**Week 3 Hands-on Exercises**

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

**1. Set Up a Spring Project**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

**2. Configure the Application Context**

<?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>

**3. Define Service and Repository Classes**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

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

}}

**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.saveBook(bookName);

}

}

**4. Run the Application**

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

bookService.addBook("Spring in Action");

}

}

**Exercise 2: Implementing Dependency Injection**

**1. Modify the XML Configuration:**

<?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>

**2. Update the BookService Class:**

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 listBooks() {

bookRepository.displayAllBooks();

}

}

**3. Create the BookRepository Class:**

package com.library.repository;

public class BookRepository {

public void displayAllBooks() {

System.out.println("Displaying all books in the library...");

}

}

**4. Test the Configuration**

**LibraryManagementApplication.java**

package com.library;

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

bookService.listBooks();

}

}

**Output:**

Displaying all books in the library.

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

**Step 1: Create a New Maven Project**

Create a Maven project named: LibraryManagement

**Step 2: Add Spring Dependencies in pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

**Step 3: Configure Maven Compiler Plugin**

<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>

**Spring Data JPA – Quick Example**

1. **Book.java – Entity Class**

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getAuthor() {

return author;

}

public void setAuthor(String author) {

this.author = author;

}

}

1. **BookRepository.java – Repository Interface**

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

public interface BookRepository extends JpaRepository<Book, Long> {

// No implementation needed

}

1. **BookController.java – REST Controller**

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

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping

public List<Book> getBooks() {

return bookRepository.findAll();

}

}

**4. application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

1. **LibraryManagementApplication.java – Main Class**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

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

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (Interface) | Implementation | Abstraction layer |
| **Purpose** | Defines a standard for ORM in Java | Provides actual implementation of JPA | Simplifies JPA-based data access using repositories |
| **Who Provides It?** | Java EE (Jakarta EE) | Hibernate ORM (by Red Hat) | Spring Framework (by Pivotal/VMware) |
| **What It Does** | Provides annotations and interfaces to map Java objects to database tables | Implements JPA and adds additional features (caching, lazy loading, etc.) | Reduces boilerplate code like DAO implementation |
| **Key Component** | @Entity, @Id, EntityManager | Session, SessionFactory, Configuration | JpaRepository, CrudRepository, @Repository |
| **Boilerplate Code** | More manual coding required | Less than JPA, but still some boilerplate | Minimal code – most CRUD methods auto-implemented |
| **Use Case** | When you want to stick to standard Java and switch providers easily | When you need advanced ORM features | When you want fast development with Spring Boot |