

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.

In Exercise 1, I set up a basic Spring application using Maven in Eclipse by creating the project structure, adding Spring Core dependency, defining beans (BookService and BookRepository) in applicationContext.xml, and successfully loading the Spring container through a main class.

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.

CODE :

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 = context.getBean("bookService", BookService.class);
        bookService.addBook("Spring in Action");
    }
}
```

BookRepository.java

```
package com.library.repository;

public class BookRepository {

    public void saveBook(String bookName) {

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

    }

}
```

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 bookName) {

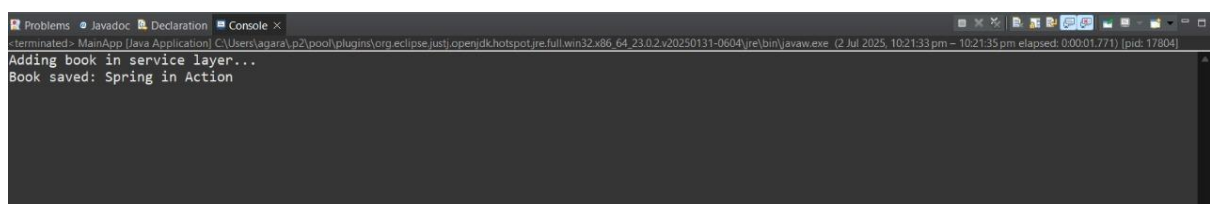
        System.out.println("Adding book in service layer...");

        bookRepository.saveBook(bookName);

    }

}
```

OUTPUT :



```
terminated> MainApp [Java Application] C:\Users\agara.p2\pool\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe (2 Jul 2025, 10:21:33 pm - 10:21:35 pm elapsed: 0:00:01.771) [pid: 17804]
Adding book in service layer...
Book saved: Spring in Action
```

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.

In Exercise 4, I created a complete Maven-based Spring project in Eclipse by configuring Spring Core, AOP, and WebMVC dependencies along with the Maven Compiler Plugin for Java 1.8 compatibility, ensuring a properly structured and build-ready application environment.

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.

2. Update the BookService Class:

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

3. Run the Application:

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

CODE:

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) {
        System.out.println("Adding book in service layer...");
        bookRepository.saveBook(bookName);
    }
}
```

MainApp.java

```
package com.library;

import com.library.service.BookService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

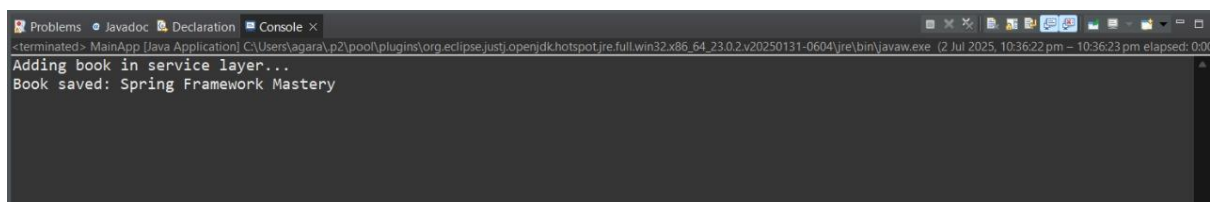
    public static void main(String[] args) {

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

        BookService bookService = context.getBean("bookService", BookService.class);
        bookService.addBook("Spring Framework Mastery");

    }
}
```

OUTPUT :

A screenshot of an IDE's console window. The window has a title bar with tabs for 'Problems', 'Javadoc', 'Declaration', and 'Console'. The 'Console' tab is active. The console output shows the following text: '<terminated> MainApp [Java Application] C:\Users\agara\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe (2 Jul 2025, 10:36:22 pm - 10:36:23 pm elapsed: 0:00)'. Below this, the application's output is displayed: 'Adding book in service layer...' followed by 'Book saved: Spring Framework Mastery' on the next line. The console background is dark gray, and the text is white.

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**.

2. Configure Setter Injection:

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

3. Test the Injection:

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

CODE:

BookService.java

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;
    private String libraryName;

    // Constructor for constructor injection
    public BookService(String libraryName) {
        this.libraryName = libraryName;
        System.out.println("Constructor Injection: Library Name = " + libraryName);
    }

    // Setter for BookRepository (Setter Injection)
    public void setBookRepository(BookRepository bookRepository) {
```

```

        this.bookRepository = bookRepository;
    }

    public void addBook(String bookName) {
        System.out.println "[" + libraryName + "] Adding book in service layer...");
        bookRepository.saveBook(bookName);
    }
}

```

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");

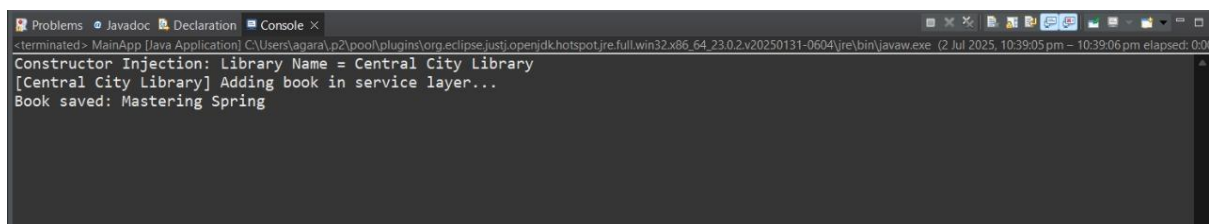
        BookService bookService = context.getBean("bookService", BookService.class);

        bookService.addBook("Mastering Spring");

    }
}

```

OUTPUT:



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**.
2. **Add Dependencies:**
 - Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
 - Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
 - Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
 - Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
 - Run the Spring Boot application and test the REST endpoints.

CODE :

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);
    }
}
```

```
    }  
}
```

BookController.java

```
package com.library.controller;  
import com.library.entity.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.*;
```

```
import java.util.List;  
import java.util.Optional;
```

```
@RestController
```

```
@RequestMapping("/books")
```

```
public class BookController {
```

```
    @Autowired
```

```
    private BookRepository bookRepository;
```

```
    // Get all books
```

```
    @GetMapping
```

```
    public List<Book> getAllBooks() {
```

```
        return bookRepository.findAll();
```

```
    }
```

```
    // Get book by ID
```

```
    @GetMapping("/{id}")
```

```
    public Book getBookById(@PathVariable Long id) {
```

```
        Optional<Book> book = bookRepository.findById(id);
```

```

        return book.orElse(null);
    }

    // Add a new book
    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }

    // Update an existing book
    @PutMapping("/{id}")
    public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {
        Optional<Book> optionalBook = bookRepository.findById(id);
        if (optionalBook.isPresent()) {
            Book book = optionalBook.get();
            book.setTitle(updatedBook.getTitle());
            book.setAuthor(updatedBook.getAuthor());
            return bookRepository.save(book);
        } else {
            return null;
        }
    }

    // Delete a book
    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookRepository.deleteById(id);
    }
}

```

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 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;

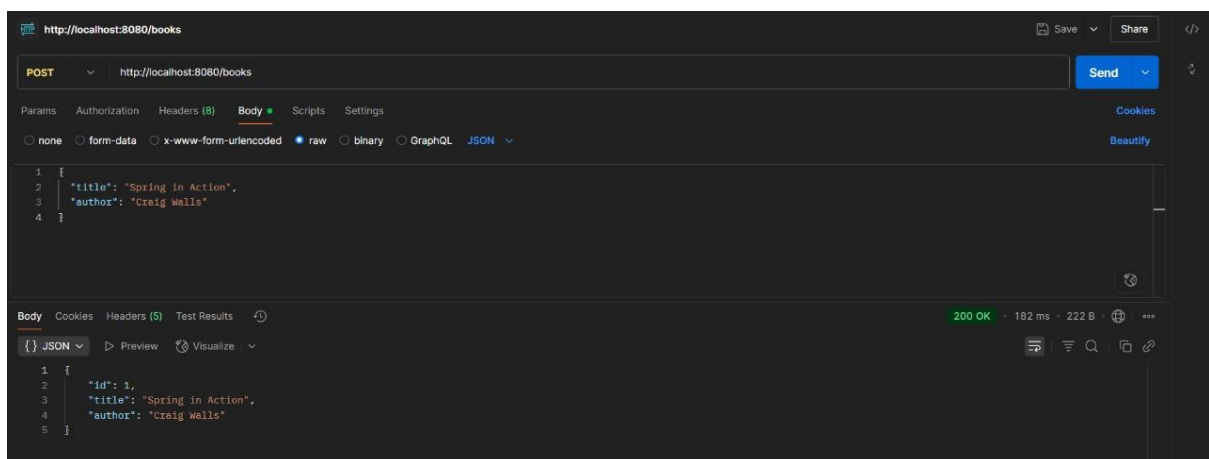
import com.library.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {
}

```

OUTPUT:

POST METHOD :



GET METHOD:

http://localhost:8080/books

GET

http://localhost:8080/books

Send

Params

Authorization

Headers (6)

Body

Scripts

Settings

Cookies

Query Params

Key	Value	Description
Key	Value	Description

Body

Cookies

Headers (5)

Test Results

200 OK · 25 ms · 340 B

JSON

Preview

Visualize

```
1 [
2   {
3     "id": 1,
4     "title": "Spring in Action",
5     "author": "Craig Walls"
6   },
7   {
8     "id": 2,
9     "title": "Effective Java",
10    "author": "Joshua Bloch"
11  },
12  {
13    "id": 3,
14    "title": "Clean Code",
15    "author": "Robert C. Martin"
16  }
17 ]
```