**Additional important hands-on**

**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.

1. **Update the BookService Class:**

* Ensure that the **BookService** class has a setter method for **BookRepository**.

1. **Run the Application:**

* Create a main class to load the Spring context and test the configuration.

**Steps:**

* + 1. **Create spring configuration file:**

Create an XML configuration file named applicatonContext.xml in the src/main/resources directory.

After creating applicationContext.xml we need to write versions, beans and classes and defining beans for Bookservce and Bookrepository in the xml file.

**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">

<!-- Define BookRepository Bean -->

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

<!-- Define BookService Bean with Dependency Injection -->

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

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

</bean>

</beans>

* + 1. **Update the Bookservice class:**

Ensuring that Bookservice class has a setter method for Bookrepository.

**Bookrepository class:**

package com.library.repository;

public class Bookrepository {

public String findBookById(int id) {

return "Book with ID: " + id;

}

}

**Bookservice class:**

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 displayBook(int id) {

String result = bookRepository.findBookById(id);

System.out.println(result);

}

}

* + 1. **Run the application:**

Creating a main class to load the spring context and for testing the configuration.

**Main class:**

package com.library;

import com.library.service.Bookservice;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Main {

public static void main(String[] args) {

// Load the Spring IoC container using the XML configuration

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

// Get the BookService bean from the context

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

// Call a method to verify that DI is working

bookService.displayBook(13);

}

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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**.

1. **Configure Setter Injection:**

* Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in applicationContext.xml.

1. **Test the Injection:**

* Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Steps:**

* + 1. **Configure constructor injection:**

Update applicationContext.xml to configure constructor injection for Bookservice.

**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">

<!-- 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="bookRepository" ref="bookRepository"/>

</bean>

</beans>

* + 1. **Configure setter injection:**

Ensure that the Bookservice class has a setter method for Bookrepository and configure it in applicationContext.xml.

**Bookservice class:**

package com.library.service;

import com.library.repository.Bookrepository;

public class Bookservice {

private Bookrepository bookRepository;

//Constructor for constructor injection

public Bookservice(Bookrepository bookRepository) {

System.out.println("Constructor Injection Called");

this.bookRepository = bookRepository;

}

//Setter for setter injection

public void setBookRepository(Bookrepository bookRepository) {

System.out.println("Setter Injection Called");

this.bookRepository = bookRepository;

}

public void displayBook(int id) {

String result = bookRepository.findBookById(id);

System.out.println(result);

}

}

**Bookrepository class:**

package com.library.repository;

public class Bookrepository {

public String findBookById(int id) {

return "Book with ID: " + id;

}

}

* + 1. **Test the injection:**

Run the LibraryManagementApplication main class to verify both constructor and setter injection.

**LibraryManagementApplication class:**

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

bookService.displayBook(42);

}

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**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**.

1. **Add Dependencies:**

* Include dependencies for **Spring** **Web**, **Spring** **Data** **JPA**, and **H2** **Database**.

1. **Create Application Properties:**

* Configure database connection properties in **application.properties**.

1. **Define Entities and Repositories:**

* Create **Book** entity and **BookRepository** interface.

1. **Create a REST Controller:**

* Create a **BookController** class to handle CRUD operations.

1. **Run the Application:**

* Run the Spring Boot application and test the REST endpoints.

**Steps:**

* + 1. **Create a spring boot project:**

Using spring initializr to create a new spring boot project named LibraryManagement:

Go to <https://start.spring.io> for spring initializr

**Configure:**

* Project: **Maven**
* Language: **Java**
* Group: com.library
* Artifact: LibraryManagement
* Name: LibraryManagement
* Packaging: Jar
* Java: 17

And create one spring boot project named LibraryManagement.

* + 1. **Add dependencies:**

Include the below dependencies in Spring Boot:

* Spring Web
* Spring Data JPA
* H2 Database

Click Generate to download the project ZIP and import it into Eclipse.

* + 1. **Create application properties:**

**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

# H2 Console

spring.h2.console.enabled=true

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

* + 1. **Define entities and repositories:**

Create Book entity and BookRepository interface:

**Book.java:**

package com.library.model.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.library.repository;

import com.library.model.entity.Book;

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

import org.springframework.stereotype.Repository;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

* + 1. **Create a REST controller:**

Create a BookController class to handle CRUD operations:

**BookController.java:**

package com.library.controller;

import com.library.model.entity.\*;

import com.library.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

* + 1. **Run the application:**

Run the Spring Boot application and test the REST endpoints:

**LibrarymanagementApplication:**

package com.library.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication(scanBasePackages = "com.library")

@EnableJpaRepositories(basePackages = "com.library.repository")

@EntityScan(basePackages = "com.library.model.entity")

public class LibrarymanagementApplication {

public static void main(String[] args) {

SpringApplication.*run*(LibrarymanagementApplication.class, args);

}

}

Run the application with java application and the spring boot has successfully running on the port 8080.

Open the postman/browers to test whether this are working or not this is called api testing

| **Method** | **Endpoint** | **Purpose** |
| --- | --- | --- |
| GET | http://localhost:8080/books | Get all books |
| POST | http://localhost:8080/books | Create new book |
| GET | http://localhost:8080/books/{id} | Get book by ID |
| PUT | http://localhost:8080/books/{id} | Update a book |
| DELETE | http://localhost:8080/books/{id} | Delete a book |
| GET | http://localhost:8080/h2-console | Access H2 console |

Given those are methods to test Get is for get all books, Post is for creating new book, another Get is for creating book by ID, Put is for updating a book, Delete is for delete a book, another Get is for Access H2 Database.

**Output:**

I have tested through postman, by all these methods.

<http://localhost:8080/h2-console> open this url in the browser.

spring.h2.console.enabled=true

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

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

click connect.

It will open h2 database to check.

Below is the image,

A screenshot of a computer

AI-generated content may be incorrect.