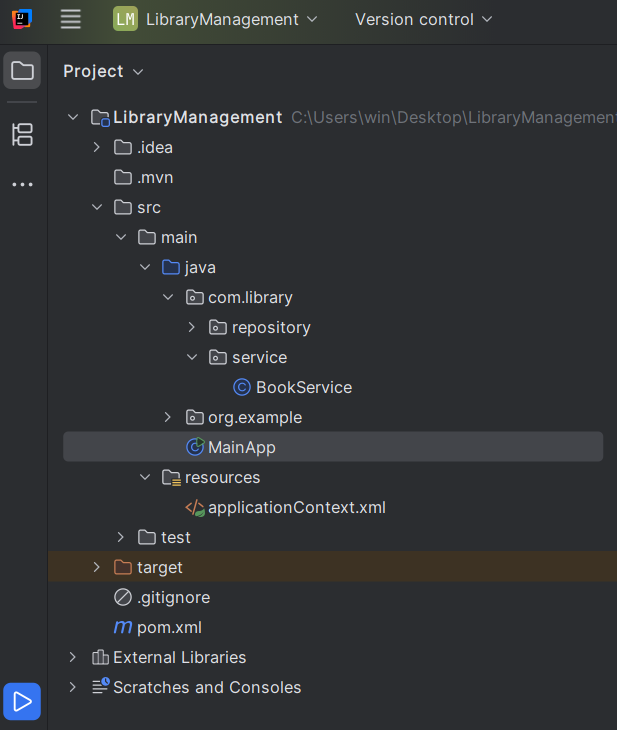
**MANDATORY HANDS ON**

**Exercise 1: Configuring a Basic Spring Application**

File Structure:



Step 1: Set Up a Spring Project

**Name the project**: LibraryManagement

**Add Spring Core Dependency in pom.xml**

Code:

<?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>org.example</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.31</version>  
 </dependency>  
 </dependencies>  
  
</project>

### **Step 2: Configure** applicationContext.xml

1. **Right-click** src/main/resources **→ New → File** → Name it applicationContext.xml
2. Paste this Spring configuration:

<?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">  
  
 <!-- Define BookRepository bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
  
 <!-- Define BookService bean -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

Step 3: Define Service and Repository Classes:

#### Create Package: com.library.repository

1. Right-click src/main/java → New → Package → com.library.repository
2. Create a class: BookRepository.java

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

#### Create Package: com.library.service

1. Right-click src/main/java → New → Package → com.library.service
2. Create a class: BookService.java

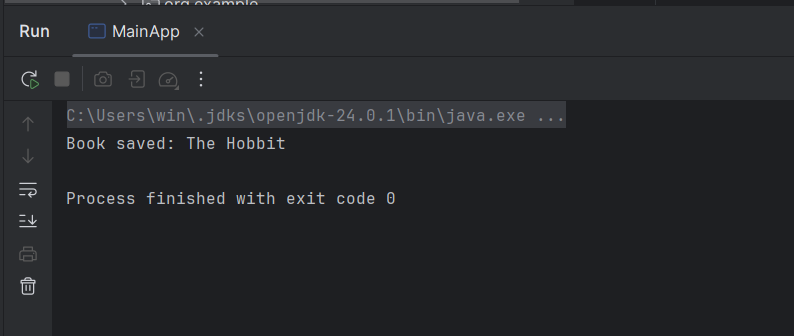
package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // Setter for dependency injection  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String bookName) {  
 bookRepository.saveBook(bookName);  
 }  
}

Step 4: Run the Application

Right-click src/main/java → New → Java Class → Name it MainApp

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("The Hobbit");  
 }  
}

Output:



**Exercise 2: Implementing Dependency Injection**

Step 1: Update applicationContext.xml

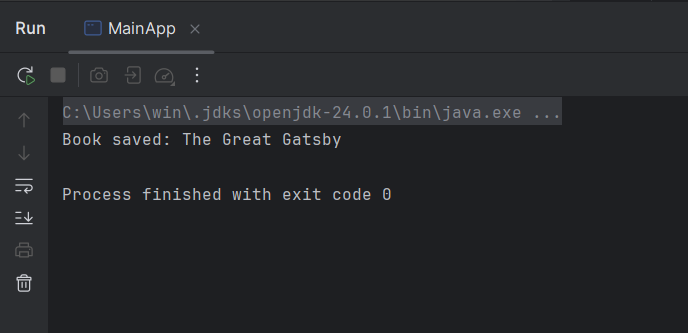
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <!-- Define BookRepository bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <!-- Define BookService bean and inject BookRepository -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

Step 2: Make Sure BookService Has a Setter Method

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
  
 private BookRepository bookRepository;  
  
 // Setter method for Spring to inject dependency  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String name) {  
 bookRepository.saveBook(name);  
 }  
}

Step 3: Test It in Your Main Class

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("The Great Gatsby");  
 }  
}

Output:  


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

### **Step 1: Create a New Maven Project**

In **IntelliJ IDEA**:

1. **File → New → Project**
2. Choose **Maven**
3. Name it:
   * **Project Name:** LibraryManagement
4. Click **Finish**

IntelliJ will generate a basic structure with a pom.xml file.

Step 2: Add Spring Dependencies in 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>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core / Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.31</version>  
 </dependency>  
  
 <!-- Spring AOP -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.31</version>  
 </dependency>  
  
 <!-- Spring Web MVC -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.31</version>  
 </dependency>  
  
 <!-- Servlet API (provided by server like Tomcat) -->  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>javax.servlet-api</artifactId>  
 <version>4.0.1</version>  
 <scope>provided</scope>  
 </dependency>  
 </dependencies>  
  
 <!-- Step 3: Maven Compiler Plugin for Java 1.8 -->  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.10.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

After updating the pom.xml:

1. IntelliJ will prompt to Reload Maven Project → click "Reload"
2. You can now:
   * Build: mvn compile
   * Run tests: mvn test
   * Run app: mvn exec:java (with plugin)

**ADDITIONAL HANDS ON:**

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

Step 1: Create Spring Configuration File

#### File: src/main/resources/applicationContext.xml

1. Right-click src/main/resources → New → File → Name it applicationContext.xml
2. Paste the following Spring bean configuration:

<?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">  
  
 <!-- BookRepository Bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <!-- BookService Bean -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <!-- Dependency Injection using setter -->  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

### Step 2: Update BookService to Have a Setter Method

File: src/main/java/com/library/service/BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
  
 private BookRepository bookRepository;  
  
 // Required for setter-based injection  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String name) {  
 bookRepository.saveBook(name);  
 }  
}

File: src/main/java/com/library/repository/BookRepository.java

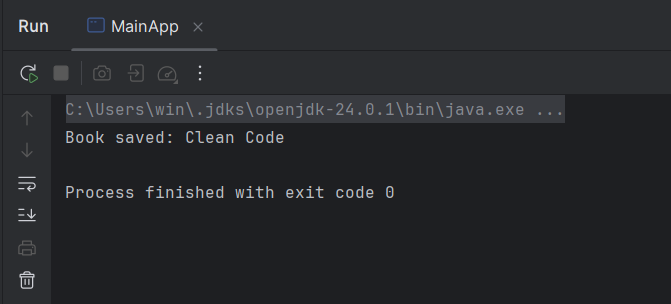
package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String name) {  
 System.*out*.println("Book saved: " + name);  
 }  
}

### Step 3: Create Main Class to Load Spring Context and Test

File: src/main/java/com/library/MainApp.java

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 Spring context  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 // Get bean and use service  
 BookService bookService = context.getBean("bookService", BookService.class);  
 bookService.addBook("Clean Code");  
 }  
}

Output:



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

Step 1: Update BookService Class

File: src/main/java/com/library/service/BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
  
 private BookRepository bookRepository;  
 private String libraryName; // example for setter injection  
  
 // Constructor Injection  
 public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 // Setter Injection  
 public void setLibraryName(String libraryName) {  
 this.libraryName = libraryName;  
 }  
  
 public void addBook(String name) {  
 System.*out*.println("Library: " + libraryName);  
 bookRepository.saveBook(name);  
 }  
}

## Step 2: Update applicationContext.xml for Both Injections

## File: src/main/resources/applicationContext.xml

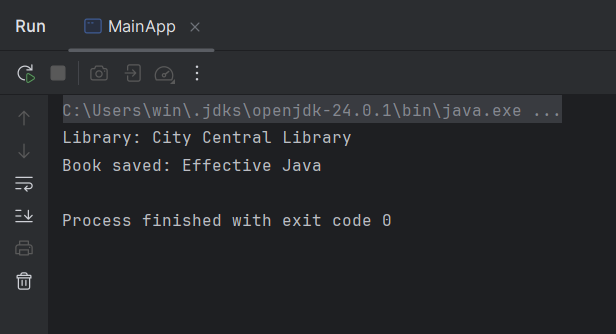
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <!-- BookRepository Bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <!-- BookService Bean with constructor + setter injection -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <!-- Constructor Injection -->  
 <constructor-arg ref="bookRepository" />  
  
 <!-- Setter Injection -->  
 <property name="libraryName" value="City Central Library" />  
 </bean>  
  
</beans>

## Step 3: Main Class to Test Injection

File: src/main/java/com/library/MainApp.java

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("Effective Java");  
 }  
}

Output:



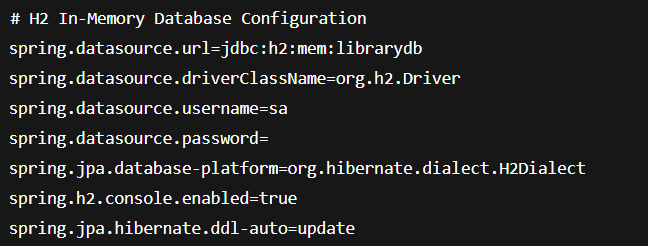
**Exercise 9: Creating a Spring Boot Application**

Step 1: Create Spring Boot Project

* Project: Maven
* Group: com.library
* Name: LibraryManagement

### Step 2: **application.properties**

File: src/main/resources/application.properties



### **Step 3: Create the Entity**

File: 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 & 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; }  
}

### **Step 4: Create the Repository**

File: 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> {  
}

### **Step 5: Create the REST Controller**

File: 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.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/books")  
public class BookController {  
  
 private final BookRepository repository;  
  
 public BookController(BookRepository repository) {  
 this.repository = repository;  
 }  
  
 @GetMapping  
 public List<Book> getAllBooks() {  
 return repository.findAll();  
 }  
  
 @PostMapping  
 public Book addBook(@RequestBody Book book) {  
 return repository.save(book);  
 }  
  
 @GetMapping("/{id}")  
 public Book getBook(@PathVariable Long id) {  
 return repository.findById(id).orElse(null);  
 }  
  
 @PutMapping("/{id}")  
 public Book updateBook(@PathVariable Long id, @RequestBody Book updated) {  
 Book book = repository.findById(id).orElse(null);  
 if (book != null) {  
 book.setTitle(updated.getTitle());  
 book.setAuthor(updated.getAuthor());  
 return repository.save(book);  
 }  
 return null;  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable Long id) {  
 repository.deleteById(id);  
 }  
}

### **Step 6: Run the Application**

File: 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);  
 }  
}

### Test It

Run the app, then test these endpoints using **Postman** or **curl**:

* GET http://localhost:8080/books
* POST http://localhost:8080/books  
  Body (JSON):

