**//SUBHANKAR PANI**

**//SUPERSET\_ID 5005259**

**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**.

Created a new Maven project named **LibraryManagement** in Eclipse IDE.

* + Add Spring Core dependencies in the **pom.xml** file.

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.15</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.15</version>

</dependency>

</dependencies>

1. **Configure the Application Context:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.

Created an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.

* + Define beans for **BookService** and **BookRepository** in the XML file.

<?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>

1. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.

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, String author) {

Book book = new Book(title, author);

bookRepository.addBook(book);

}

}

* + Create a package **com.library.repository** and add a class **BookRepository**.

package com.library.repository;

import java.util.ArrayList;

import java.util.List;

public class BookRepository {

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

public void addBook(Book book) {

books.add(book);

}

public List<Book> getBooks() {

return books;

}

}

class Book {

private String title;

private String author;

public Book(String title, String author) {

this.title = title;

this.author = author;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

} }

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.addBook("The Great Gatsby", "F. Scott Fitzgerald");

} }

**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**.

<?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>

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

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, String author) {

Book book = new Book(title, author);

bookRepository.addBook(book);

}

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

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.addBook("The Great Gatsby", "F. Scott Fitzgerald");

}

}

**Exercise 3: Implementing Logging with Spring AOP**

**Scenario:**

The library management application requires logging capabilities to track method execution times.

**Steps:**

1. **Add Spring AOP Dependency:**
   * Update **pom.xml** to include Spring AOP dependency.

<dependencies>

<!-- ... other dependencies ... -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.15</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.7</version>

</dependency>

</dependencies>

1. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Aspect

public class LoggingAspect {

private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

@Around("execution(\* com.library.service.BookService.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long startTime = System.currentTimeMillis();

Object result = joinPoint.proceed();

long endTime = System.currentTimeMillis();

logger.info("Method {} executed in {}ms", joinPoint.getSignature().getName(), endTime - startTime);

return result;

}

}

1. **Enable AspectJ Support:**
   * Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.

<?xml version="1.0" encoding="UTF-8"?>

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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">

<!-- ... other beans ... -->

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

<aop:aspectj-autoproxy/>

<aop:config>

<aop:aspect ref="loggingAspect"/>

</aop:config>

</beans>

1. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

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.addBook("The Great Gatsby", "F. Scott Fitzgerald");

}

}

**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**.

Created a new Maven project named **LibraryManagement** using Eclipse IDE.

1. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

<dependencies>

<!-- Spring dependencies -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.15</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.15</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.15</version>

</dependency>

</dependencies>

1. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

<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>

**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.

Create a new XML file named **applicationContext.xml** in the **src/main/resources** directory:

<?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">

<!-- Define BookRepository bean -->

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

<!-- Define BookService bean -->

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

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

</bean>

</beans>

1. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

import java.util.List;

import java.util.Optional;

public class BookServiceImpl implements BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

@Override

public Book createBook(Book book) {

return bookRepository.save(book);

}

@Override

public Book updateBook(Book book) {

return bookRepository.save(book);

}

@Override

public Optional<Book> getBookById(Long id) {

return bookRepository.findById(id);

}

@Override

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@Override

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

@Override

public void deleteAllBooks() {

bookRepository.findAll().forEach(book -> bookRepository.deleteById(book.getId()));

} }

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

// Load the Spring context

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

// Get the BookService bean

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

// Test the configuration

System.out.println("Book Service: " + bookService);

}

}

**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

You need to simplify the configuration of beans in the library management application using annotations.

**Steps:**

1. **Enable Component Scanning:**
   * Update **applicationContext.xml** to include component scanning for the **com.library** package.

<?xml version="1.0" encoding="UTF-8"?>

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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 component scanning for the com.library package -->

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

</beans>

1. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class BookServiceImpl implements BookService {

private BookRepository bookRepository;

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

@Override

public Book saveBook(Book book) {

return bookRepository.save(book);

}

@Override

public Optional<Book> findBookById(Long id) {

return bookRepository.findById(id);

}

@Override

public List<Book> findAllBooks() {

return bookRepository.findAll();

}

@Override

public void deleteBookById(Long id) {

bookRepository.deleteById(id);

} }

* + Use **@Repository** annotation for the **BookRepository** class.

package com.library.repository;

import com.library.model.Book;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

@Repository

public class BookRepositoryImpl implements BookRepository {

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

@Override

public Book save(Book book) {

books.add(book);

return book;

}

@Override

public Optional<Book> findById(Long id) {

return books.stream()

.filter(book -> book.getId().equals(id))

.findFirst();

}

@Override

public List<Book> findAll() {

return books;

}

@Override

public void deleteById(Long id) {

books.removeIf(book -> book.getId().equals(id));

} }

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

// Load the Spring context

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

// Get the BookService bean

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

// Test the configuration

System.out.println("Book Service: " + bookService);

}

}

**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**.

<?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">

<!-- Configure BookRepository bean -->

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

<!-- Configure BookService bean with constructor injection -->

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

<constructor-arg 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**.

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

import java.util.List;

import java.util.Optional;

public class BookServiceImpl implements BookService {

private BookRepository bookRepository;

// Constructor injection

public BookServiceImpl(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

@Override

public Book createBook(Book book) {

return bookRepository.save(book);

}

@Override

public Book updateBook(Book book) {

return bookRepository.save(book);

}

@Override

public Optional<Book> getBookById(Long id) {

return bookRepository.findById(id);

}

@Override

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@Override

public void deleteBook(Long id) {

bookRepository.deleteById(id);

}

@Override

public void deleteAllBooks() {

bookRepository.deleteAll();

}

}

1. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

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");

// Verify constructor injection

System.out.println("BookRepository instance: " + bookService.getBookRepository());

// Verify setter injection

bookService.setBookRepository(new BookRepositoryImpl());

System.out.println("Updated BookRepository instance: " + bookService.getBookRepository());

}

}

**Exercise 8: Implementing Basic AOP with Spring**

**Scenario:**

The library management application requires basic AOP functionality to separate cross-cutting concerns like logging and transaction management.

**Steps:**

1. **Define an Aspect:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect**.

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Aspect

public class LoggingAspect {

private static final Logger LOGGER = LoggerFactory.getLogger(LoggingAspect.class);

// Advice methods will be defined here

}

1. **Create Advice Methods:**
   * Define advice methods in **LoggingAspect** for logging before and after method execution.

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@Aspect

public class LoggingAspect {

private static final Logger LOGGER = LoggerFactory.getLogger(LoggingAspect.class);

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

public void logBeforeMethodExecution(JoinPoint joinPoint) {

LOGGER.info("Before executing method: {}", joinPoint.getSignature());

}

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

public void logAfterMethodExecution(JoinPoint joinPoint) {

LOGGER.info("After executing method: {}", joinPoint.getSignature());

}

}

1. **Configure the Aspect:**
   * Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.

<?xml version="1.0" encoding="UTF-8"?>

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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">

<!-- Configure BookRepository bean -->

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

<!-- Configure BookService bean -->

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

<constructor-arg ref="bookRepository"/>

</bean>

<!-- Configure LoggingAspect -->

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

<!-- Enable AspectJ auto-proxying -->

<aop:aspectj-autoproxy/>

</beans>

1. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class to verify the AOP functionality.

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");

// Call a method on the BookService instance

bookService.createBook(new Book("Java Programming", "John Doe"));

// Verify the logging 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**.

Created.

1. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.

<!-- Maven dependencies -->

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

</dependency>

</dependencies>

// Gradle dependencies

dependencies {

implementation 'org.springframework.boot:spring-boot-starter-web'

implementation 'org.springframework.boot:spring-boot-starter-data-jpa'

implementation 'com.h2database:h2'

}

1. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.

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

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create-drop

1. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.

// Book entity

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// Getters and setters

}

// BookRepository interface

public interface BookRepository extends JpaRepository<Book, Long> {

}

1. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

return bookRepository.findById(id).orElseThrow();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

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

Book existingBook = bookRepository.findById(id).orElseThrow();

existingBook.setTitle(book.getTitle());

existingBook.setAuthor(book.getAuthor());

return bookRepository.save(existingBook);

}

@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.

mvn spring-boot:run

Once the application is running, we can test the REST endpoints using the following URLs:

* **GET /api/books**: Retrieve all books
* **GET /api/books/{id}**: Retrieve a book by ID
* **POST /api/books**: Create a new book
* **PUT /api/books/{id}**: Update a book
* **DELETE /api/books/{id}**: Delete a book