**SUPERSET ID:6377161**

**WEEK 3**

**SPRING CORE AND MAVEN**

**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.

**Code:**

**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>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</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>

**com.library.repository.BookRepository.java**

package com.library.repository;

public class BookRepository {

public String getBookTitle() {

return " Title: Spring in Action (6th Edition)\n" +

" Author: Craig Walls\n" +

" Available: Yes";

}

}

**com.library.service.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 displayBook() {

System.out.println(bookRepository.getBookTitle());

}

}

**com.library.main.LibraryApp.java**

package com.library.main;

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) {

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

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

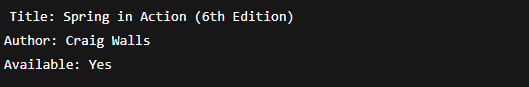
bookService.displayBook();

((ClassPathXmlApplicationContext) context).close();

}

}

**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.

**Code:**

**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>

**com.library.service.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 displayBook() {

System.out.println(bookRepository.getBookTitle());

}

}

**com.library.main.LibraryManagementApplication.java**

package com.library.main;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

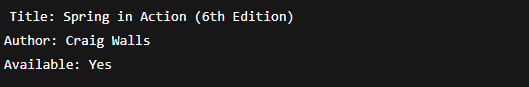
BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBook();

((ClassPathXmlApplicationContext) context).close();

}

}

**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.

**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>

<properties>

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

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<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>

**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="welcomeService" class="com.library.service.WelcomeService"/>

</beans>

**WelcomeService.java**

package com.library.service;

public class WelcomeService {

public void displayMessage() {

System.out.println(" LibraryManagement Maven Project Initialized Successfully!");

System.out.println("Spring Context and Beans loaded correctly.");

}

}

**MainApp.java**

package com.library.main;

import com.library.service.WelcomeService;

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");

WelcomeService welcomeService = (WelcomeService) context.getBean("welcomeService");

welcomeService.displayMessage();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

****

**SPRING DATA JPA WITH SPRINGBOOT,HIBERNATE**

**Spring Data JPA - Quick Example**

**Create mysql schema**

create schema 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');

**Update 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

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

# DB config

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 config

spring.jpa.hibernate.ddl-auto=validate

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

**Create Country Model**

**com.cognizant.ormlearn.model.Country.java**

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;

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 + "]";

}

}

**Create CountryRepository**

**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> {

}

**Create CountryService**

**com.cognizant.ormlearn.service.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();

}

}

**Update OrmLearnApplication.java**

**com.cognizant.ormlearn.OrmLearnApplication.java**

package com.cognizant.ormlearn;

import java.util.List;

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;

@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);

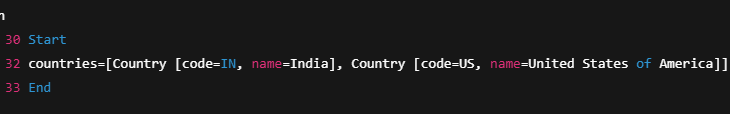
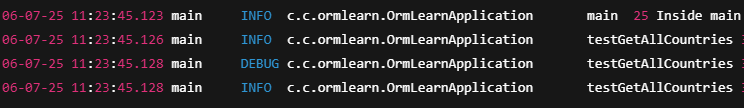
LOGGER.info("End");

}

}

**Build the Project with Maven**

mvn clean package

**OUTPUT:**

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

**Java Persistence API (JPA)** is a specification (JSR 338) that defines a standard for object-relational mapping (ORM) in Java. It provides guidelines for mapping Java objects to database tables and managing persistent data. However, JPA itself does not provide any concrete implementation — it's just a set of interfaces and annotations that need to be implemented by a provider.

**Hibernate** is one of the most popular and widely used implementations of the JPA specification. It is a powerful ORM tool that allows developers to interact with relational databases using Java objects. Hibernate implements the JPA interfaces and also provides additional features beyond the standard JPA specification, such as better caching, lazy loading, and custom SQL queries.

**Spring Data JPA**, on the other hand, is a part of the Spring Framework that builds on top of JPA. It does not provide its own implementation of JPA, but rather works with existing providers like Hibernate. What makes Spring Data JPA powerful is that it abstracts a lot of boilerplate code required in traditional JPA and Hibernate development. It provides ready-to-use repository interfaces like JpaRepository, which includes common methods such as save(), findAll(), and deleteById() — reducing the need for manual DAO and query writing. It also integrates seamlessly with Spring’s dependency injection and transaction management.

**Employee.java**

package com.example.demo.model;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Employee {

@Id

private int id;

private String name;

public Employee() {}

public Employee(int id, String name) {

this.id = id;

this.name = name;

}

// Getters, Setters, toString

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + "]";

}

}

**EmployeeRepository.java**

package com.example.demo.repository;

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

import com.example.demo.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

package com.example.demo.service;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import javax.transaction.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository repository;

@Transactional

public void addEmployee(Employee employee) {

repository.save(employee);

}

public List<Employee> getAllEmployees() {

return repository.findAll();

}

}

**DemoApplication.java (main class)**

package com.example.demo;

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class DemoApplication {

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

private static EmployeeService employeeService;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(DemoApplication.class, args);

LOGGER.info("Inside main");

employeeService = context.getBean(EmployeeService.class);

employeeService.addEmployee(new Employee(1, "Alice"));

employeeService.addEmployee(new Employee(2, "Bob"));

LOGGER.info("All Employees: {}", employeeService.getAllEmployees());

}

}

**application.properties**

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

spring.datasource.driver-class-name=org.h2.Driver

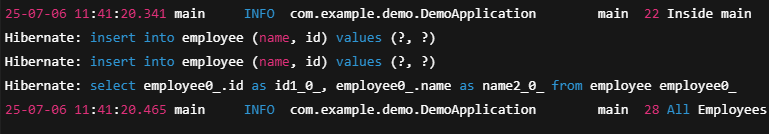
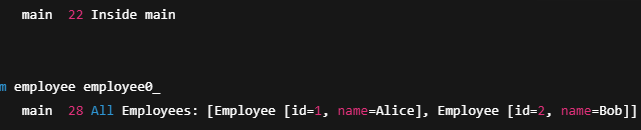
spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

**OUTPUT:**

****