**WEEK 3 – ASSIGNMENT**

**Superset ID:** 6390124

**Spring Core and Maven Exercises:-**

**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.

**src/main/resources/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>

**src/main/java/com/library/repository/BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

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

}

}

**src/main/java/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 addBook(String title) {

System.out.println("Adding book via BookService...");

bookRepository.saveBook(title);

}

}

**src/main/java/com/library/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) {

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

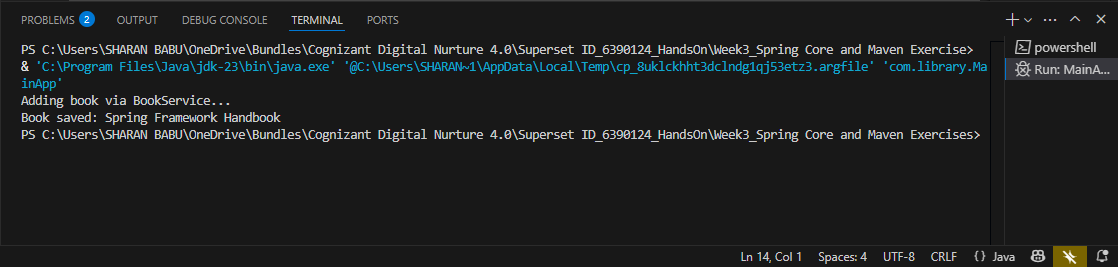
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Java Spring Fundamentals");

}

}

**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.

**src/main/resources/applicationContext1.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 for BookRepository -->

    <bean id="bookRepository" class="com.library.repository.BookRepository1"/>

    <!-- Bean for BookService (inject BookRepository using setter) -->

    <bean id="bookService" class="com.library.service.BookService1">

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

    </bean>

</beans>

**src/main/java/com/library/service/BookService1.java**

package com.library.service;

import com.library.repository.BookRepository1;

public class BookService1 {

    private BookRepository1 bookRepository;

    // Setter for Dependency Injection

    public void setBookRepository(BookRepository1 bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String title) {

        System.out.println("BookService: Adding book...");

        bookRepository.saveBook(title);

    }

}

**src/main/java/com/library/repository/BookRepository1.java**

package com.library.repository;

public class BookRepository1 {

    public void saveBook(String title) {

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

    }

}

**src/main/java/com/library/LibraryManagementApplication.java**

package com.library;

import com.library.service.BookService1;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

    public static void main(String[] args) {

        try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml")) {

            BookService1 bookService = context.getBean("bookService", BookService1.class);

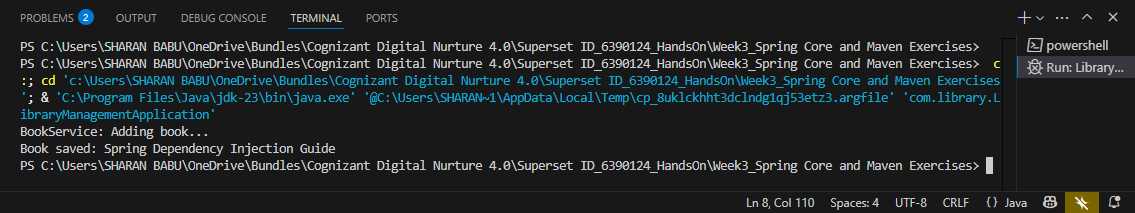
            bookService.addBook("Spring Dependency Injection Guide");

        }

    }

}

**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.

**Maven Command:**

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

**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>

<dependencies>

<!-- Spring Context (Core container and DI) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP (for Aspect-Oriented Programming) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring Web MVC (for web applications using Spring MVC) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

<build>

<plugins>

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

<plugin>

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

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<!-- Exec Plugin to run the main class -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.library.LibraryManagementApplication</mainClass>

</configuration>

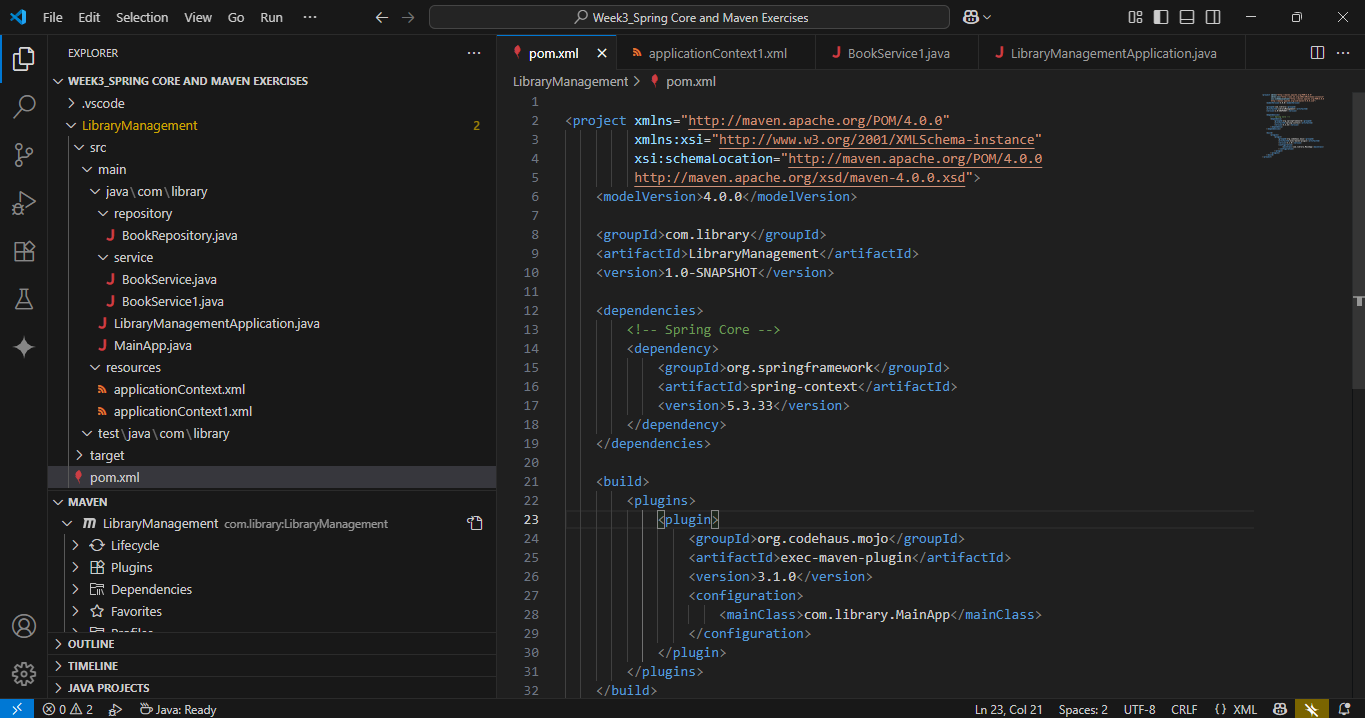
</plugin>

</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.

**src/main/resources/applicationContext2.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 for BookRepository2 -->

<bean id="bookRepository2" class="com.library.repository.BookRepository2" />

<!-- Bean for BookService2 with DI -->

<bean id="bookService2" class="com.library.service.BookService2">

<property name="bookRepository2" ref="bookRepository2"/>

</bean>

</beans>

**src/main/java/com/library/service/BookService2.java**

package com.library.service;

import com.library.repository.BookRepository2;

public class BookService2 {

private BookRepository2 bookRepository2;

// Setter for Dependency Injection

public void setBookRepository2(BookRepository2 bookRepository2) {

this.bookRepository2 = bookRepository2;

}

public void addBook(String title) {

System.out.println("BookService2: Adding book...");

bookRepository2.saveBook(title);

}

}

**src/main/java/com/library/repository/BookRepository2.java**

package com.library.repository;

public class BookRepository2 {

public void saveBook(String title) {

System.out.println("BookRepository2: Book saved - " + title);

}

}

**src/main/java/com/library/LibraryMainApp2.java**

package com.library;

import com.library.service.BookService2;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMainApp2 {

public static void main(String[] args) {

try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext2.xml")) {

BookService2 service = context.getBean("bookService2", BookService2.class);

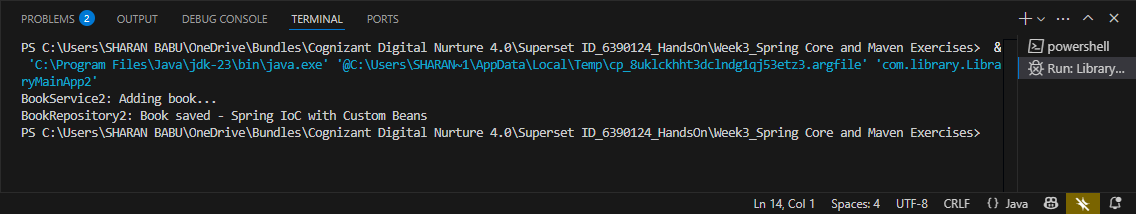
service.addBook("Spring IoC with Custom Beans");

}

}

}

**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.

**src/main/resources/applicationContext3.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">

<!-- Constructor Injection for BookService3 -->

<bean id="bookService3" class="com.library.service.BookService3">

<constructor-arg value="Central Library"/>

<property name="bookRepository3" ref="bookRepository3"/>

</bean>

<!-- BookRepository3 Bean -->

<bean id="bookRepository3" class="com.library.repository.BookRepository3"/>

</beans>

**src/main/java/com/library/service/BookService3.java**

package com.library.service;

import com.library.repository.BookRepository3;

public class BookService3 {

private String libraryName;

private BookRepository3 bookRepository3;

// Constructor injection

public BookService3(String libraryName) {

this.libraryName = libraryName;

}

// Setter injection

public void setBookRepository3(BookRepository3 bookRepository3) {

this.bookRepository3 = bookRepository3;

}

public void addBook(String title) {

System.out.println("[" + libraryName + "] BookService3: Adding book...");

bookRepository3.saveBook(title);

}

}

**src/main/java/com/library/repository/BookRepository3.java**

package com.library.repository;

public class BookRepository3 {

public void saveBook(String title) {

System.out.println("BookRepository3: Book saved - " + title);

}

}

**src/main/java/com/library/LibraryMainApp3.java**

package com.library;

import com.library.service.BookService3;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMainApp3 {

public static void main(String[] args) {

try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext3.xml")) {

BookService3 service = context.getBean("bookService3", BookService3.class);

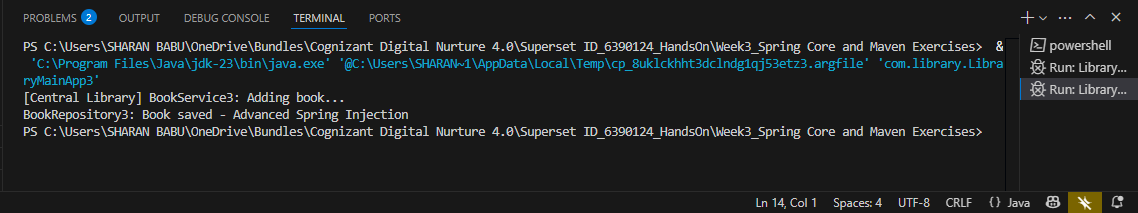
service.addBook("Advanced Spring Injection");

}

}

}

**Output:**

****

**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

**src/main/resources/application.properties**

# H2 DB Config

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA Config

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**src/main/java/com/library/entity/Book.java**

package com.library.entity;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// Getters and Setters

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;

}

}

**src/main/java/com/library/repository/BookRepository.java**

package com.library.repository;

import com.library.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**src/main/java/com/library/controller/BookController.java**

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book book) {

book.setId(id);

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**src/main/java/com/library/LibraryManagementApplication.java**

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

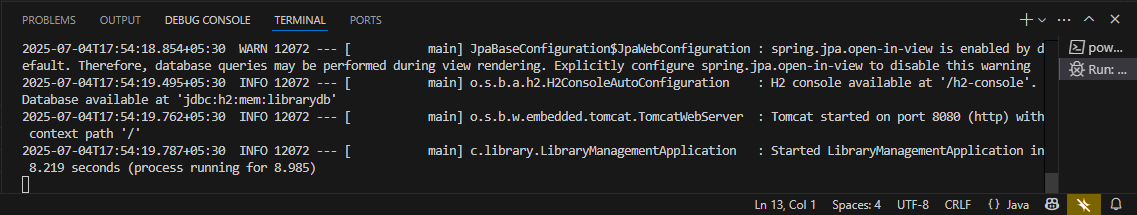
public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

**Output:**

****