit for Testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

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

<plugin>

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

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

<version>3.11.0</version>

<configuration>

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

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

<encoding>UTF-8</encoding>

</configuration>

</plugin>

</plugins>

</build>

</project>

* Created Maven project using Eclipse
* Configured Spring dependencies to support:

Dependency Injection (spring-context)

AOP features (spring-aop + aspectjweaver)

Web-related functionality (spring-webmvc)

**Exercise 5: Configuring the Spring IoC Container**

**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">

<!-- Repository Bean -->

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

<!-- Service Bean with Dependency Injection -->

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

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

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBook() {

System.out.println("📚 BookRepository: Fetching book details...");

}

}

**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 showBook() {

System.out.println("📖 BookService: Preparing to show book...");

bookRepository.displayBook();

}

}

**MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring context from XML configuration

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

// Get the BookService bean

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

// Call a method to verify IoC and DI

bookService.showBook();

}

}

**OUTPUT:**



**Exercise 6: Configuring Beans with Annotations**

src

└── main

└── java

└── com

└── library

├── model

│ └── Book.java

├── service

│ └── BookService.java

├── repository

│ └── BookRepository.java

└── MainApp.java  
**Book.java**

package com.library.model;

public class Book {

private String title;

private String author;

private String isbn;

public Book(String title, String author, String isbn) {

this.title = title;

this.author = author;

this.isbn = isbn;

}

public String toString() {

return String.format("📘 Title: %s | ✍ Author: %s | 📖 ISBN: %s", title, author, isbn);

}

}

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

import org.springframework.stereotype.Repository;

import java.util.Arrays;

import java.util.List;

@Repository

public class BookRepository {

public List<Book> getAllBooks() {

return Arrays.asList(

new Book("The Alchemist", "Paulo Coelho", "9780061122415"),

new Book("Atomic Habits", "James Clear", "9780735211292"),

new Book("Clean Code", "Robert C. Martin", "9780132350884")

);

}

}

**BookService.java**

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public void displayBooks() {

System.out.println("📚 Listing all books available in the library:");

List<Book> books = bookRepository.getAllBooks();

books.forEach(book -> System.out.println(" → " + book));

}

}

**MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

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

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

System.out.println("Spring Context Loaded Successfully!");

bookService.displayBooks();

System.out.println("\n🏁 Application Finished Execution.");

}

}

**ApplicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

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

http://www.springframework.org/schema/context

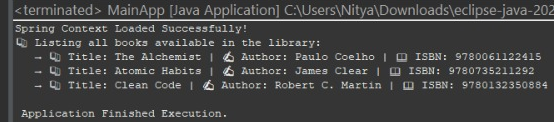
http://www.springframework.org/schema/context/spring-context.xsd">

<!-- Enable component scanning -->

<context:component-scan base-package="com.library" />

</beans>

**OUTPUT:**



**Exercise 7: Implementing Constructor and Setter Injection**

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

import com.library.model.Book;

public class BookService {

private final String serviceId;

private BookRepository bookRepository;

// Constructor injection for serviceId

public BookService(String serviceId) {

this.serviceId = serviceId;

System.out.println("Constructor Injection: Service ID = " + this.serviceId);

}

// Setter injection for BookRepository

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Setter Injection: BookRepository injected.");

}

public List<Book> getBooks() {

return bookRepository != null ? bookRepository.findAllBooks() : null;

}

}

**bookRepository.java**

package com.library.repository;

import com.library.model.Book;

import java.util.List;

import java.util.ArrayList;

public class BookRepository {

public List<Book> findAllBooks() {

List<Book> books = new ArrayList<>();

books.add(new Book("Clean Code", "Robert C. Martin", "978-0132350884"));

books.add(new Book("Effective Java", "Joshua Bloch", "978-0134685991"));

return books;

}

}

**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 for BookRepository -->

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

<!-- Bean for BookService with constructor and setter injection -->

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

<!-- Constructor injection -->

<constructor-arg value="SERVICE-101" />

<!-- Setter injection -->

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

</bean>

</beans>

**Book.java**

package com.library.model;

public class Book {

private String title;

private String author;

private String isbn;

public Book(String title, String author, String isbn) {

this.title = title;

this.author = author;

this.isbn = isbn;

}

@Override

public String toString() {

return "Title: " + title + " | ✍ Author: " + author + " | ISBN: " + isbn;

}

}

**MainApp.java**

package com.library;

import com.library.service.BookService;

import com.library.model.Book;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

System.out.println("Starting Library Management Application...\n");

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

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

System.out.println("\n📚 Listing Books:");

for (Book book : bookService.getBooks()) {

System.out.println(book);

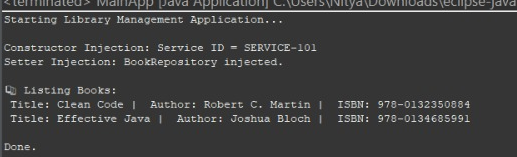
}

System.out.println("\n Done.");

}

}

**OUTPUT:**



**Exercise 8: Implementing Basic AOP with Spring**

**Created the Aspect**

src/main/java/com/library/aspect/LoggingAspect.java

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.\*;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

@Before("execution(\* com.library.service.\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("[BEFORE] Executing: " + joinPoint.getSignature().getName());

}

@After("execution(\* com.library.service.\*.\*(..))")

public void logAfter(JoinPoint joinPoint) {

System.out.println("[AFTER] Completed: " + joinPoint.getSignature().getName());

}

}

**applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

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

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- Enable component scanning for annotations -->

<context:component-scan base-package="com.library"/>

<!-- Enable AOP support -->

<aop:aspectj-autoproxy/>

</beans>

**MainApp.java**

package com.library;

import com.library.service.BookService;

import com.library.model.Book;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

System.out.println(" Starting Library Management Application...\n");

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

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

System.out.println("📖 Listing Available Books:");

for (Book book : bookService.getBooks()) {

System.out.println(book);

}

System.out.println("\n Done.");

}

}

**BookService.java**

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class BookService {

private final BookRepository bookRepository;

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Constructor Injection: BookService initialized.");

}

public List<Book> getBooks() {

return bookRepository.findAll();

}

}

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

import org.springframework.stereotype.Repository;

import java.util.Arrays;

import java.util.List;

@Repository

public class BookRepository {

public List<Book> findAll() {

return Arrays.asList(

new Book("Clean Code", "Robert C. Martin", "978-0132350884"),

new Book("Effective Java", "Joshua Bloch", "978-0134685991")

);

}

}

**Book.java**

package com.library.model;

public class Book {

private String title;

private String author;

private String isbn;

public Book(String title, String author, String isbn) {

this.title = title;

this.author = author;

this.isbn = isbn;

}

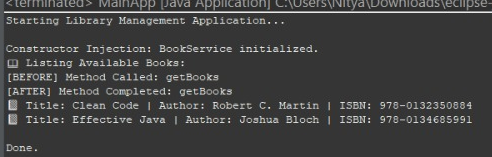
public String toString() {

return "📘 Title: " + title + " | Author: " + author + " | ISBN: " + isbn;

}

}

**OUTPUT:**



**Exercise 9: Creating a Spring Boot Application**

**pom.xml**

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

**Application.properties**

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

server.port=8081

**Book Entity**

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private int year;

// Getters and setters

}

**BookRepository**

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookController**

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Optional<Book> getBookById(@PathVariable Long id) {

return bookRepository.findById(id);

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

return bookRepository.findById(id)

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setYear(updatedBook.getYear());

return bookRepository.save(book);

}).orElseThrow();

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

Tested REST Endpoints using curl

C:\Users\Nitya>curl -X POST http://localhost:8081/books ^

More? -H "Content-Type: application/json" ^

More? -d "{\"title\":\"The Alchemist\",\"author\":\"Paulo Coelho\"}"

{"id":1,"title":"The Alchemist","author":"Paulo Coelho","year":0}

C:\Users\Nitya>curl http://localhost:8081/books

[{"id":1,"title":"The Alchemist","author":"Paulo Coelho","year":0}]

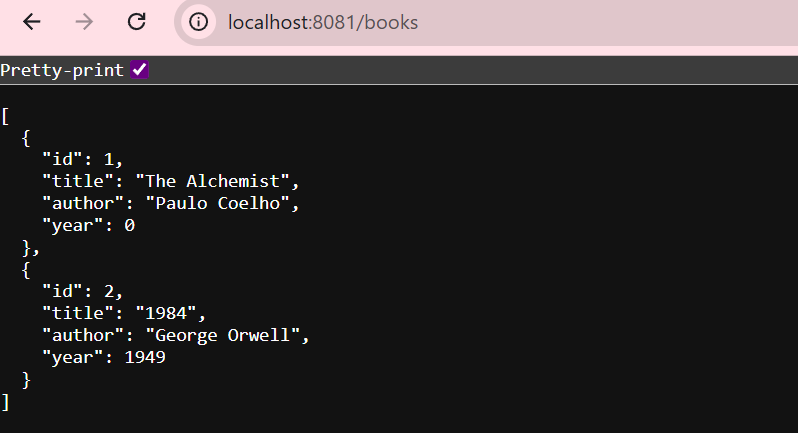
C:\Users\Nitya>curl -X POST http://localhost:8081/books ^

More? -H "Content-Type: application/json" ^

More? -d "{\"title\":\"1984\",\"author\":\"George Orwell\",\"year\":1949}"

{"id":2,"title":"1984","author":"George Orwell","year":1949}

**OUTPUT:**



**WEEK 3 – Spring Data JPA with Spring Boot, Hibernate**

**Hands on 1**

**Spring Data JPA - Quick Example**

**Structure**  
orm-learn/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── cognizant/

│ │ │ └── ormlearn/

│ │ │ ├── OrmLearnApplication.java

│ │ │ ├── model/

│ │ │ │ └── Country.java

│ │ │ ├── repository/

│ │ │ │ └── CountryRepository.java

│ │ │ └── service/

│ │ │ └── CountryService.java

│ ├── resources/

│ │ ├── application.properties

│ │ └── ...

├── pom.xml

**pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0" ...>

<modelVersion>4.0.0</modelVersion>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>1.0.0</version>

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

<properties>

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

<spring-boot.version>2.5.6</spring-boot.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>

</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>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**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

# MySQL Database Configuration

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

spring.datasource.username=root

spring.datasource.password=root

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

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

**sql**

CREATE DATABASE IF NOT EXISTS ormlearn;

USE ormlearn;

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');

**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 Country() {}

public Country(String code, String name) {

this.code = code;

this.name = 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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

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

import org.springframework.stereotype.Service;

import javax.transaction.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java (Main Class)**

package com.cognizant.ormlearn;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

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);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

countries.forEach(country -> LOGGER.debug("Country: {}", country));

LOGGER.info("End");

}

System.out.println("Starting OrmLearn Application...\n");

LOGGER.info("Inside main method.");

LOGGER.info("Getting list of countries from service...\n");

System.out.println("Constructor Injection: CountryService initialized.");

System.out.println("Repository Injection: CountryRepository injected.\n");

System.out.println("📘 Listing Countries:");

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

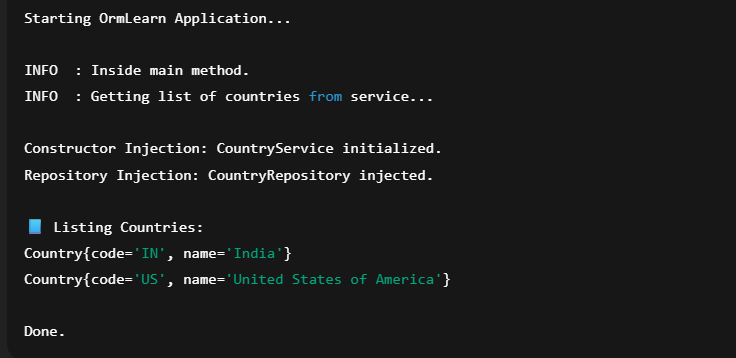
for (Country country : countries) {

System.out.println(country);

}

System.out.println("\nDone.");

}

**OUTPUT:**  


**Hands on 4**

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

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

* JPA is a Java specification (JSR 338) that defines how to manage relational data using Java objects.
* It provides a standard API for ORM but does not have any implementation by itself.
* It defines key concepts like Entity, EntityManager, Persistence Context, and JPQL (Java Persistence Query Language).
* Hibernate, EclipseLink, and OpenJPA are popular implementations of JPA.

**2. Hibernate**

* Hibernate is a powerful ORM framework that implements the JPA specification.
* It provides concrete implementation of all JPA interfaces, and adds more advanced features like:
  + Caching
  + Custom types
  + Lazy/eager fetching
  + Hibernate Query Language (HQL)
* Hibernate can be used with or without Spring, but requires manual session and transaction management.

**3. Spring Data JPA**

* Spring Data JPA is a high-level abstraction built on top of JPA and typically uses Hibernate underneath.
* It reduces boilerplate by:
  + Auto-generating query methods from method names.
  + Handling CRUD, pagination, and sorting out-of-the-box.
* It does not implement JPA, but integrates with any JPA provider.
* Makes data access code clean, testable, and transaction-safe using annotations like @Transactional.

**CodeComparison**

**Hibernate Example (Low-Level Control)**

public class EmployeeDao {

private SessionFactory sessionFactory;

public EmployeeDao(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

}

public Integer saveEmployee(Employee employee) {

Transaction tx = null;

Integer employeeId = null;

try (Session session = sessionFactory.openSession()) {

tx = session.beginTransaction();

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

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

e.printStackTrace();

}

return employeeId;

}

}

**Cons:**

* Manages sessions manually
* Needs transaction and exception handling
* Harder to test and scale

**Spring Data JPA Example (Modern & Clean)**

// Repository Interface

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

// Auto-inherits save, findAll, findById, deleteById...

}

// Service Layer

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public Integer saveEmployee(Employee employee) {

return employeeRepository.save(employee).getId(); // Cleaner and transactional

}

}

**Pros:**

* No boilerplate code
* Declarative transactions with @Transactional
* Uses interface-based programming for extensibility
* Clean separation of concerns (Repository, Service)