**1) 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:**

**BookRepository.java:**

**package** com.library.repository;

**public** **class** BookRepository {

**public** **void** save(String bookName) {

System.***out***.println("BookRepository: Saving book - " + bookName);

}

}

**BookService.java:**

**package** com.library.service;

**import** com.library.repository.BookRepository;

**public** **class** BookService {

**private** BookRepository bookRepository;

// Setter injection

**public** **void** setBookRepository(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

}

**public** **void** addBook(String bookName) {

System.***out***.println("BookService: Adding book - " + bookName);

bookRepository.save(bookName);

}

}

**LibraryManagement.java:**

**package** com.library;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** LibraryManagement {

**public** **static** **void** main(String[] args) {

// Loading Spring context from applicationContext.xml

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

// Getting the bookService bean

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

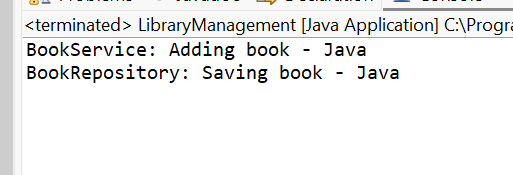
// Testing the configuration

bookService.addBook("Java");

}

}

**Output:**

****

**2) 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 serviceName;

// Constructor injection

**public** BookService(String serviceName) {

**this**.serviceName = serviceName;

}

// Setter injection

**public** **void** setBookRepository(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

}

**public** **void** addBook(String bookName) {

System.***out***.println(serviceName + ": Adding book - " + bookName);

bookRepository.save(bookName);

}

}

**LibraryManagement.java:**

**package** com.library;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** LibraryManagement{

**public** **static** **void** main(String[] args) {

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

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

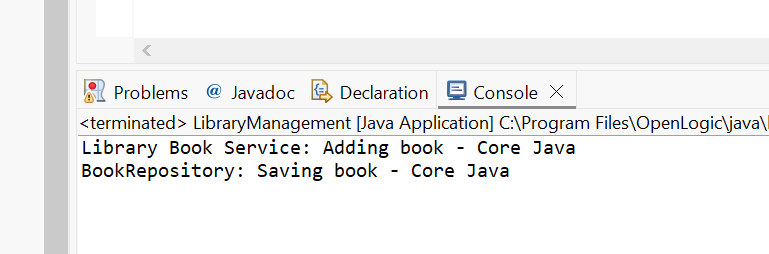
// This should print service name + book addition message

bookService.addBook("Core Java");

}

}

**Output:**

****