**Student Name:** KAMALESHWARAN A

**Registration No:** 22CSR087

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**EXERCISE 1: CONFIGURING A BASIC SPRING APPLICATION**

**Introduction:**

This is a basic **Spring Core (non-Boot)** Library Management application that demonstrates how to configure beans using an XML file (applicationContext.xml) and manage backend logic using BookService and BookRepository. It simulates typical library operations such as displaying books, adding a new book, finding a book by ID, and deleting a book.

**Objective:**

* To set up a Spring application using Maven and configure beans using XML-based configuration.
* To implement service and repository layers to handle business logic and data access separately.
* To demonstrate basic CRUD operations using Spring's ApplicationContext and dependency injection.

**Implementation Breakdown:**

**LibraryApp.java:**

package com.library;

import com.library.model.Book;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

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

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

System.out.println("Initial Book List:");

bookService.displayAllBooks();

System.out.println("\nAdding a new book:");

Book newBook = new Book(11, "Clean Code", "Robert C. Martin");

bookService.addNewBook(newBook);

System.out.println("\nUpdated Book List:");

bookService.displayAllBooks();

System.out.println("\nFind Book by ID 2:");

bookService.displayBookById(2);

System.out.println("\nDelete Book by ID 1:");

bookService.deleteBook(1);

System.out.println("\nFinal Book List:");

bookService.displayAllBooks();

}

}

**Book.java:**

package com.library.model;

public class Book {

private int id;

private String title;

private String author;

public Book() {}

public Book(int id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

public int getId() { return id; }

public void setId(int 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; }

@Override

public String toString() {

return "[" + id + "] " + title + " by " + author;

}

}

**BookService.java:**

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayAllBooks() {

List<Book> books = bookRepository.findAll();

books.forEach(System.out::println);

}

public void displayBookById(int id) {

Book book = bookRepository.findById(id);

System.out.println(book != null ? book : "Book with ID " + id + " not found.");

}

public void addNewBook(Book book) {

bookRepository.addBook(book);

System.out.println("Book added: " + book);

}

public void deleteBook(int id) {

boolean removed = bookRepository.removeBook(id);

System.out.println(removed ? "Book with ID " + id + " removed." : "Cannot remove. Book not found.");

}

}

**BookRepository.java:**

package com.library.repository;

import com.library.model.Book;

import java.util.\*;

public class BookRepository {

private final Map<Integer, Book> bookDatabase = new HashMap<>();

public BookRepository() {

bookDatabase.put(1, new Book(1, "The Alchemist", "Paulo Coelho"));

bookDatabase.put(2, new Book(2, "1984", "George Orwell"));

bookDatabase.put(3, new Book(3, "To Kill a Mockingbird", "Harper Lee"));

bookDatabase.put(4, new Book(4, "Pride and Prejudice", "Jane Austen"));

bookDatabase.put(5, new Book(5, "The Great Gatsby", "F. Scott Fitzgerald"));

bookDatabase.put(6, new Book(6, "The Catcher in the Rye", "J.D. Salinger"));

bookDatabase.put(7, new Book(7, "Moby Dick", "Herman Melville"));

bookDatabase.put(8, new Book(8, "Brave New World", "Aldous Huxley"));

bookDatabase.put(9, new Book(9, "Little Women", "Louisa May Alcott"));

bookDatabase.put(10, new Book(10, "The Lord of the Rings", "J.R.R. Tolkien"));

}

public Book findById(int id) {

return bookDatabase.get(id);

}

public List<Book> findAll() {

return new ArrayList<>(bookDatabase.values());

}

public void addBook(Book book) {

bookDatabase.put(book.getId(), book);

}

public boolean removeBook(int id) {

return bookDatabase.remove(id) != null;

}

}

**Application.properties:**

<?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">

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

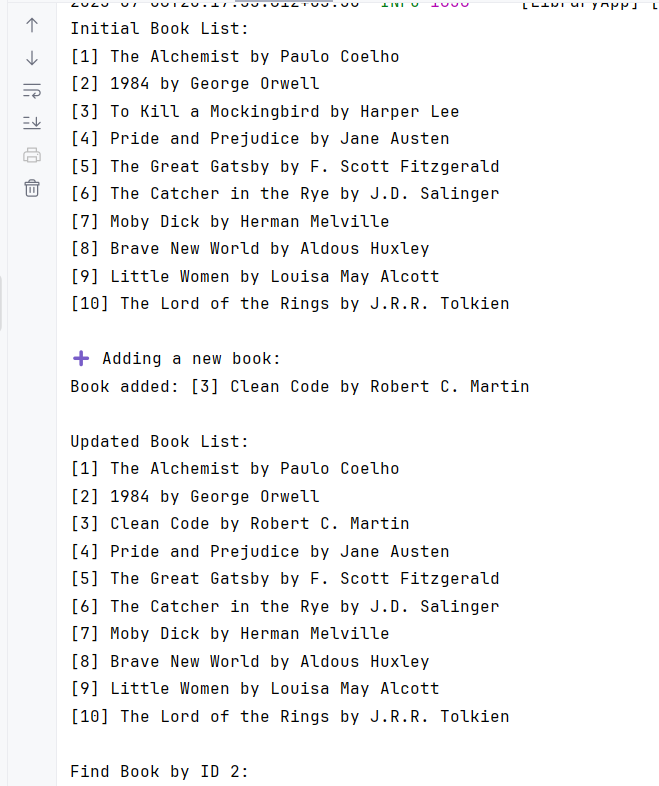
<bean id="bookService" class="com.library.service.BookService">

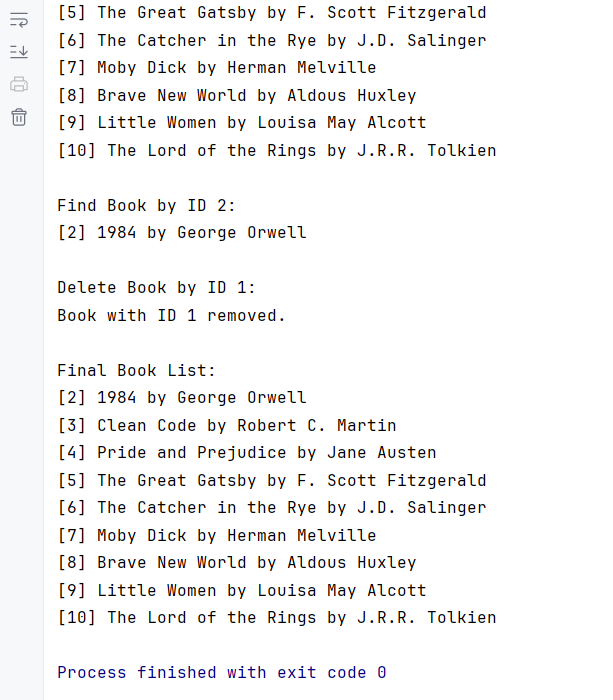
<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**Output:**

****

****

**Conclusion:**

This exercise successfully demonstrates the use of Spring Core with XML-based configuration for building a simple Library Management System.

**EXERCISE 2: IMPLEMENTING A DEPENDENCY INJECTION**

**Introduction:**

This exercise demonstrates how to implement **Dependency Injection (DI)** using the **Spring IoC container** with **XML-based configuration**. The system simulates a simple library management backend that allows users to borrow books, showcasing how service and repository classes can be wired without using annotations like @Autowired or @Service.

**Objective:**

* To configure beans in **applicationContext.xml** and wire dependencies manually using **setter injection**.
* To implement **Spring’s IoC (Inversion of Control)** principle to manage class dependencies like BookService, BookRepository, and UserService.
* To validate correct bean wiring by performing operations like listing books and borrowing them, all driven by Spring's XML configuration.

**Implementation Breakdown:**

**LibraryApp.java:**

package com.library;

import com.library.model.User;

import com.library.service.BookService;

import com.library.service.UserService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

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

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

UserService userService = context.getBean("userService", UserService.class);

System.out.println("===== Starting Library Management System =====");

System.out.println("Available Books in Library:");

bookService.displayAllBooks();

User user1 = new User(101, "Alice");

System.out.println("\nNew User Created: " + user1.getName() + " [User ID: " + user1.getUserId() + "]");

System.out.println("\nProcessing Borrow Requests...");

userService.borrowBook(user1, 2);

userService.borrowBook(user1, 5);

userService.borrowBook(user1, 999);

System.out.println("\nBorrowing Completed.");

userService.showBorrowed(user1);

System.out.println("\nTotal Books Borrowed: " + user1.getBorrowedBooks().size());

System.out.println("\n===== Library Management System Terminated =====");

}

}

**Book.java:**

package com.library.model;

public class Book {

private int id;

private String title;

private String author;

public Book() {}

public Book(int id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

public int getId() { return id; }

public String getTitle() { return title; }

public String getAuthor() { return author; }

public void setId(int id) { this.id = id; }

public void setTitle(String title) { this.title = title; }

public void setAuthor(String author) { this.author = author; }

@Override

public String toString() {

return "[" + id + "] " + title + " by " + author;

}

}

**User.java:**

package com.library.model;

import java.util.ArrayList;

import java.util.List;

public class User {

private int userId;

private String name;

private List<Book> borrowedBooks = new ArrayList<>();

public User() {}

public User(int userId, String name) {

this.userId = userId;

this.name = name;

}

public int getUserId() { return userId; }

public String getName() { return name; }

public List<Book> getBorrowedBooks() { return borrowedBooks; }

public void borrowBook(Book book) {

borrowedBooks.add(book);

System.out.println(name + " borrowed: " + book.getTitle());

}

public void listBorrowedBooks() {

System.out.println("Books borrowed by " + name + ":");

for (Book book : borrowedBooks) {

System.out.println(book);

}

}

}

**BookRepository.java:**

package com.library.repository;

import com.library.model.Book;

import java.util.\*;

public class BookRepository {

private final Map<Integer, Book> bookDatabase = new HashMap<>();

public BookRepository() {

bookDatabase.put(1, new Book(1, "The Alchemist", "Paulo Coelho"));

bookDatabase.put(2, new Book(2, "1984", "George Orwell"));

bookDatabase.put(3, new Book(3, "To Kill a Mockingbird", "Harper Lee"));

bookDatabase.put(4, new Book(4, "Pride and Prejudice", "Jane Austen"));

bookDatabase.put(5, new Book(5, "The Great Gatsby", "F. Scott Fitzgerald"));

}

public Book findById(int id) {

return bookDatabase.get(id);

}

public List<Book> findAll() {

return new ArrayList<>(bookDatabase.values());

}

**}**

**BookService.java:**

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayAllBooks() {

List<Book> books = bookRepository.findAll();

books.forEach(System.out::println);

}

public Book getBookById(int id) {

return bookRepository.findById(id);

}

}

**UserService.java:**

package com.library.service;

import com.library.model.Book;

import com.library.model.User;

public class UserService {

private BookService bookService;

public void setBookService(BookService bookService) {

this.bookService = bookService;

}

public void borrowBook(User user, int bookId) {

Book book = bookService.getBookById(bookId);

if (book != null) {

user.borrowBook(book);

} else {

System.out.println("Book not found.");

}

}

public void showBorrowed(User user) {

user.listBorrowedBooks();

}

}

**Application.properties:**

<?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">

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

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

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

</bean>

<bean id="userService" class="com.library.service.UserService">

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

</bean>

</beans>

**Output:**

****

**Conclusion:**

This exercise successfully demonstrates Spring’s **Dependency Injection using XML configuration**, replacing annotations with setter-based wiring. It highlights how Spring IoC promotes **loose coupling** and modular design through externalized bean management.

**EXERCISE 4: CREATING AND CONFIGURING A MAVEN PROJECT**

**Introduction:**

This exercise involves setting up the foundational build configuration for the Library Management System using Maven, a popular build automation tool. The project is structured to support Spring-based development by including essential dependencies and plugins to ensure proper compilation and lifecycle management.

**Objective:**

* To create a Maven-based project named LibraryManagement for managing the library application lifecycle.
* To include Spring dependencies such as spring-context, spring-aop, and spring-webmvc required for application functionality.
* To configure the Maven Compiler Plugin to use Java 1.8 for consistent compilation across environments.

**Implementation:**

**LibraryApp.java:**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

System.out.println("========================================================");

System.out.println("Welcome to The Athenaeum: Virtual Book Nexus");

System.out.println("========================================================");

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

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

bookService.displayBooks();

bookService.searchBookByTitle("history");

bookService.displayBooksByCategory("Fantasy");

bookService.borrowBook("The Hobbit");

bookService.borrowBook("1984"); // already borrowed

bookService.displayBooks(); // updated view

((ClassPathXmlApplicationContext) context).close();

}

}

**Book.java:**

package com.library.repository;

public class Book {

private String title;

private String category;

private String status;

public Book(String title, String category, String status) {

this.title = title;

this.category = category;

this.status = status;

}

public String getTitle() {

return title;

}

public String getCategory() {

return category;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

public String toString() {

return "[" + category + "] " + title + " - " + status;

}

}

**BookRepository.java:**

package com.library.repository;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

public class BookRepository {

private List<Book> books;

public BookRepository() {

books = new ArrayList<>();

books.add(new Book("The Hobbit", "Fantasy", "Available"));

books.add(new Book("1984", "Dystopian", "Borrowed"));

books.add(new Book("Brave New World", "Science Fiction", "Available"));

books.add(new Book("The Alchemist", "Philosophy", "Reserved"));

books.add(new Book("A Brief History of Time", "Science", "Available"));

books.add(new Book("The Story of Art", "Art", "Available"));

}

public List<Book> findAllBooks() {

return books;

}

public List<Book> findBooksByCategory(String category) {

return books.stream()

.filter(book -> book.getCategory().equalsIgnoreCase(category))

.collect(Collectors.toList());

}

public List<Book> searchBooks(String keyword) {

return books.stream()

.filter(book -> book.getTitle().toLowerCase().contains(keyword.toLowerCase()))

.collect(Collectors.toList());

}

public boolean borrowBook(String title) {

for (Book book : books) {

if (book.getTitle().equalsIgnoreCase(title) && book.getStatus().equalsIgnoreCase("Available")) {

book.setStatus("Borrowed");

return true;

}

}

return false;

}

}

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

import com.library.repository.Book;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

List<Book> books = bookRepository.findAllBooks();

if (books.isEmpty()) {

System.out.println("Archive is currently empty.");

return;

}

System.out.println("Books in Archive:\n");

int count = 1;

for (Book book : books) {

System.out.println(count++ + ". " + book);

}

System.out.println("\nTotal books: " + books.size());

}

public void searchBookByTitle(String keyword) {

List<Book> results = bookRepository.searchBooks(keyword);

System.out.println("\nSearch Results for \"" + keyword + "\":");

if (results.isEmpty()) {

System.out.println("No books found.");

} else {

results.forEach(System.out::println);

}

}

public void displayBooksByCategory(String category) {

List<Book> filtered = bookRepository.findBooksByCategory(category);

System.out.println("\nBooks in category: " + category);

if (filtered.isEmpty()) {

System.out.println("No books in this category.");

} else {

filtered.forEach(System.out::println);

}

}

public void borrowBook(String title) {

boolean success = bookRepository.borrowBook(title);

if (success) {

System.out.println("\nSuccess: '" + title + "' has been borrowed.");

} else {

System.out.println("\nFailed: Book either not available or already borrowed.");

}

}

}

**Application.properties:**

<?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">

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

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

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

</bean>

</beans>

**Output:**

****

**Conclusion:**

By completing this exercise, the Maven project is correctly configured to support a **Spring Core application**. The inclusion of necessary Spring dependencies and proper compiler settings ensures a stable foundation for building, running, and maintaining the Library Management System efficiently using industry-standard practices.