**Week3\_Spring Core and Maven\_HandsOn**

**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.

**CODE :**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.*out*.println("Book '" + bookName + "' saved to the database.");

}

}

**BookService.java**

package com.library.service

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring to inject

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.*out*.println("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**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) {

// Load Spring context from XML

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

// Get BookService bean from the context

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

// Use the service

bookService.addBook("The Great Gatsby");

}

}

OUTPUT :

A screenshot of a computer

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.

**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.

**CODE :**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.*out*.println("Book '" + bookName + "' saved to the database.");

}}

**BookService.java**

package com.library.service

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring to inject

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.*out*.println("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**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 the Spring context from XML

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

// Get the BookService bean from Spring container

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

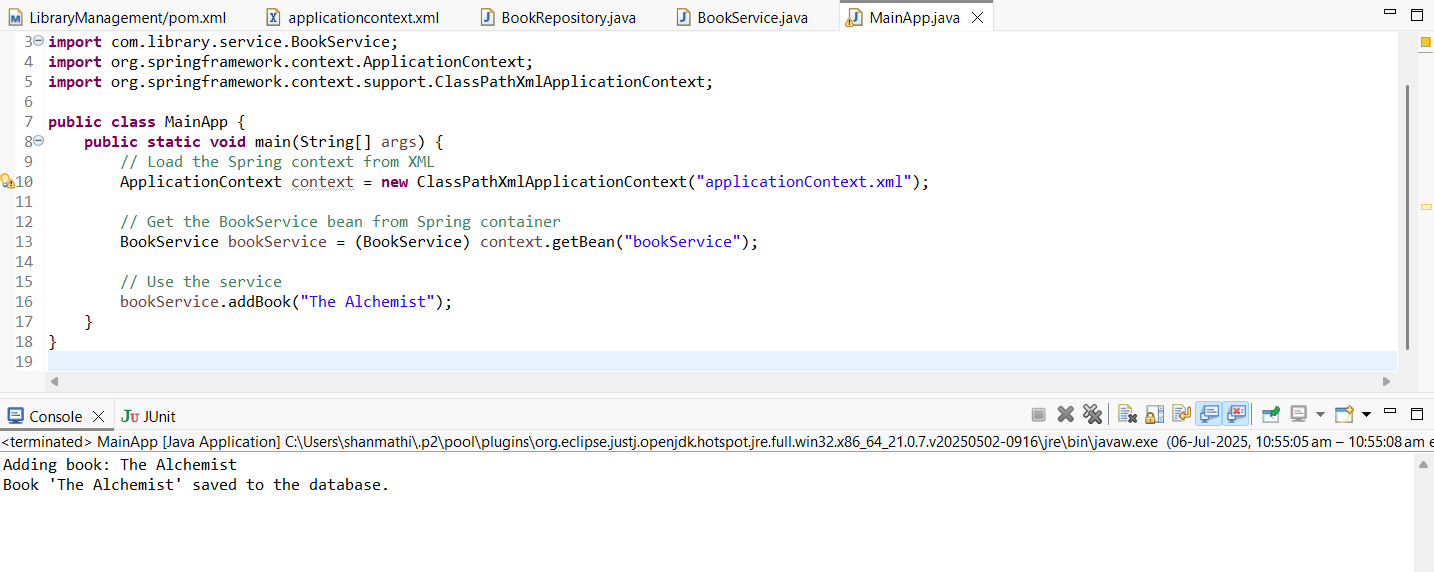
// Use the service

bookService.addBook("The Alchemist");

}

}

**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.

**CODE :**

**LibraryManagement/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>

<!-- Spring Core Container -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Servlet API (only for compilation, not deployment) -->

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<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>

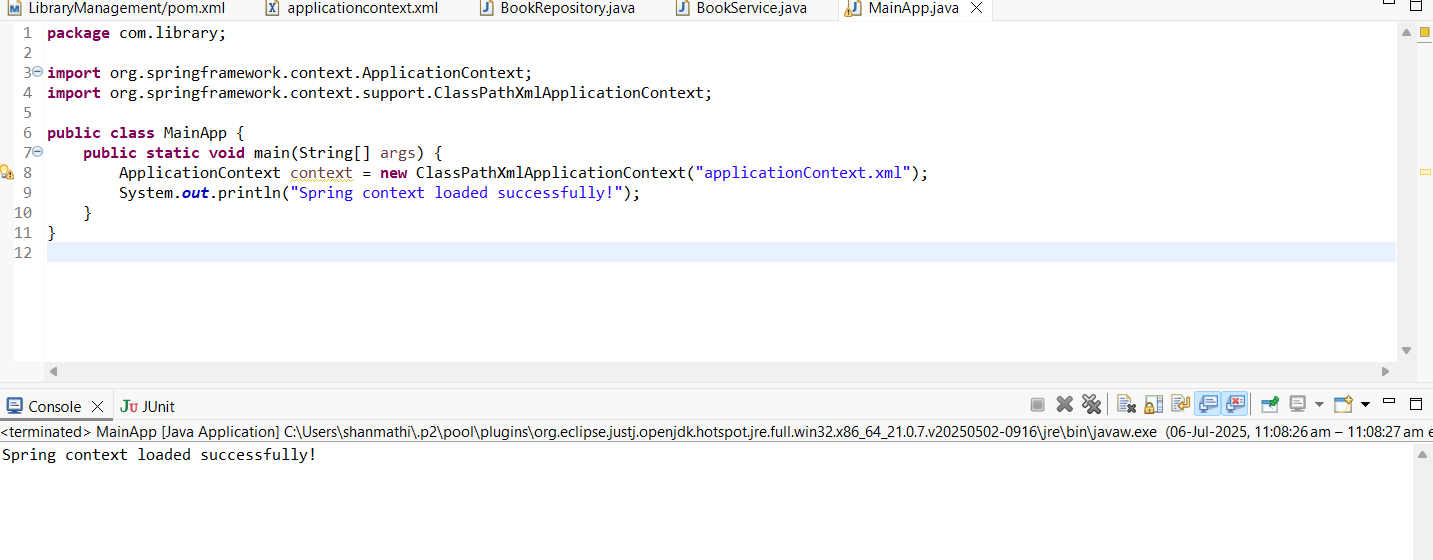
</plugin>

</plugins>

</build>

</project>

**OUTPUT :**

****

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

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**CODE :**

**Main App.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring to inject

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.*out*.println("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**Book Repository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.*out*.println("Book '" + bookName + "' saved to the database.");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring to inject

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.*out*.println("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**Output:**

## 

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

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. Configure Constructor Injection:
   * Update applicationContext.xml to configure constructor injection for BookService.
2. Configure Setter Injection:
   * Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.
3. Test the Injection:
   * Run the LibraryManagementApplication main class to verify both constructor and setter injection.

**CODE :**

**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) {

// Load Spring context

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

// Get BookService bean

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

// Test method

bookService.addBook("Effective Java");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String libraryName; // for constructor injection

// Constructor for constructor injection

public BookService(String libraryName) {

this.libraryName = libraryName;

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.*out*.println("[" + libraryName + "] Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**Book Repository.java:**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

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

}

}

**OUTPUT :**

****

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

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.xml to configure constructor injection for BookService.
2. **Configure Setter Injection:**
   * Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.
3. **Test the Injection:**
   * Run the LibraryManagementApplication main class to verify both constructor and setter injection.

**CODE :**

**MainApp Code:**

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) {

// Load Spring context from XML file

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

// Get BookService bean from context

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

// Test the injections

bookService.addBook("Clean Code");

}

}

**Book Service.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String libraryName; // Constructor injection

private BookRepository bookRepository; // Setter injection

// Constructor for libraryName

public BookService(String libraryName) {

this.libraryName = libraryName; }

// Setter for BookRepository

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository; }

public void addBook(String bookName) {

System.*out*.println("[" + libraryName + "] Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**Book Repository.java**

package com.library.repository;

public class BookRepository {

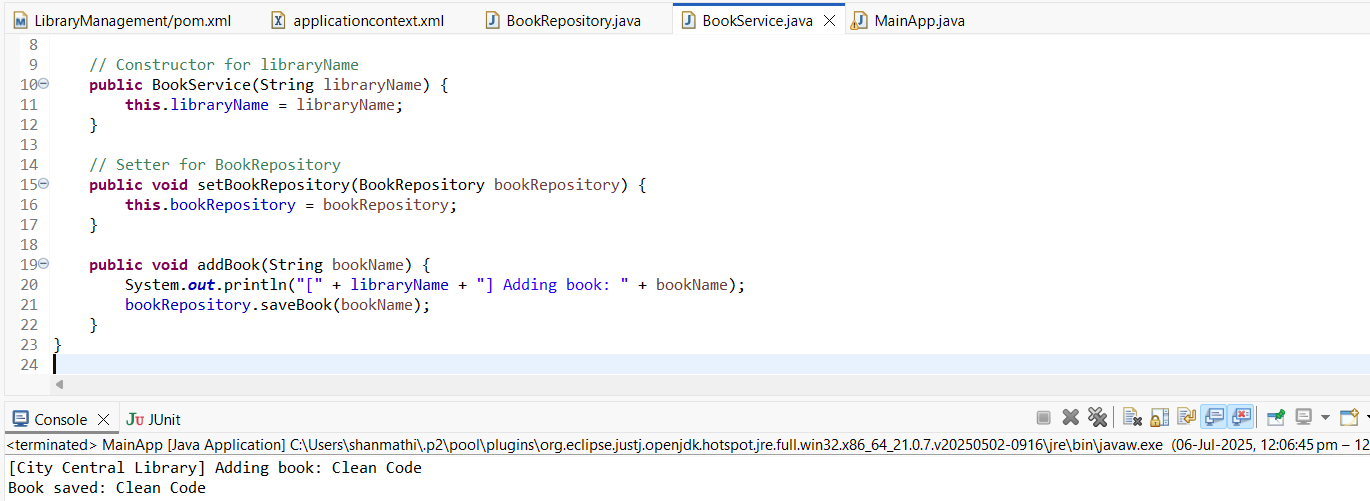
public void saveBook(String bookName) {

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

}

}

**OUTPUT :**

****