Ex:1

Scenario 1

DECLARE

CURSOR customer\_cursor IS

SELECT CustomerID, InterestRate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM c.DOB) > 60;

BEGIN

FOR customer\_rec IN customer\_cursor LOOP

UPDATE Loans

SET InterestRate = InterestRate - 1

WHERE CustomerID = customer\_rec.CustomerID;

END LOOP;

COMMIT;

END;

Scenario 2:

DECLARE

CURSOR vip\_cursor IS

SELECT CustomerID

FROM Customers

WHERE Balance > 10000;

BEGIN

FOR vip\_rec IN vip\_cursor LOOP

UPDATE Customers

SET IsVIP = 'T'

WHERE CustomerID = vip\_rec.CustomerID;

END LOOP;

COMMIT; END;

Scenario 3:

DECLARE

CURSOR loan\_cursor IS

SELECT c.CustomerID, c.Name, l.EndDate

FROM Customers c

JOIN Loans l ON c.CustomerID = l.CustomerID

WHERE l.EndDate BETWEEN SYSDATE AND SYSDATE + 30;

BEGIN

FOR loan\_rec IN loan\_cursor LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan for customer ' || loan\_rec.Name || ' (ID: ' || loan\_rec.CustomerID || ') is due on ' || TO\_CHAR(loan\_rec.EndDate, 'YYYY-MM-DD'));

END LOOP;

END;

Ex 2

Scenario 1:

CREATE OR REPLACE PROCEDURE SafeTransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

BEGIN

-- Check if the from account has sufficient funds

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END SafeTransferFunds;

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateSalary (

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) IS

e\_employee\_not\_found EXCEPTION;

v\_count NUMBER;

BEGIN

-- Check if the employee exists

SELECT COUNT(\*) INTO v\_count FROM Employees WHERE EmployeeID = p\_employee\_id;

IF v\_count = 0 THEN

RAISE e\_employee\_not\_found;

END IF;

-- Update the employee's salary

UPDATE Employees

SET Salary = Salary \* (1 + p\_percentage / 100)

WHERE EmployeeID = p\_employee\_id;

COMMIT;

EXCEPTION

WHEN e\_employee\_not\_found THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Employee ID ' || p\_employee\_id || ' not found.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END UpdateSalary;

Scenario 3:

CREATE OR REPLACE PROCEDURE AddNewCustomer (

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) IS

e\_customer\_exists EXCEPTION;

v\_count NUMBER;

BEGIN

-- Check if the customer ID already exists

SELECT COUNT(\*) INTO v\_count FROM Customers WHERE CustomerID = p\_customer\_id;

IF v\_count > 0 THEN

RAISE e\_customer\_exists;

END IF;

-- Insert the new customer

INSERT INTO Customers (CustomerID, Name, DOB, Balance, LastModified)

VALUES (p\_customer\_id, p\_name, p\_dob, p\_balance, SYSDATE);

COMMIT;

EXCEPTION

WHEN e\_customer\_exists THEN

DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID ' || p\_customer\_id || ' already exists.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END AddNewCustomer;

Ex 3

Scenario 1:

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01

WHERE AccountType = 'Savings';

COMMIT;

END ProcessMonthlyInterest;

Scenario 2:

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus (

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) IS

BEGIN

UPDATE Employees

SET Salary = Salary \* (1 + p\_bonus\_percentage / 100)

WHERE Department = p\_department;

COMMIT;

END UpdateEmployeeBonus;

Scenario 3:

CREATE OR REPLACE PROCEDURE TransferFunds (

p\_from\_account\_id IN NUMBER,

p\_to\_account\_id IN NUMBER,

p\_amount IN NUMBER

) IS

e\_insufficient\_funds EXCEPTION;

v\_from\_balance NUMBER;

BEGIN

-- Check if the source account has sufficient funds

SELECT Balance INTO v\_from\_balance FROM Accounts WHERE AccountID = p\_from\_account\_id FOR UPDATE;

IF v\_from\_balance < p\_amount THEN

RAISE e\_insufficient\_funds;

END IF;

-- Deduct the amount from the source account

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account\_id;

-- Add the amount to the destination account

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account\_id;

COMMIT;

EXCEPTION

WHEN e\_insufficient\_funds THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: Insufficient funds in account ' || p\_from\_account\_id);

WHEN OTHERS THEN

ROLLBACK;

DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);

END TransferFunds;

Exscercise 4:

Scenario 1:

CREATE OR REPLACE FUNCTION CalculateAge (

date\_of\_birth DATE

) RETURN NUMBER IS

age NUMBER;

BEGIN

-- Calculate the age in years

age := FLOOR((SYSDATE - date\_of\_birth) / 365.25);

RETURN age;

END;

/

Scenario 2:

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment (

loan\_amount NUMBER,

annual\_interest\_rate NUMBER,

loan\_duration\_years NUMBER

) RETURN NUMBER IS

monthly\_interest\_rate NUMBER;

number\_of\_payments NUMBER;

monthly\_installment NUMBER;

BEGIN

monthly\_interest\_rate := annual\_interest\_rate / 12 / 100;

number\_of\_payments := loan\_duration\_years \* 12;

monthly\_installment := loan\_amount \* monthly\_interest\_rate /

(1 - POWER(1 + monthly\_interest\_rate, -number\_of\_payments));

RETURN monthly\_installment;

END;

/

Scenario 3:

CREATE OR REPLACE FUNCTION HasSufficientBalance (

account\_id NUMBER,

amount NUMBER

) RETURN BOOLEAN IS

account\_balance NUMBER;

BEGIN

SELECT balance INTO account\_balance

FROM accounts

WHERE account\_id = account\_id;

IF account\_balance >= amount THEN

RETURN TRUE;

ELSE

RETURN FALSE;

END IF;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

/

Exercise 5:

Scenario 1:

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

/

Scenario 2:

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (transaction\_id, account\_id, transaction\_date, amount, action)

VALUES (:NEW.transaction\_id, :NEW.account\_id, :NEW.transaction\_date, :NEW.amount, 'INSERT');

END;

/

Scenario 3:

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

account\_balance NUMBER;

BEGIN

SELECT balance INTO account\_balance

FROM accounts

WHERE account\_id = :NEW.account\_id FOR UPDATE;

IF :NEW.amount < 0 THEN

IF account\_balance + :NEW.amount < 0 THEN

RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds for this withdrawal.');

END IF;

END IF;

IF :NEW.amount <= 0 THEN

RAISE\_APPLICATION\_ERROR(-20002, 'Deposit amount must be positive.');

END IF;

END;

/

Exercise 6:

Scenario 1:

DECLARE

CURSOR GenerateMonthlyStatements IS

SELECT c.customer\_id, c.customer\_name, t.transaction\_date, t.amount, t.description

FROM Customers c

JOIN Transactions t ON c.customer\_id = t.customer\_id

WHERE EXTRACT(MONTH FROM t.transaction\_date) = EXTRACT(MONTH FROM SYSDATE)

AND EXTRACT(YEAR FROM t.transaction\_date) = EXTRACT(YEAR FROM SYSDATE)

ORDER BY c.customer\_id, t.transaction\_date;

transaction\_record GenerateMonthlyStatements%ROWTYPE;

current\_customer\_id NUMBER := NULL;

BEGIN

OPEN GenerateMonthlyStatements;

LOOP

FETCH GenerateMonthlyStatements INTO transaction\_record;

EXIT WHEN GenerateMonthlyStatements%NOTFOUND;

IF current\_customer\_id IS NULL OR current\_customer\_id != transaction\_record.customer\_id THEN

IF current\_customer\_id IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('--- End of Statement ---');

END IF;

current\_customer\_id := transaction\_record.customer\_id;

DBMS\_OUTPUT.PUT\_LINE('Customer ID: ' || transaction\_record.customer\_id);

DBMS\_OUTPUT.PUT\_LINE('Customer Name: ' || transaction\_record.customer\_name);

DBMS\_OUTPUT.PUT\_LINE('--- Transactions ---');

END IF;

DBMS\_OUTPUT.PUT\_LINE('Date: ' || TO\_CHAR(transaction\_record.transaction\_date, 'YYYY-MM-DD'));

DBMS\_OUTPUT.PUT\_LINE('Amount: ' || transaction\_record.amount);

DBMS\_OUTPUT.PUT\_LINE('Description: ' || transaction\_record.description);

DBMS\_OUTPUT.PUT\_LINE('----------------------');

END LOOP;

IF current\_customer\_id IS NOT NULL THEN

DBMS\_OUTPUT.PUT\_LINE('--- End of Statement ---');

END IF;

CLOSE GenerateMonthlyStatements;

END;

/

Scenario 2:

DECLARE

CURSOR ApplyAnnualFee IS

SELECT account\_id, balance

FROM Accounts

FOR UPDATE;

account\_record ApplyAnnualFee%ROWTYPE;

annual\_fee CONSTANT NUMBER := 100; -- Example fee amount

BEGIN

OPEN ApplyAnnualFee;

LOOP

FETCH ApplyAnnualFee INTO account\_record;

EXIT WHEN ApplyAnnualFee%NOTFOUND;

UPDATE Accounts

SET balance = balance - annual\_fee

WHERE account\_id = account\_record.account\_id;

DBMS\_OUTPUT.PUT\_LINE('Applied annual fee to account ID: ' || account\_record.account\_id ||

', New Balance: ' || (account\_record.balance - annual\_fee));

END LOOP;

CLOSE ApplyAnnualFee;

END;

/

Scenario 3:

DECLARE

CURSOR UpdateLoanInterestRates IS

SELECT loan\_id, interest\_rate

FROM Loans

FOR UPDATE;

loan\_record UpdateLoanInterestRates%ROWTYPE;

new\_interest\_rate CONSTANT NUMBER := 4.5; -- Example new interest rate

BEGIN

OPEN UpdateLoanInterestRates;

LOOP

FETCH UpdateLoanInterestRates INTO loan\_record;

EXIT WHEN UpdateLoanInterestRates%NOTFOUND;

UPDATE Loans

SET interest\_rate = new\_interest\_rate

WHERE loan\_id = loan\_record.loan\_id;

DBMS\_OUTPUT.PUT\_LINE('Updated interest rate for loan ID: ' || loan\_record.loan\_id ||

', New Interest Rate: ' || new\_interest\_rate);

END LOOP;

CLOSE UpdateLoanInterestRates;

END;

/

Exercise 7:

Scenario 1:

CREATE OR REPLACE PACKAGE CustomerManagement AS

PROCEDURE AddCustomer(p\_customer\_id NUMBER, p\_customer\_name VARCHAR2, p\_balance NUMBER);

PROCEDURE UpdateCustomer(p\_customer\_id NUMBER, p\_customer\_name VARCHAR2);

FUNCTION GetCustomerBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

PROCEDURE AddCustomer(p\_customer\_id NUMBER, p\_customer\_name VARCHAR2, p\_balance NUMBER) IS

BEGIN

INSERT INTO Customers (customer\_id, customer\_name, balance)

VALUES (p\_customer\_id, p\_customer\_name, p\_balance);

END AddCustomer;

PROCEDURE UpdateCustomer(p\_customer\_id NUMBER, p\_customer\_name VARCHAR2) IS

BEGIN

UPDATE Customers

SET customer\_name = p\_customer\_name

WHERE customer\_id = p\_customer\_id;

END UpdateCustomer;

FUNCTION GetCustomerBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_balance NUMBER;

BEGIN

SELECT balance INTO v\_balance

FROM Customers

WHERE customer\_id = p\_customer\_id;

RETURN v\_balance;

END GetCustomerBalance;

END CustomerManagement;

/

Scenario 2:

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_employee\_name VARCHAR2, p\_salary NUMBER);

PROCEDURE UpdateEmployeeDetails(p\_employee\_id NUMBER, p\_employee\_name VARCHAR2);

FUNCTION CalculateAnnualSalary(p\_employee\_id NUMBER) RETURN NUMBER;

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id NUMBER, p\_employee\_name VARCHAR2, p\_salary NUMBER) IS

BEGIN

INSERT INTO Employees (employee\_id, employee\_name, salary)

VALUES (p\_employee\_id, p\_employee\_name, p\_salary);

END HireEmployee;

PROCEDURE UpdateEmployeeDetails(p\_employee\_id NUMBER, p\_employee\_name VARCHAR2) IS

BEGIN

UPDATE Employees

SET employee\_name = p\_employee\_name

WHERE employee\_id = p\_employee\_id;

END UpdateEmployeeDetails;

FUNCTION CalculateAnnualSalary(p\_employee\_id NUMBER) RETURN NUMBER IS

v\_salary NUMBER;

BEGIN

SELECT salary \* 12 INTO v\_salary

FROM Employees

WHERE employee\_id = p\_employee\_id;

RETURN v\_salary;

END CalculateAnnualSalary;

END EmployeeManagement;

/

Scenario 3:

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_initial\_balance NUMBER);

PROCEDURE CloseAccount(p\_account\_id NUMBER);

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER;

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id NUMBER, p\_customer\_id NUMBER, p\_initial\_balance NUMBER) IS

BEGIN

INSERT INTO Accounts (account\_id, customer\_id, balance)

VALUES (p\_account\_id, p\_customer\_id, p\_initial\_balance);

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE account\_id = p\_account\_id;

END CloseAccount;

FUNCTION GetTotalBalance(p\_customer\_id NUMBER) RETURN NUMBER IS

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(balance) INTO v\_total\_balance

FROM Accounts

WHERE customer\_id = p\_customer\_id;

RETURN v\_total\_balance;

END GetTotalBalance;

END AccountOperations;

/

WEEK 2:

SPRING CORE MAVEN:

Pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

</dependencies>

</project>

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

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>

BookService class:

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 performService() {

System.out.println("BookService is performing an operation...");

bookRepository.saveBook();

}

}

BookRepository class:

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("BookRepository is saving a book...");

}

}

LibraryApp:

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) {

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

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

bookService.performService();

}

}

Exercise 2:

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

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Define the BookRepository bean -->

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

<!-- Define the BookService bean and inject the BookRepository bean into it -->

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

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

</bean>

</beans>

BookService class:

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 performService() {

System.out.println("BookService is performing an operation...");

bookRepository.saveBook();

}

}

LibraryApp:

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) {

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

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

bookService.performService();

}

}

Exercise 3:

Pom.xml

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.29</version>

</dependency>

</dependencies>

LoggingAspect class:

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class LoggingAspect {

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

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object proceed = joinPoint.proceed();

long executionTime = System.currentTimeMillis() - start;

System.out.println(joinPoint.getSignature() + " executed in " + executionTime + "ms");

return proceed;

}

}

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"

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">

<aop:aspectj-autoproxy/>

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

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

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

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

</bean>

</beans>

LibraryApp:

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) {

// Load the Spring context from the applicationContext.xml file

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

// Get the BookService bean and use it

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

bookService.performService();

}

}

Pom.xml:

<project xmlns="http://maven.apache.org/POM/4.0.0"

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Context dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring AOP dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring WebMVC dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Servlet API (required for Spring WebMVC) -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</depe

Pom.xml:

<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>

Pom.xml:

<project xmlns="http://maven.apache.org/POM/4.0.0"

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Context dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.29</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<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>

</project>

Exercise 5:

BookService class:

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method for dependency injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Method to perform some operation (for testing)

public void performService() {

System.out.println("BookService is performing an operation...");

bookRepository.saveBook();

}

}

BookRepository class:

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("BookRepository is saving a book...");

}

}

LibraryApp

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) {

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

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

bookService.performService();

}

}

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"

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 in the com.library package -->

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

</beans>

BookService class:

package com.library.service;

import org.springframework.stereotype.Service;

import com.library.repository.BookRepository;

@Service

public class BookService {

private final BookRepository bookRepository;

// Constructor-based dependency injection

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void performService() {

System.out.println("BookService is performing an operation...");

bookRepository.saveBook();

}

}

BookRepository class:

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void saveBook() {

System.out.println("BookRepository is saving a book...");

}

}

LibraryMangementApplication:

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) {

// Load the Spring context from the applicationContext.xml file

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

// Get the BookService bean (automatically configured via annotations)

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

bookService.performService();

}

}

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"

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">

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

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

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

<constructor-arg ref="bookRepository"/>

</bean>

</beans>

BookService class:

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void performService() {

System.out.println("BookService is performing an operation...");

bookRepository.saveBook();

}

}

Bookservice bean

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

<constructor-arg ref="bookRepository"/>

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

</bean>

LibraryManagementApplication:

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) {

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

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

bookService.performService();

}

}

Exercise 8:

LoggingAspect class:

package com.library.aspect;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.After;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

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

public void logBefore() {

System.out.println("LoggingAspect: Method execution started...");

}

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

public void logAfter() {

System.out.println("LoggingAspect: Method execution finished...");

}

}

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"

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

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/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

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

<aop:aspectj-autoproxy/>

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

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

<constructor-arg ref="bookRepository"/>

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

</bean>

</beans>

LiberaryManagementApplication:

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) {

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

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

bookService.performService();

}

}

Exercise 9:

Step 1: Create a Spring Boot Project

1. Use Spring Initializr:
   * Go to [Spring Initializr](https://start.spring.io/).
   * Set the project name as LibraryManagement.
   * Choose the following settings:
     + **Project:** Maven Project
     + **Language:** Java
     + **Spring Boot Version:** 3.x.x (or the latest stable version)
     + **Packaging:** Jar
     + **Java Version:** 17 (or 11, depending on your setup)
   * Add the following dependencies:
     + Spring Web
     + Spring Data JPA
     + H2 Database
   * Then generate

Application.properties:

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

Book class

package com.librarymanagement.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;

private String isbn;

// 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;

}

public String getIsbn() {

return isbn;

}

public void setIsbn(String isbn) {

this.isbn = isbn;

}

}

BookREpository interface:

package com.librarymanagement.repository;

import com.librarymanagement.entity.Book;

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

import org.springframework.stereotype.Repository;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

BookController class:

package com.librarymanagement.controller;

import com.librarymanagement.entity.Book;

import com.librarymanagement.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

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

Book book = bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setIsbn(bookDetails.getIsbn());

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

Book book = bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

bookRepository.delete(book);

}

}

LibraryManagementApplication

package com.librarymanagement;

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);

}

}