### **Exercise 1: Control Structures**

**Scenario 1: Apply Discount to Loan Interest Rates**

DECLARE  
 CURSOR c\_customers IS  
 SELECT CustomerID, DOB  
 FROM Customers;  
   
 v\_age NUMBER;  
BEGIN  
 FOR r\_customer IN c\_customers LOOP  
 v\_age := FLOOR(MONTHS\_BETWEEN(SYSDATE, r\_customer.DOB) / 12);  
   
 IF v\_age > 60 THEN  
 UPDATE Loans  
 SET InterestRate = InterestRate - 1  
 WHERE CustomerID = r\_customer.CustomerID;  
 END IF;  
 END LOOP;  
   
 COMMIT;  
END;  
/

**Scenario 2: Promote to VIP Status**

DECLARE  
 CURSOR c\_customers IS  
 SELECT CustomerID, Balance  
 FROM Customers;  
BEGIN  
 FOR r\_customer IN c\_customers LOOP  
 IF r\_customer.Balance > 10000 THEN  
 UPDATE Customers  
 SET IsVIP = TRUE  
 WHERE CustomerID = r\_customer.CustomerID;  
 END IF;  
 END LOOP;  
   
 COMMIT;  
END;  
/

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

DECLARE  
 CURSOR c\_loans IS  
 SELECT CustomerID, LoanID, EndDate  
 FROM Loans  
 WHERE EndDate BETWEEN SYSDATE AND SYSDATE + 30;  
BEGIN  
 FOR r\_loan IN c\_loans LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Reminder: Loan ID ' || r\_loan.LoanID ||   
 ' for Customer ' || r\_loan.CustomerID ||  
 ' is due on ' || TO\_CHAR(r\_loan.EndDate, 'YYYY-MM-DD'));  
 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  
) IS  
 v\_balance NUMBER;  
BEGIN  
 -- Check balance of the source account  
 SELECT Balance INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account;  
   
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');  
 END IF;  
   
 -- Perform the transfer  
 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 OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;  
/

**Scenario 2: Update Employee Salary**

CREATE OR REPLACE PROCEDURE UpdateSalary(  
 p\_employee\_id IN NUMBER,  
 p\_percentage IN NUMBER  
) IS  
BEGIN  
 UPDATE Employees  
 SET Salary = Salary \* (1 + p\_percentage / 100)  
 WHERE EmployeeID = p\_employee\_id;  
   
 IF SQL%ROWCOUNT = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Employee ID does not exist');  
 END IF;  
   
 COMMIT;  
EXCEPTION  
 WHEN OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;  
/

**Scenario 3: Add New Customer**

CREATE OR REPLACE PROCEDURE AddNewCustomer(  
 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);  
   
EXCEPTION  
 WHEN DUP\_VAL\_ON\_INDEX THEN  
 DBMS\_OUTPUT.PUT\_LINE('Error: Customer ID already exists');  
 ROLLBACK;  
END;  
/

### **Exercise 3: Stored Procedures**

**Scenario 1: Process Monthly Interest**

CREATE OR REPLACE PROCEDURE ProcessMonthlyInterest IS  
BEGIN  
 UPDATE Accounts  
 SET Balance = Balance \* 1.01; -- Applying 1% interest  
   
 COMMIT;  
END;  
/

**Scenario 2: Update Employee Bonus**

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;  
/

**Scenario 3: Transfer Funds Between Accounts**

CREATE OR REPLACE PROCEDURE TransferFunds(  
 p\_from\_account IN NUMBER,  
 p\_to\_account IN NUMBER,  
 p\_amount IN NUMBER  
) IS  
 v\_balance NUMBER;  
BEGIN  
 -- Check balance of the source account  
 SELECT Balance INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = p\_from\_account;  
   
 IF v\_balance < p\_amount THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Insufficient funds');  
 END IF;  
   
 -- Perform the transfer  
 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 OTHERS THEN  
 ROLLBACK;  
 DBMS\_OUTPUT.PUT\_LINE('Error: ' || SQLERRM);  
END;  
/

### **Exercise 4: Functions**

**Scenario 1: Calculate Age**

CREATE OR REPLACE FUNCTION CalculateAge(  
 p\_dob DATE  
) RETURN NUMBER IS  
BEGIN  
 RETURN FLOOR(MONTHS\_BETWEEN(SYSDATE, p\_dob) / 12);  
END;  
/

**Scenario 2: Calculate Monthly Installment**

CREATE OR REPLACE FUNCTION CalculateMonthlyInstallment(  
 p\_loan\_amount NUMBER,  
 p\_interest\_rate NUMBER,  
 p\_duration\_years NUMBER  
) RETURN NUMBER IS  
 v\_monthly\_interest\_rate NUMBER;  
 v\_number\_of\_payments NUMBER;  
 v\_installment NUMBER;  
BEGIN  
 v\_monthly\_interest\_rate := p\_interest\_rate / 12 / 100;  
 v\_number\_of\_payments := p\_duration\_years \* 12;  
   
 v\_installment := (p\_loan\_amount \* v\_monthly\_interest\_rate) /   
 (1 - POWER(1 + v\_monthly\_interest\_rate, -v\_number\_of\_payments));  
   
 RETURN v\_installment;  
END;  
/

**Scenario 3: Check Sufficient Balance**

CREATE OR REPLACE FUNCTION HasSufficientBalance(  
 p\_account\_id IN NUMBER,  
 p\_amount IN 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: Update Last Modified Date**

CREATE OR REPLACE TRIGGER UpdateCustomerLastModified  
BEFORE UPDATE ON Customers  
FOR EACH ROW  
BEGIN  
 :NEW.LastModified := SYSDATE;  
END;  
/

**Scenario 2: Log Transactions**

CREATE OR REPLACE TRIGGER LogTransaction  
AFTER INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 INSERT INTO AuditLog (TransactionID, AccountID, TransactionDate, Amount, TransactionType, LogDate)  
 VALUES (:NEW.TransactionID, :NEW.AccountID, :NEW.TransactionDate, :NEW.Amount, :NEW.TransactionType, SYSDATE);  
END;  
/

**Scenario 3: Check Transaction Rules**

CREATE OR REPLACE TRIGGER CheckTransactionRules  
BEFORE INSERT ON Transactions  
FOR EACH ROW  
BEGIN  
 IF :NEW.TransactionType = 'Withdrawal' THEN  
 DECLARE  
 v\_balance NUMBER;  
 BEGIN  
 SELECT Balance INTO v\_balance  
 FROM Accounts  
 WHERE AccountID = :NEW.AccountID;  
   
 IF v\_balance < :NEW.Amount THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Insufficient balance for withdrawal');  
 END IF;  
 END;  
 ELSIF :NEW.TransactionType = 'Deposit' THEN  
 IF :NEW.Amount <= 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20003, 'Deposit amount must be positive');  
 END IF;  
 END IF;  
END;  
/

### **Exercise 6: Cursors**

**Scenario 1: Generate Monthly Statements**

DECLARE  
 CURSOR c\_transactions IS  
 SELECT t.AccountID, a.CustomerID, t.TransactionDate, t.Amount, t.TransactionType  
 FROM Transactions t  
 JOIN Accounts a ON t.AccountID = a.AccountID  
 WHERE t.TransactionDate BETWEEN TRUNC(SYSDATE, 'MONTH') AND SYSDATE;  
BEGIN  
 FOR r\_transaction IN c\_transactions LOOP  
 DBMS\_OUTPUT.PUT\_LINE('Statement for Customer ' || r\_transaction.CustomerID ||  
 ': Account ' || r\_transaction.AccountID ||  
 ', Date: ' || TO\_CHAR(r\_transaction.TransactionDate, 'YYYY-MM-DD') ||  
 ', Amount: ' || r\_transaction.Amount ||  
 ', Type: ' || r\_transaction.TransactionType);  
 END LOOP;  
END;  
/

**Scenario 2: Apply Annual Fee**

DECLARE  
 CURSOR c\_accounts IS  
 SELECT AccountID, Balance  
 FROM Accounts;  
BEGIN  
 FOR r\_account IN

### **WEEK 2 ASISGNMENT 2nd PART**

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

#### **1. Set Up a Spring 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>  
 <packaging>jar</packaging>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-core</artifactId>  
 <version>5.3.24</version>  
 </dependency>  
 <!-- Add other dependencies as needed -->  
 </dependencies>  
</project>

#### **2. Configure the Application Context**

**src/main/resources/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 definition for BookRepository -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <!-- Bean definition for BookService -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

#### **3. Define Service and Repository Classes**

**com/library/service/BookService.java**:

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // Setter for BookRepository  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 // Business methods  
 public void manageBooks() {  
 // Implementation here  
 }  
}

**com/library/repository/BookRepository.java**:

package com.library.repository;  
  
public class BookRepository {  
 // Data access methods  
}

#### **4. Run the Application**

**com/library/LibraryManagementApplication.java**:

package com.library;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import com.library.service.BookService;  
  
public class LibraryManagementApplication {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.manageBooks();  
 }  
}

### **Exercise 2: Implementing Dependency Injection**

#### **1. Modify the XML Configuration**

Already configured in **applicationContext.xml** from Exercise 1. It wires BookRepository into BookService.

#### **2. Update the BookService Class**

Already done in **BookService.java** from Exercise 1. Ensure BookService has a setter for BookRepository.

#### **3. Test the Configuration**

Run **LibraryManagementApplication.java** to verify that BookService is correctly configured with BookRepository.

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

#### **1. Add Spring AOP Dependency**

**pom.xml**:

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aspects</artifactId>  
 <version>5.3.24</version>  
</dependency>

#### **2. Create an Aspect for Logging**

**com/library/aspect/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;  
 }  
}

#### **3. Enable AspectJ Support**

**src/main/resources/applicationContext.xml**:

<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">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
   
 <aop:aspectj-autoproxy/>  
 <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>  
</beans>

#### **4. Test the Aspect**

Run **LibraryManagementApplication.java** and check the console for log messages indicating method execution times.

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

#### **1. Create a New Maven Project**

Already done in **Exercise 1**.

#### **2. Add Spring Dependencies in pom.xml**

Already done in **Exercise 1**.

#### **3. Configure Maven Plugins**

**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>

### **Exercise 5: Configuring the Spring IoC Container**

Already covered in **Exercise 1**. Configuration details in **applicationContext.xml**.

### **Exercise 6: Configuring Beans with Annotations**

#### **1. Enable Component Scanning**

**src/main/resources/applicationContext.xml**:

<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"/>  
 <aop:aspectj-autoproxy/>  
</beans>

#### **2. Annotate Classes**

**com/library/service/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;  
 }  
  
 public void manageBooks() {  
 // Implementation here  
 }  
}

**com/library/repository/BookRepository.java**:

package com.library.repository;  
  
import org.springframework.stereotype.Repository;  
  
@Repository  
public class BookRepository {  
 // Data access methods  
}

#### **3. Test the Configuration**

Run **LibraryManagementApplication.java** to verify that annotation-based configuration is working.

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

#### **1. Configure Constructor Injection**

**src/main/resources/applicationContext.xml**:

<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"/>  
 <aop:aspectj-autoproxy/>  
  
 <bean id="bookService" class="com.library.service.BookService">  
 <constructor-arg ref="bookRepository"/>  
 </bean>  
</beans>

#### **2. Configure Setter Injection**

**com/library/service/BookService.java**:

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // Constructor-based injection  
 public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 // Setter for BookRepository  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void manageBooks() {  
 // Implementation here  
 }  
}

#### **3. Test the Injection**

Run **LibraryManagementApplication.java** to verify that both constructor and setter injection are working.

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

#### **1. Define an Aspect**

Already done in **Exercise 3**.

#### **2. Create Advice Methods**

Already done in **LoggingAspect.java**.

#### **3. Configure the Aspect**

Already covered in **Exercise 3**.

#### **4. Test the Aspect**

Run **LibraryManagementApplication.java** and verify that logging information is printed to the console.

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

#### **1. Create a Spring Boot Project**

Use Spring Initializr (<https://start.spring.io/>) to create a new Spring Boot project with the following settings:

* **Project:** Maven
* **Language:** Java
* **Spring Boot:** 3.x.x (latest stable)
* **Group:** com.library
* **Artifact:** LibraryManagement
* **Dependencies:** Spring Web, Spring Data JPA, H2 Database

#### **2. Add Dependencies**

Dependencies will be added automatically by Spring Initializr. The pom.xml will include:

<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>org.h2</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
</dependency>

#### **3. Create Application Properties**

**src/main/resources/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  
spring.h2.console.enabled=true

#### **4. Define Entities and Repositories**

**com/library/entity/Book.java**:

package com.library.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;  
  
 // Getters and setters  
}

**com/library/repository/BookRepository.java**:

package com.library.repository;  
  
import com.library.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BookRepository extends JpaRepository<Book, Long> {  
}

#### **5. Create a REST Controller**

**com/library/controller/BookController.java**:

package com.library.controller;  
  
import com.library.entity.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 addBook(@RequestBody Book book) {  
 return bookRepository.save(book);  
 }  
  
 @GetMapping("/{id}")  
 public Book getBookById(@PathVariable Long id) {  
 return bookRepository.findById(id).orElse(null);  
 }  
  
 @PutMapping("/{id}")  
 public Book updateBook(@PathVariable Long id, @RequestBody Book book) {  
 book.setId(id);  
 return bookRepository.save(book);  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable Long id) {  
 bookRepository.deleteById(id);  
 }  
}

#### **6. Run the Application**

**com/library/LibraryManagementApplication.java**:

package com.library;  
  
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);  
 }  
}