# 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.

## 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>  
 <packaging>jar</packaging>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <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 bookService = context.getBean("bookService", BookService.class);  
 bookService.display();  
 }  
}

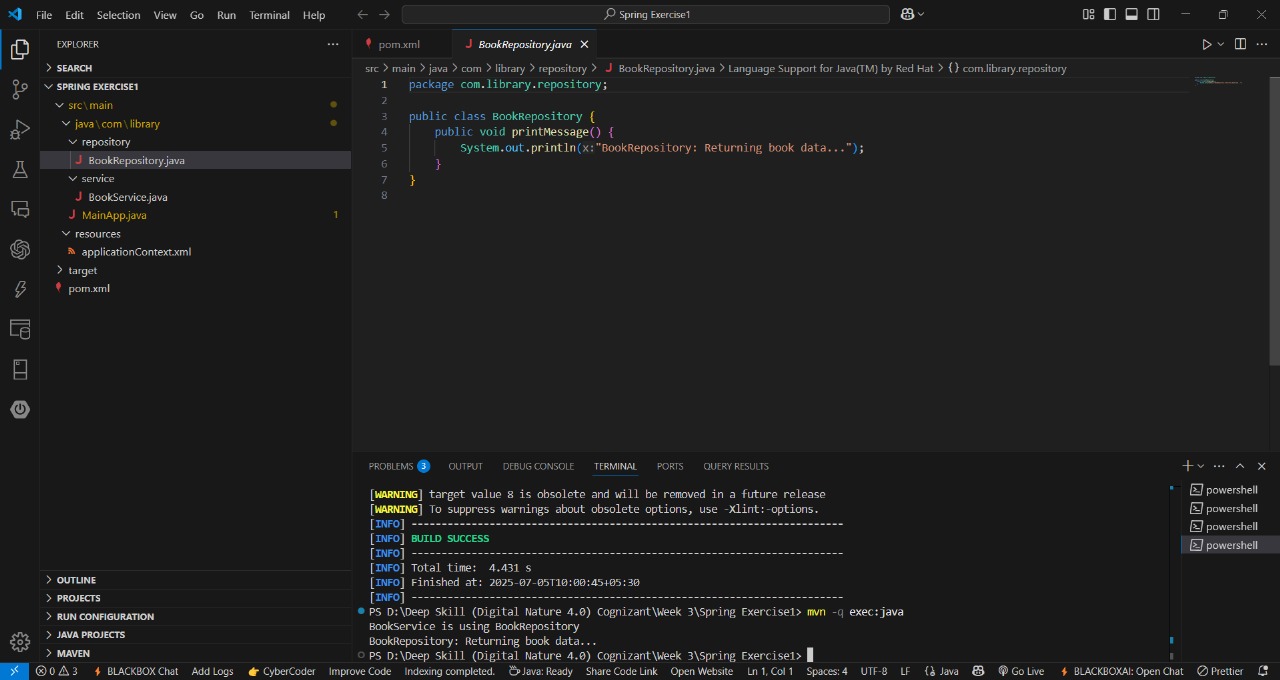
## BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public void printMessage() {  
 System.out.println("BookRepository: Returning book data...");  
 }  
}

## 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 display() {  
 System.out.println("BookService is using BookRepository");  
 bookRepository.printMessage();  
 }  
}

## 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.

## 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.example</groupId>  
 <artifactId>LibraryManagementApp</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <dependencies>  
 <!-- Spring Context for Dependency Injection -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.32</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Plugin to run Java main class -->  
 <plugin>  
 <groupId>org.codehaus.mojo</groupId>  
 <artifactId>exec-maven-plugin</artifactId>  
 <version>3.1.0</version>  
 <configuration>  
 <mainClass>com.example.library.LibraryManagementApplication</mainClass>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

## BookRepository.java

package com.example.library;  
  
public class BookRepository {  
 public String getBookDetails() {  
 return "Effective Java by Joshua Bloch";  
 }  
}

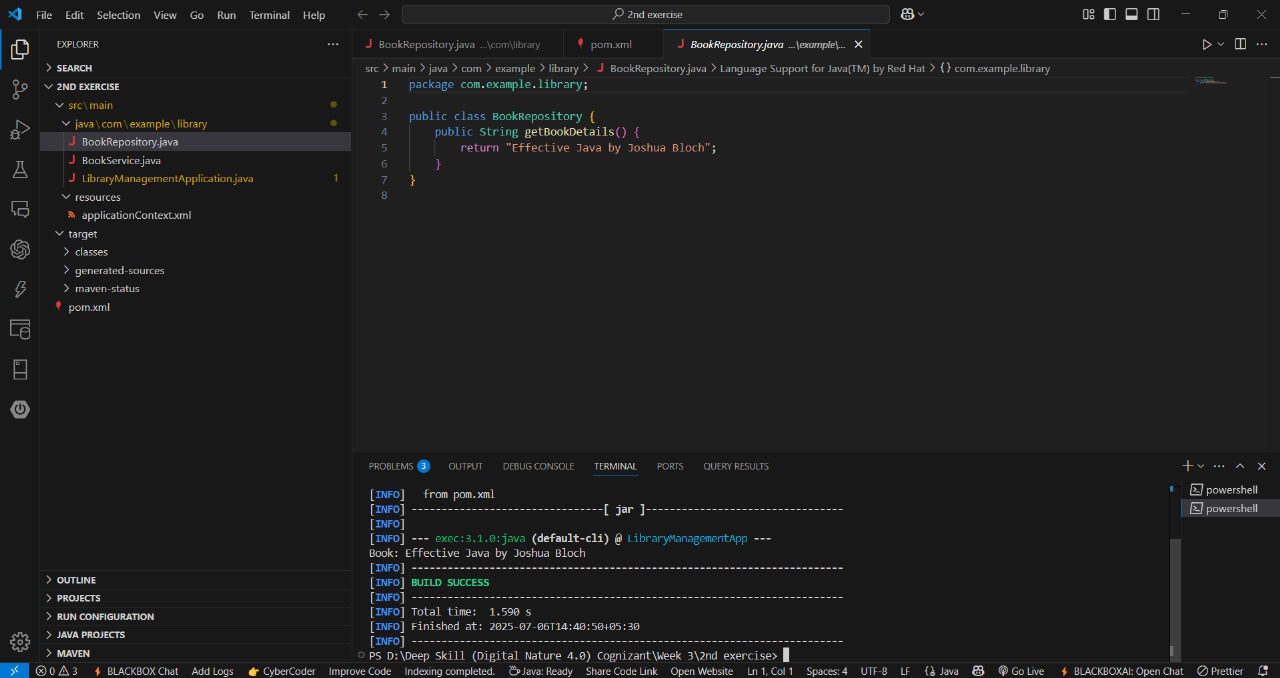
## BookService.java

package com.example.library;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBook() {  
 System.out.println("Book: " + bookRepository.getBookDetails());  
 }  
}

## LibraryManagementApplication.java

package com.example.library;  
  
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 = (BookService) context.getBean("bookService");  
 bookService.displayBook();  
 }  
}

## 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.

## 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>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.22</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Compiler Plugin -->  
 <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 Java class -->  
 <plugin>  
 <groupId>org.codehaus.mojo</groupId>  
 <artifactId>exec-maven-plugin</artifactId>  
 <version>3.1.0</version>  
 <configuration>  
 <mainClass>com.library.LibraryApp</mainClass>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
</project>

## LibraryApp.java

package com.library;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryApp {  
 public static void main(String[] args) {  
 System.out.println("Library Management Application Started");  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 System.out.println("Spring Context Initialized");  
 }  
}

## Output

