**PL/SQL(EXERCISES)**

**Exercise 1: Control Structures**

**Scenario 1: Applying a Discount to Loan Interest Rates**

BEGIN

FOR rec IN (SELECT CustomerID, InterestRate FROM Loans JOIN Customers ON Loans.CustomerID = Customers.CustomerID WHERE EXTRACT(YEAR FROM SYSDATE) - EXTRACT(YEAR FROM DOB) > 60) LOOP

UPDATE Loans

SET InterestRate = InterestRate - 0.01

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

END;

**Scenario 2: Promoting Customers to VIP Status**

BEGIN

FOR rec IN (SELECT CustomerID, Balance FROM Customers WHERE Balance > 10000) LOOP

UPDATE Customers

SET IsVIP = TRUE

WHERE CustomerID = rec.CustomerID;

END LOOP;

COMMIT;

END;

**Scenario 3: Sending Reminders for Due Loans**

BEGIN

FOR rec IN (SELECT CustomerID, LoanID, EndDate FROM Loans WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30) LOOP

DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ' || rec.LoanID || ' for Customer ' || rec.CustomerID || ' is due on ' || rec.EndDate);

END LOOP;

END;

**Exercise 2: Error Handling**

**Scenario 1: Safe Fund Transfer**

CREATE OR REPLACE PROCEDURE SafeTransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) AS

BEGIN

BEGIN

-- Transfer funds

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Account not found.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

ROLLBACK;

END;

END;

**Scenario 2: Updating Employee Salary**

CREATE OR REPLACE PROCEDURE UpdateSalary(

p\_employee\_id IN NUMBER,

p\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE EmployeeID = p\_employee\_id;

IF SQL%ROWCOUNT = 0 THEN

DBMS\_OUTPUT.PUT\_LINE('Employee ID does not exist.');

ROLLBACK;

ELSE

COMMIT;

END IF;

EXCEPTION

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

ROLLBACK;

END;

**Scenario 3: Adding a New Customer**

CREATE OR REPLACE PROCEDURE AddNewCustomer(

p\_customer\_id IN NUMBER,

p\_name IN VARCHAR2,

p\_dob IN DATE,

p\_balance IN NUMBER

) AS

BEGIN

BEGIN

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

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

COMMIT;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN

DBMS\_OUTPUT.PUT\_LINE('Customer with this ID already exists.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

ROLLBACK;

END;

END;

**Exercise 3: Stored Procedures**

**Scenario 1: Processing Monthly Interest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest AS

BEGIN

UPDATE Accounts

SET Balance = Balance \* 1.01;

COMMIT;

END;

**Scenario 2: Updating Employee Bonuses**

CREATE OR REPLACE PROCEDURE UpdateEmployeeBonus(

p\_department IN VARCHAR2,

p\_bonus\_percentage IN NUMBER

) AS

BEGIN

UPDATE Employees

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

WHERE Department = p\_department;

COMMIT;

END;

**Scenario 3: Transferring Funds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds(

p\_from\_account IN NUMBER,

p\_to\_account IN NUMBER,

p\_amount IN NUMBER

) AS

BEGIN

DECLARE

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_from\_account;

IF v\_balance < p\_amount THEN

DBMS\_OUTPUT.PUT\_LINE('Insufficient funds.');

RETURN;

END IF;

UPDATE Accounts

SET Balance = Balance - p\_amount

WHERE AccountID = p\_from\_account;

UPDATE Accounts

SET Balance = Balance + p\_amount

WHERE AccountID = p\_to\_account;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

DBMS\_OUTPUT.PUT\_LINE('Account not found.');

ROLLBACK;

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE('An error occurred: ' || SQLERRM);

ROLLBACK;

END;

END;

**Exercise 4: Functions**

**Scenario 1: Calculating Customer Age**

CREATE OR REPLACE FUNCTION CalculateAge(p\_dob DATE) RETURN NUMBER IS

BEGIN

RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);

END;

**Scenario 2: Computing Monthly Installment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(

p\_loan\_amount NUMBER,

p\_interest\_rate NUMBER,

p\_duration\_years NUMBER

) RETURN NUMBER IS

v\_monthly\_rate NUMBER;

v\_num\_payments NUMBER;

BEGIN

v\_monthly\_rate := p\_interest\_rate / 12 / 100;

v\_num\_payments := p\_duration\_years \* 12;

RETURN (p\_loan\_amount \* v\_monthly\_rate) / (1 - POWER(1 + v\_monthly\_rate, -v\_num\_payments));

END;

**Scenario 3: Checking Account Balance**

CREATE OR REPLACE FUNCTION HasSufficientBalance(

p\_account\_id NUMBER,

p\_amount NUMBER

) RETURN BOOLEAN IS

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = p\_account\_id;

RETURN v\_balance >= p\_amount;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN FALSE;

END;

**Exercise 5: Triggers**

**Scenario 1: Updating Last Modified Date**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified

BEFORE UPDATE ON Customers

FOR EACH ROW

BEGIN

:NEW.LastModified := SYSDATE;

END;

**Scenario 2: Maintaining an Audit Log**

CREATE OR REPLACE TRIGGER LogTransaction

AFTER INSERT ON Transactions

FOR EACH ROW

BEGIN

INSERT INTO AuditLog (TransactionID, Action, ActionDate)

VALUES (:NEW.TransactionID, 'INSERT', SYSDATE);

END;

**Scenario 3: Enforcing Transaction Rules**

CREATE OR REPLACE TRIGGER CheckTransactionRules

BEFORE INSERT ON Transactions

FOR EACH ROW

DECLARE

v\_balance NUMBER;

BEGIN

IF :NEW.TransactionType = 'Withdrawal' THEN

SELECT Balance INTO v\_balance

FROM Accounts

WHERE AccountID = :NEW.AccountID;

IF v\_balance < :NEW.Amount THEN

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

END IF;

ELSIF :NEW.TransactionType = 'Deposit' AND :NEW.Amount <= 0 THEN

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

END IF;

END;

**Exercise 6: Cursors**

**Scenario 1: Generating Monthly Statements**

DECLARE

CURSOR c\_monthly\_statements IS

SELECT AccountID, SUM(Amount) AS TotalAmount

FROM Transactions

WHERE EXTRACT(MONTH FROM TransactionDate) = EXTRACT(MONTH FROM SYSDATE)

GROUP BY AccountID;

v\_account\_id NUMBER;

v\_total\_amount NUMBER;

BEGIN

OPEN c\_monthly\_statements;

LOOP

FETCH c\_monthly\_statements INTO v\_account\_id, v\_total\_amount;

EXIT WHEN c\_monthly\_statements%NOTFOUND;

DBMS\_OUTPUT.PUT\_LINE('Statement for Account ' || v\_account\_id || ': Total Transactions = ' || v\_total\_amount);

END LOOP;

CLOSE c\_monthly\_statements;

END;

**Scenario 2: Applying Annual Fee**

DECLARE

CURSOR c\_accounts IS

SELECT AccountID

FROM Accounts;

v\_account\_id NUMBER;

BEGIN

OPEN c\_accounts;

LOOP

FETCH c\_accounts INTO v\_account\_id;

EXIT WHEN c\_accounts%NOTFOUND;

UPDATE Accounts

SET Balance = Balance - 50 -- assuming the annual fee is $50

WHERE AccountID = v\_account\_id;

END LOOP;

CLOSE c\_accounts;

COMMIT;

END;

**Scenario 3: Updating Loan Interest Rates**

DECLARE

CURSOR c\_loans IS

SELECT LoanID, InterestRate

FROM Loans;

v\_loan\_id NUMBER;

v\_interest\_rate NUMBER;

BEGIN

OPEN c\_loans;

LOOP

FETCH c\_loans INTO v\_loan\_id, v\_interest\_rate;

EXIT WHEN c\_loans%NOTFOUND;

UPDATE Loans

SET InterestRate = InterestRate + 0.5 -- assuming a 0.5% increase

WHERE LoanID = v\_loan\_id;

END LOOP;

CLOSE c\_loans;

COMMIT;

END;

**Exercise 7: Packages**

**Scenario 1: Customer Management Package**

CREATE OR REPLACE PACKAGE CustomerManagement AS

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

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

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

END CustomerManagement;

/

CREATE OR REPLACE PACKAGE BODY CustomerManagement AS

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

BEGIN

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

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

COMMIT;

END AddCustomer;

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

BEGIN

UPDATE Customers

SET Name = p\_name, DOB = p\_dob, LastModified = SYSDATE

WHERE CustomerID = p\_customer\_id;

COMMIT;

END UpdateCustomer;

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

v\_balance NUMBER;

BEGIN

SELECT Balance INTO v\_balance

FROM Customers

WHERE CustomerID = p\_customer\_id;

RETURN v\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN NULL;

END GetCustomerBalance;

END CustomerManagement;

**Scenario 2: Employee Management Package**

CREATE OR REPLACE PACKAGE EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2, p\_hire\_date IN DATE);

PROCEDURE UpdateEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER);

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

END EmployeeManagement;

/

CREATE OR REPLACE PACKAGE BODY EmployeeManagement AS

PROCEDURE HireEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER, p\_department IN VARCHAR2, p\_hire\_date IN DATE) IS

BEGIN

INSERT INTO Employees (EmployeeID, Name, Position, Salary, Department, HireDate)

VALUES (p\_employee\_id, p\_name, p\_position, p\_salary, p\_department, p\_hire\_date);

COMMIT;

END HireEmployee;

PROCEDURE UpdateEmployee(p\_employee\_id IN NUMBER, p\_name IN VARCHAR2, p\_position IN VARCHAR2, p\_salary IN NUMBER) IS

BEGIN

UPDATE Employees

SET Name = p\_name, Position = p\_position, Salary = p\_salary

WHERE EmployeeID = p\_employee\_id;

COMMIT;

END UpdateEmployee;

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

v\_salary NUMBER;

BEGIN

SELECT Salary INTO v\_salary

FROM Employees

WHERE EmployeeID = p\_employee\_id;

RETURN v\_salary \* 12;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END CalculateAnnualSalary;

END EmployeeManagement;

**Scenario 3: Account Operations Package**

CREATE OR REPLACE PACKAGE AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER);

PROCEDURE CloseAccount(p\_account\_id IN NUMBER);

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

END AccountOperations;

/

CREATE OR REPLACE PACKAGE BODY AccountOperations AS

PROCEDURE OpenAccount(p\_account\_id IN NUMBER, p\_customer\_id IN NUMBER, p\_account\_type IN VARCHAR2, p\_balance IN NUMBER) IS

BEGIN

INSERT INTO Accounts (AccountID, CustomerID, AccountType, Balance, LastModified)

VALUES (p\_account\_id, p\_customer\_id, p\_account\_type, p\_balance, SYSDATE);

COMMIT;

END OpenAccount;

PROCEDURE CloseAccount(p\_account\_id IN NUMBER) IS

BEGIN

DELETE FROM Accounts

WHERE AccountID = p\_account\_id;

COMMIT;

END CloseAccount;

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

v\_total\_balance NUMBER;

BEGIN

SELECT SUM(Balance) INTO v\_total\_balance

FROM Accounts

WHERE CustomerID = p\_customer\_id;

RETURN v\_total\_balance;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

RETURN 0;

END GetTotalBalance;

END AccountOperations;

**SPRING CORE MAVEN**

**Exercise 1: Configuring a Basic Spring Application**

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.3.10</version>

</dependency>

</dependencies>

**applicationContext.xml**

<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.java**

package com.library.service;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Other service methods

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

// Repository methods

}

**LibraryManagementApplication.java**

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");

// Test the configuration

}

}

**Exercise 2: Implementing Dependency Injection**

**applicationContext.xml**

<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.java**

package com.library.service;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Other service methods

}

**LibraryManagementApplication.java**

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");

// Test the configuration

}

}

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

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.10</version>

</dependency>

</dependencies>

**applicationContext.xml**

<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"/>

</beans>

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.springframework.stereotype.Component;

@Aspect

@Component

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;

}

}

**LibraryManagementApplication.java**

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");

// Test logging

}

}

**Exercise 4: Creating and Configuring a Maven Project**

**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/POM/4.0.0">

<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-core</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.10</version>

</dependency>

<!-- Add other dependencies as needed -->

</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: Configuring the Spring IoC Container**

**applicationContext.xml**

<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.java**

package com.library.service;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Other service methods

}

**LibraryManagementApplication.java**

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");

// Test the configuration

}

}

**Exercise 6: Configuring Beans with Annotations**

**applicationContext.xml**

<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">

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

</beans>

**BookService.java**

package com.library.service;

import org.springframework.stereotype.Service;

@Service

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Other service methods

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

// Repository methods

}

**LibraryManagementApplication.java**

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.class);

// Test the annotation-based configuration

}

}

**Exercise 7: Implementing Constructor and Setter Injection**

**applicationContext.xml**

<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">

<constructor-arg ref="bookRepository"/>

</bean>

</beans>

**BookService.java**

package com.library.service;

public class BookService {

private BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Other service methods

}

**LibraryManagementApplication.java**

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");

// Test constructor and setter injection

}

}

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

**pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.10</version>

</dependency>

</dependencies>

**applicationContext.xml**

<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"/>

</beans>

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.springframework.stereotype.Component;

@Aspect

@Component

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;

}

}

**LibraryManagementApplication.java**

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");

// Test AOP logging

}

}

**Exercise 9: Creating a Spring Boot Application**

**pom.xml**

<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>

<scope>runtime</scope>

</dependency>

</dependencies>

**application.properties**

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

**Book.java**

package com.library.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// Getters and Setters

}

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookController.java**

java

Copy code

package com.library.controller;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

}

**LibraryManagementApplication.java**

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);

}

}