**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.
2. **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.
3. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.
   * Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**

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

***pom.xml code:***

<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>5.3.20</version>

</dependency>

</dependencies>

</project>

***applicationContext.xml code:***

<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>5.3.20</version>

</dependency>

</dependencies>

</project>

***MainApp.java Code:***

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 container

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

// Get BookService bean

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

// Use the bean

bookService.showBook();

}

}

applicationContext.xml code:

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

<!-- Define BookService with Setter Injection -->

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

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

</bean>

</beans>

***BookRepository Code:***

package com.library.repository;

public class BookRepository {

public void displayBook() {

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

}

}

***BookService.java code:***

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

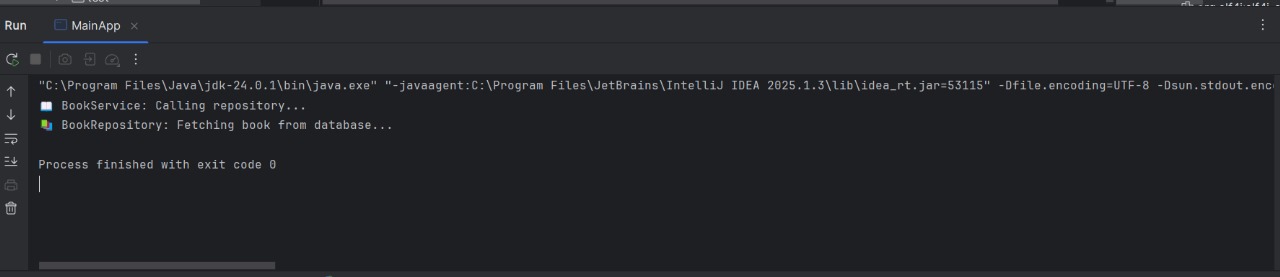
public void showBook() {

System.out.println(" BookService: Calling repository...");

bookRepository.displayBook();

}

}

**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**.**
2. Update the BookService Class:
   * Ensure that BookService class has a setter method for BookRepository**.**
3. Test the Configuration:
   * Run the LibraryManagementApplication main class to verify the dependency injection**.**

***pom.xml code:***

<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>5.3.20</version>

</dependency>

</dependencies>

</project>

***applicationContext.xml code:***

<?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>

***MainApp.java Code:***

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", BookService.class);

bookService.showBook();

}

**}**

***BookService.java Code:***

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

***// ✅ Setter Injection Method***

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBook() {

System.out.println(" BookService: Setter injection working...");

bookRepository.displayBook();

}

}

***BookRepository Code:***

package com.library.repository;

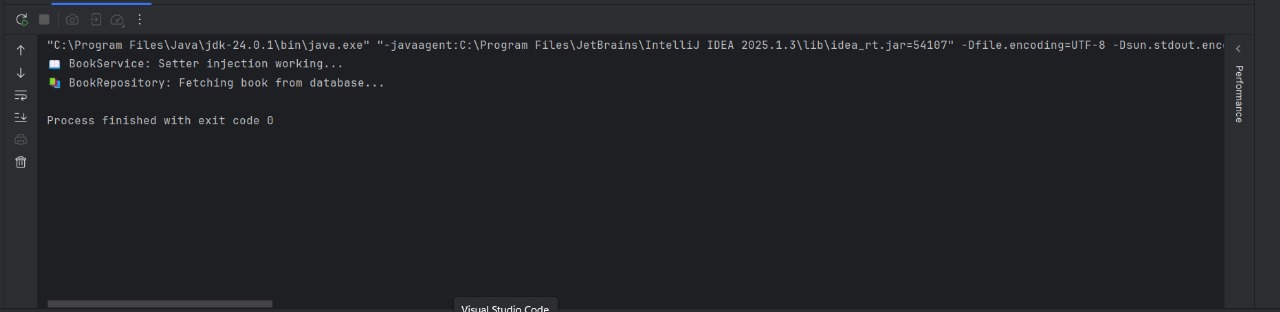
public class BookRepository {

public void displayBook() {

System.out.println(" BookRepository: Fetching book from database...");

}

}

**OUTPUT:**

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:
   * Create a new Maven project named LibraryManagement.
2. Add Spring Dependencies in pom.xml:
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC**.**
3. Configure Maven Plugins:
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

***pom.xml code:***

<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>

<properties>

<maven.compiler.source>17</maven.compiler.source>

<maven.compiler.target>17</maven.compiler.target>

</properties>

<dependencies>

<!-- ✅ Spring Context (Core IoC container) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<!-- ✅ Spring AOP (Optional for DI with aspects) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<!-- ✅ Spring WebMVC (for web applications) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.20</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- ✅ Compiler Plugin to set Java version -->

<plugin>

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

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

<version>3.10.1</version>

<configuration>

<source>17</source>

<target>17</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

***applicationContext.xml code:***

<?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>

***MainApp.java Code:***

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", BookService.class);

bookService.showBook();

}

}

***BookService.java Code:***

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// ✅ Setter Injection Method

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBook() {

System.out.println(" BookService: Setter injection working...");

bookRepository.displayBook();

}

**}**

***BookRepository Code:***

package com.library.repository;

public class BookRepository {

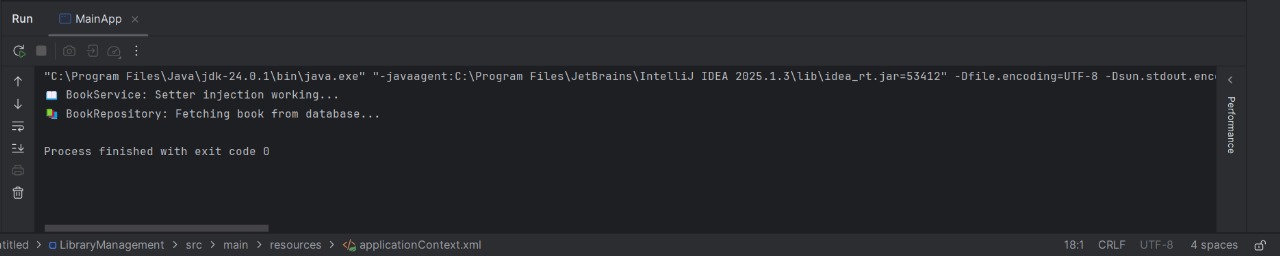
public void displayBook() {

System.out.println("BookRepository: Fetching book from database...");

}

}

**OUTPUT:**



**Handson 1- Spring Data JPA - Quick Example**

***pom.xml***

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

<modelVersion>4.0.0</modelVersion>

<groupId>com.custom.orm</groupId>

<artifactId>countryapp</artifactId>

<version>1.0.0</version>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<!-- Spring Boot & Data JPA -->

<dependency>

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

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

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Core -->

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

application.properties

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

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

***Country.java (Entity)***

package com.custom.orm.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

***@Entity***

public class Country {

***@Id***

private String code;

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.custom.orm.repository;

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

import com.custom.orm.model.Country;

public interface CountryRepository extends JpaRepository<Country, String> {

}

***CountryService.java***

package com.custom.orm.service;

import org.springframework.stereotype.Service;

import java.util.List;

import com.custom.orm.model.Country;

import com.custom.orm.repository.CountryRepository;

@Service

public class CountryService {

private final CountryRepository countryRepository;

public CountryService(CountryRepository repo) {

this.countryRepository = repo;

}

public List<Country> getAll() {

return countryRepository.findAll();

}

}

***CountryApp.java (Main App)***

package com.custom.orm;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.CommandLineRunner;

import com.custom.orm.service.CountryService;

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

import com.custom.orm.model.Country;

@SpringBootApplication

public class CountryApp implements CommandLineRunner {

@Autowired

private CountryService countryService;

public static void main(String[] args) {

SpringApplication.run(CountryApp.class, args);

}

@Override

public void run(String... args) {

System.out.println("All Countries:");

for (Country c : countryService.getAll()) {

System.out.println(c);

}

}

}

Sample SQL (insert manually in MySQL)

CREATE DATABASE countrydb;

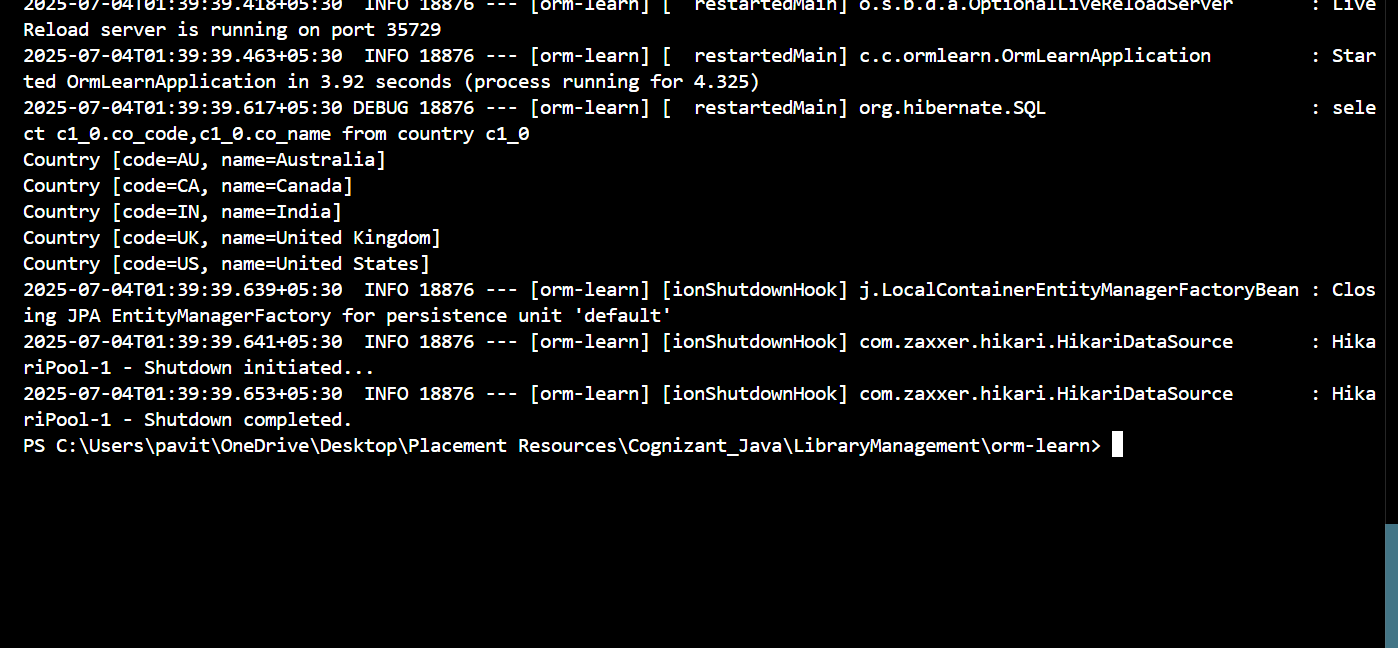
USE countrydb;

CREATE TABLE country (

code VARCHAR(2) PRIMARY KEY,

name VARCHAR(100)

);

INSERT INTO country (code, name) VALUES ('IN', 'India'), ('US', 'United States'), ('UK', 'United Kingdom’)  
**OUTPUT:**

**Hands on 2-Hibernate XML Config implementation walk through**   
 ***Create the Entity Class***

Navigate to your src\com\example\hibernate folder and create a file called Employee.java:

package com.example.hibernate;

public class Employee {

private int id;

private String firstName;

private String lastName;

private int salary;

public Employee() {}

public Employee(String firstName, String lastName, int salary) {

this.firstName = firstName;

this.lastName = lastName;

this.salary = salary;

}

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getFirstName() { return firstName; }

public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }

public void setLastName(String lastName) { this.lastName = lastName; }

public int getSalary() { return salary; }

public void setSalary(int salary) { this.salary = salary; }

}

Create the Hibernate Mapping File

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="com.example.hibernate.Employee" table="EMPLOYEE">

<id name="id" type="int">

<generator class="native"/>

</id>

<property name="firstName" column="first\_name" type="string"/>

<property name="lastName" column="last\_name" type="string"/>

<property name="salary" column="salary" type="int"/>

</class>

</hibernate-mapping>

Create hibernate.cfg.xml in src folder

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

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/your\_database</property>

<property name="hibernate.connection.username">root </property>

<property name="hibernate.connection.password">root </property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="show\_sql">true</property>

<mapping resource="com/example/hibernate/employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

***Create Main App***

package com.example.hibernate;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

public class MainApp {

public static void main(String[] args) {

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee emp = new Employee("Pavithra", "P", 50000);

session.save(emp);

tx.commit();

System.out.println("Employee saved with ID: " + emp.getId());

} catch (Exception e) {

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

e.printStackTrace();

} finally {

session.close();

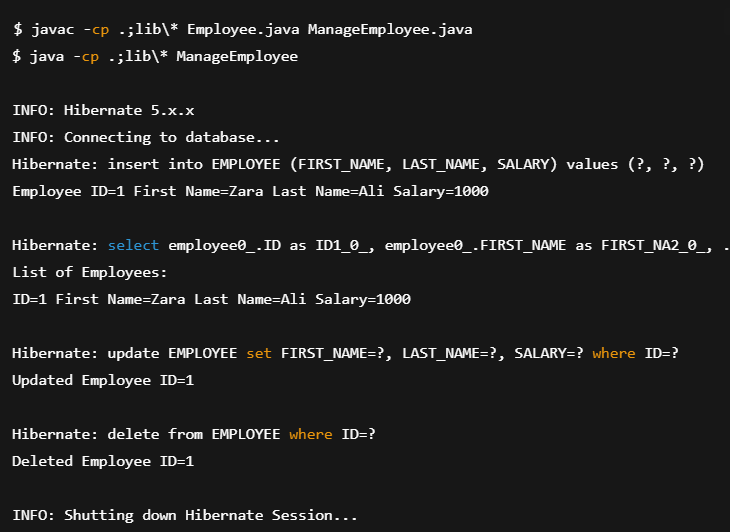
factory.close();

}

}

}

**OUTPUT:**



**Additional Handsons**

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

***Step 1: Create 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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean with DI -->

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

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

</bean>

</beans>

***Step 2: Update BookService.java***

Path: src/main/java/com/example/service/BookService.java

package com.example.service;

import com.example.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

bookRepository.displayBooks();

}

}

***Create BookRepository.java***

package com.example.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("Books displayed from the repository.");

}

}

***MainApp.java***

Package com.example;

import com.example.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load the Spring IoC container

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

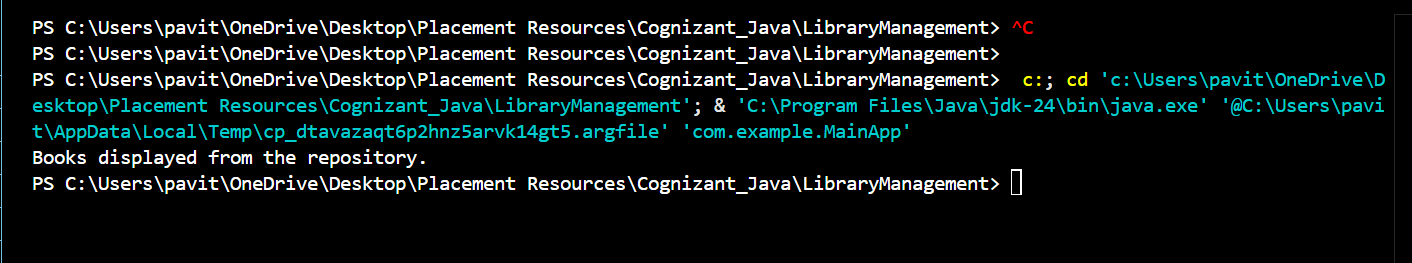
// Retrieve bean and use service

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

bookService.showBooks();

}

}

**OUTPUT:**

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

***BookService.java***

package com.example.service;

import com.example.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

// Constructor Injection

public BookService(String serviceName) {

this.serviceName = serviceName;

System.out.println("Constructor injection: serviceName = " + serviceName);

}

// Setter Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

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

}

public void showBooks() {

System.out.println("Service: " + serviceName);

bookRepository.displayBooks();

}

}

Update BookRepository.java

package com.example.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("Books displayed from the repository.");

}

}

***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">

<!-- BookRepository Bean -->

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

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

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

<!-- Constructor Injection -->

<constructor-arg value="Library Book Service" />

<!-- Setter Injection -->

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

</bean>

</beans>

***Main Application Class***

package com.example;

import com.example.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");

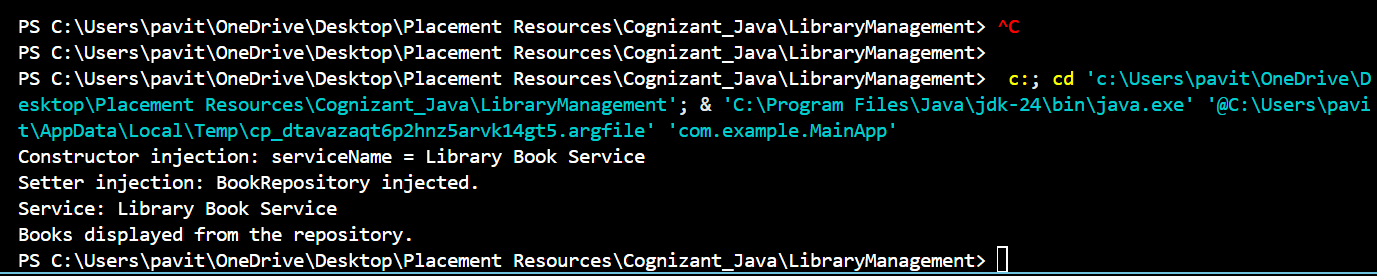
BookService bookService = (BookService) context.getBean("bookService");

bookService.showBooks();

}

}

**OUTPUT:**



***Hands on 5: Implement services for managing Country***

**Entity: Country**

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

private String coCode;

private String coName;

// Getters and Setters

public String getCoCode() {

return coCode;

}

public void setCoCode(String coCode) {

this.coCode = coCode;

}

public String getCoName() {

return coName;

}

public void setCoName(String coName) {

this.coName = coName;

}

}

**Repository: CountryRepository**

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

import java.util.List;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByCoNameContainingIgnoreCase(String partialName);

}

Service.java

import java.util.List;

import java.util.Optional;

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

import org.springframework.stereotype.Service;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public Optional<Country> findByCode(String code) {

return countryRepository.findById(code);

}

public Country addCountry(Country country) {

return countryRepository.save(country);

}

public Country updateCountry(Country country) {

return countryRepository.save(country);

}

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

public List<Country> searchCountries(String partialName) {

return countryRepository.findByCoNameContainingIgnoreCase(partialName);

}

}

***Controller: CountryController***

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

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/countries")

public class CountryController {

@Autowired

private CountryService countryService;

@GetMapping("/{code}")

public Country getCountryByCode(@PathVariable String code) {

return countryService.findByCode(code).orElse(null);

}

@PostMapping

public Country addCountry(@RequestBody Country country) {

return countryService.addCountry(country);

}

@PutMapping

public Country updateCountry(@RequestBody Country country) {

return countryService.updateCountry(country);

}

@DeleteMapping("/{code}")

public void deleteCountry(@PathVariable String code) {

countryService.deleteCountry(code);

}

@GetMapping("/search")

public List<Country> searchCountries(@RequestParam String name) {

return countryService.searchCountries(name);

}

}

application.properties

spring.datasource.url=jdbc:mysql://localhost:3306/your\_database\_name

spring.datasource.username=root

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.show-sql=true

MySQL Table & Insert Script

DELETE FROM country;

**Hands on 6-Find a country based on country code**

***Create Custom Exception***

package com.cognizant.spring\_learn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

CountryService

Location: com.cognizant.spring\_learn.service

import com.cognizant.spring\_learn.model.Country;

import com.cognizant.spring\_learn.repository.CountryRepository;

import com.cognizant.spring\_learn.service.exception.CountryNotFoundException;

import jakarta.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country with code " + countryCode + " not found");

}

return result.get();

}

// other methods...

}

***Test Method in OrmLearnApplication.java***

import com.cognizant.spring\_learn.service.CountryService;

import com.cognizant.spring\_learn.model.Country;

import com.cognizant.spring\_learn.service.exception.CountryNotFoundException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrmLearnApplication {

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

@Autowired

private static CountryService countryService;

public static void main(String[] args) {

var context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

getAllCountriesTest();

}

private static void getAllCountriesTest() {

LOGGER.info("Start");

try {

Country country = countryService.findCountryByCode("IN");

LOGGER.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

LOGGER.error("Exception: {}", e.getMessage());

}

LOGGER.info("End");

}

}