Week 3: Spring Core & 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.

# Procedure:

# Step 1: Set up a Spring Project

# Create a Maven Project named LibraryManagement.

# Add Spring Core dependencies in the pom.xml file.

# Step 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.

# Step 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.

# Step 4: Run the Application

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

# Implementation:

# 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  
 https://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 Class:

package com.library.repository;  
public class BookRepository {  
 public void saveBook(String bookName) {  
 System.*out*.println("Book saved: " + bookName);  
 }  
}

# 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 addBook(String bookName) {  
 System.*out*.println("Adding book: " + bookName);  
 bookRepository.saveBook(bookName);  
 }  
}

# MainApp Class:

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.addBook("Harry Potter and The Deathly Hollows");  
 }  
}

# Output:

# A screenshot of a computer program AI-generated content may be incorrect.

**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.

# Procedure:

# Step 1: Modify the XML Configuration

# Update applicationContext.xml to wire BookRepository into BookService.

# Step 2: Update the BookService Class

# Ensure that BookService class has a setter method for BookRepository.

# Step 3: Test the Configuration

# Run the LibraryManagementApplication to verify the dependency injection.

# Implementation:

# 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  
 https://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 Class:

package com.library.repository;  
public class BookRepository {  
 public void saveBook(String bookName) {  
 System.*out*.println("BookRepository: Book saved - " + bookName);  
 }  
}

# 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 addBook(String bookName) {  
 System.*out*.println("BookService: Adding book - " + bookName);  
 bookRepository.saveBook(bookName);  
 }  
}

# MainApp Class:

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.addBook("Harry Potter and The Deathly Hollows");  
 }  
}

# Output:

# 

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

# Scenario:

You need to set up a new Maven project for the library management application and add Spring dependencies.

# Procedure:

# Step 1: Create a New Maven Project

# Create a new Maven project named LibraryManagement.

# Step 2: Add Spring Dependencies in pom.xml

# Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

# Step 3: Configure Maven Plugins

# Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

# Implementation:

<?xml version="1.0" encoding="UTF-8"?>  
<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>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>6.1.6</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>6.1.6</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>6.1.6</version>  
 </dependency>  
 </dependencies>  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.11.0</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

# Output:

# A screen shot of a computer AI-generated content may be incorrect.