**Mandatory Exercie**

**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  
<?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>demo</artifactId>

  <version>1.0-SNAPSHOT</version>

  <name>Library Management Demo</name>

  <url>http://www.example.com</url>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

    <maven.compiler.source>17</maven.compiler.source>

    <maven.compiler.target>17</maven.compiler.target>

  </properties>

  <dependencies>

    <!-- Spring Core / Context -->

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-context</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.11</version>

      <scope>test</scope>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <!-- 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.MainApp</mainClass>

        </configuration>

      </plugin>

    </plugins>

  </build>

</project>

**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) {

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

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

        service.addBook("The Alchemist");

        ((ClassPathXmlApplicationContext) context).close();

    }

}

**BookService.java**package main.java.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 name) {

        bookRepository.save(name);

    }

}

**BookRepository.java**

package main.java.com.library.repository;

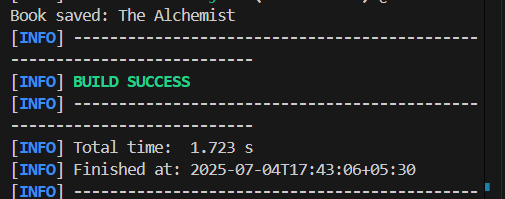
public class BookRepository {

    public void save(String bookName) {

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

    }

}

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

        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>

**BookService.java**

package main.java.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.save(bookName);

    }

}

**Main.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) {

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

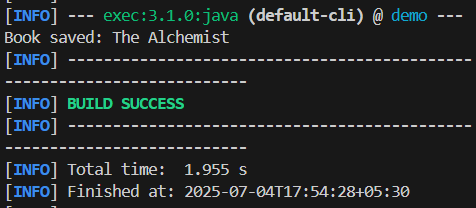
        BookService service = context.getBean("bookService", BookService.class);

        service.addBook("The Alchemist");

        ((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.  
  
Code:  
  
POM.xml  
  
<?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>

  <properties>

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  </properties>

  <dependencies>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-context</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-aop</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>org.springframework</groupId>

      <artifactId>spring-webmvc</artifactId>

      <version>5.3.30</version>

    </dependency>

    <dependency>

      <groupId>junit</groupId>

      <artifactId>junit</artifactId>

      <version>4.11</version>

      <scope>test</scope>

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <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>

    </plugins>

  </build>

</project>

**BookServices.java**

package main.java.com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

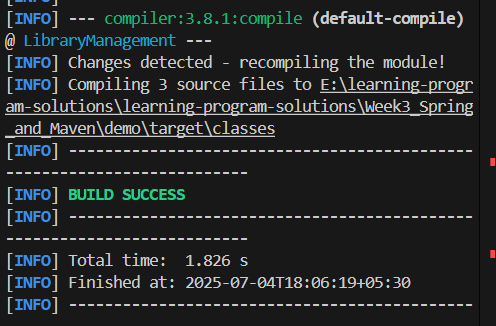
    public void addBook(String name) {

        bookRepository.save(name);

    }

}

Output:



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

**Scenario:**

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

**Code:  
  
BookRepository.java**package com.library.repository;

public class BookRepository {

    public void displayRepository() {

        System.out.println("BookRepository is working!");

    }

}

**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 displayService() {

        System.out.println("BookService is working!");

        bookRepository.displayRepository();

    }

}

**MainApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

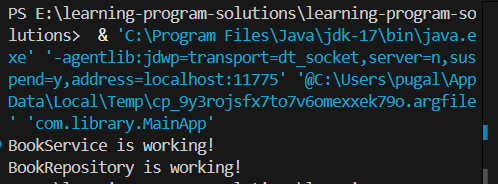
public static void main(String[] args) {

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

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

bookService.displayService();

}

}  
  
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.

**Code:  
BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String libraryName;

private int libraryId;

private BookRepository bookRepository;

public BookService(String libraryName, int libraryId) {

this.libraryName = libraryName;

this.libraryId = libraryId;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayService() {

System.out.println("Library Name: " + libraryName);

System.out.println("Library ID: " + libraryId);

bookRepository.displayRepository();

}

}

applicationContent.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">

<!-- Constructor Injection -->

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

<constructor-arg value="101" />

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

</bean>

</beans>

**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");

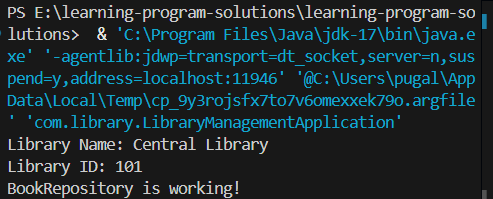
BookService service = (BookService) context.getBean("bookService");

service.displayService();

}

}

Output



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

