**Spring Data JPA with Spring Boot, Hibernate**

# 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:

* Set Up a Spring Project: Create a Maven project named LibraryManagement. Add Spring Core dependencies in the pom.xml file.
* Configure the Application Context: Create an XML configuration file named applicationContext.xml. Define beans for BookService and BookRepository.
* Define Service and Repository Classes: Create BookService and BookRepository under appropriate packages.
* Run the Application: Create a main class to load the Spring context and test the configuration.

## pom.xml

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.20</version>  
 </dependency>  
</dependencies>

## applicationContext.xml

<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>

## BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String title) {  
 System.out.println("Saving book: " + title);  
 }  
}

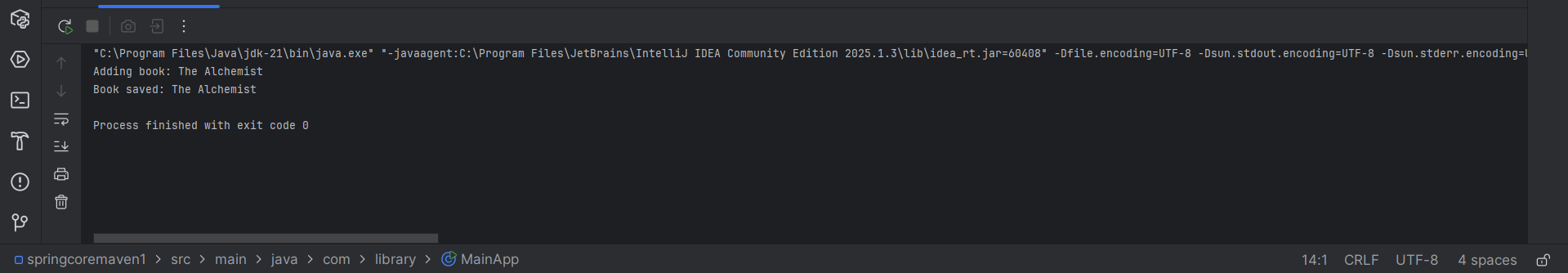
## BookService.java

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) {  
 bookRepository.saveBook(title);  
 }  
}

## LibraryManagementApplication.java

import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import com.library.service.BookService;  
  
public class LibraryManagementApplication {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService service = context.getBean("bookService", BookService.class);  
 service.addBook("Spring in Action");  
 }  
}

**Output:**

****

# Exercise 2: Implementing Dependency Injection

Scenario:  
You need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

Steps:

* Modify the XML Configuration to wire BookRepository into BookService.
* Ensure that BookService class has a setter method for BookRepository.
* Run the LibraryManagementApplication main class to verify the dependency injection.

## applicationContext.xml

<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"> <!-- Repository Bean --> <bean id="bookRepository" class="com.library.repository.BookRepository" /> <!-- Service Bean with Setter Injection --> <bean id="bookService" class="com.library.service.BookService"> <property name="bookRepository" ref="bookRepository" /> </bean></beans>

## BookRepository.java

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

## 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 title) { bookRepository.saveBook(title); }}

## LibraryManagementApplication.java

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

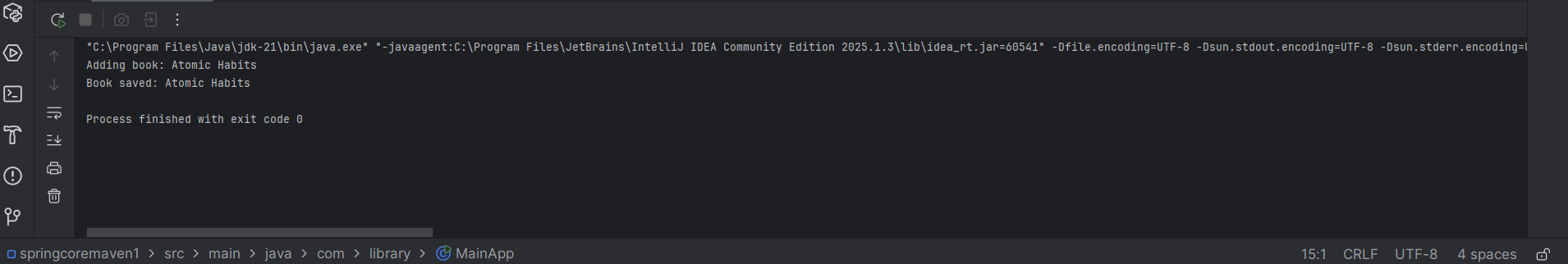
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Effective Java");

}

}

**Output:**

****

# Exercise 3: Implementing Logging with Spring AOP

Scenario:  
The application requires logging capabilities to track method execution times.

Steps:

* Add Spring AOP Dependency in pom.xml.
* Create LoggingAspect class in com.library.aspect.
* Enable AspectJ Support in XML.
* Run the main class and observe logs.

## pom.xml (AOP)

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aspects</artifactId>  
 <version>5.3.20</version>  
</dependency>

## LoggingAspect.java

package com.library.aspect;  
  
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Aspect;  
  
@Aspect  
public class LoggingAspect {  
 @Around("execution(\* com.library.service.\*.\*(..))")  
 public Object log(ProceedingJoinPoint pjp) throws Throwable {  
 long start = System.currentTimeMillis();  
 Object result = pjp.proceed();  
 long end = System.currentTimeMillis();  
 System.out.println(pjp.getSignature() + " executed in " + (end - start) + "ms");  
 return result;  
 }  
}

## applicationContext.xml AOP

<aop:aspectj-autoproxy/>  
<bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>

# Exercise 4: Creating and Configuring a Maven Project

Scenario:  
You need to set up a new Maven project for the application and add dependencies.

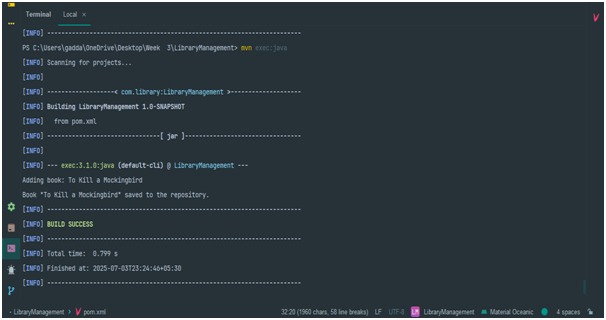
Steps:

* Create a new Maven project named LibraryManagement.
* Add Spring Context, AOP, and WebMVC dependencies.
* Configure Maven Compiler Plugin for Java 1.8.

## pom.xml (compiler plugin)

<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>

**Output:**



# Exercise 5: Configuring the Spring IoC Container

Scenario:  
The application requires central configuration for beans and dependencies.

Steps:

* Create applicationContext.xml.
* Define BookService and BookRepository as beans.
* Create a main class to test the configuration.

## applicationContext.xml

<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 Definitions -->

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

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

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

</bean>

</beans>

## BookRepository.java

package com.library.repository;

public class BookRepository {

public void save(String title) {

System.out.println("Saved book: " + title);

}

}

## BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void registerBook(String title) {

bookRepository.save(title);

}

}

## LibraryManagementApplication.java

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

bookService.registerBook("Spring Framework Essentials");

}

}

# Exercise 6: Configuring Beans with Annotations

Scenario:  
You need to simplify the configuration using annotations.

Steps:

* Update applicationContext.xml to include component scanning.
* Use @Service and @Repository annotations.
* Test the annotation-based configuration.

## applicationContext.xml

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

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

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<!-- Enable annotation-based configuration -->

<context:component-scan base-package="com.library" />

</beans>

## BookRepository.java

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void save(String title) {

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

}

}

## BookService.java

package com.library.service;

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

import org.springframework.stereotype.Service;

import com.library.repository.BookRepository;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public void registerBook(String title) {

bookRepository.save(title);

}

}

## LibraryManagementApplication.java

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

bookService.registerBook("Spring Annotations Simplified");

}

}

# Exercise 7: Implementing Constructor and Setter Injection

Scenario:  
Constructor and setter injection are needed for better control.

Steps:

* Update applicationContext.xml to configure constructor injection.
* Ensure setter method exists and is wired via XML.
* Test both injection types.

## applicationContext.xml

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

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

<constructor-arg ref="bookRepository" />

</bean>

## BookService.java (Constructor Injection)

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Constructor Injection

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void registerBook(String title) {

bookRepository.save(title);

}

}

## BookRepository.java

package com.library.repository;

public class BookRepository {

public void save(String title) {

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

}

}

# Exercise 8: Implementing Basic AOP with Spring

Scenario:  
Implement basic AOP to separate cross-cutting concerns.

Steps:

* Define LoggingAspect in com.library.aspect.
* Define advice methods before/after method execution.
* Register aspect and enable auto-proxying.
* Test AOP functionality.

## LoggingAspect.java

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.\*;

@Aspect

public class LoggingAspect {

@Before("execution(\* com.library.service.\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("Before executing: " + joinPoint.getSignature().getName());

}

@After("execution(\* com.library.service.\*.\*(..))")

public void logAfter(JoinPoint joinPoint) {

System.out.println("After executing: " + joinPoint.getSignature().getName());

}

}

## applicationContext.xml

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

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

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<context:component-scan base-package="com.library" />

<aop:aspectj-autoproxy />

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

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

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

</bean>

<!-- Register the aspect -->

<bean id="loggingAspect" class="com.library.aspect.LoggingAspect" />

</beans>

## 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 registerBook(String title) {

System.out.println("Inside registerBook()");

bookRepository.save(title);

}

}

## BookRepository.java

package com.library.repository;

public class BookRepository {

public void save(String title) {

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

}

}

## LibraryManagementApplication.java

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

service.registerBook("AOP in Spring");

}

}

# 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:

* Use Spring Initializr to create a new Spring Boot project named **LibraryManagement**.  
  • Include dependencies for **Spring Web**, **Spring Data JPA**, and **H2 Database**.  
  • Configure database connection properties in **application.properties**.  
  • Create **Book** entity and **BookRepository** interface.  
  • Create **BookController** class to handle CRUD operations.  
  • Run the Spring Boot application and test the REST endpoints.

## Book.java

package com.librarymanagement.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@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;

}

}

## BookRepository.java

package com.librarymanagement.repository;

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

import com.librarymanagement.entity.Book;

public interface BookRepository extends JpaRepository<Book, Long> {

}

## BookComtroller.java

package com.librarymanagement.controller;

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

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

import com.librarymanagement.entity.Book;

import com.librarymanagement.repository.BookRepository;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

return bookRepository.findById(id).map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

return bookRepository.save(book);

}).orElse(null);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

## LibraryManagementApplication.java

package com.librarymanagement;

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);

}

}