## **Spring Core and Maven Exercises**

### **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.

#### **Code:**

**MainApp.java**

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

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String bookName) {  
 System.out.println("Book '" + bookName + "' saved to the database.");  
 }  
}

**BookService.java**

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 bookName) {  
 System.out.println("Processing the book: " + bookName);  
 bookRepository.saveBook(bookName);  
 }  
}

### **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.

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String bookName) {  
 System.out.println("Book '" + bookName + "' saved to the database.");  
 }  
}

**BookService.java**

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 bookName) {  
 System.out.println("Processing the book: " + bookName);  
 bookRepository.saveBook(bookName);  
 }  
}

### **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.

**MainApp.java**

package com.example;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class MainApp {  
 public static void main(String[] args) {  
 System.out.println("Starting Spring Application...");  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 System.out.println("Spring context loaded successfully!");  
  
 MessageService service = (MessageService) context.getBean("messageService");  
 System.out.println("Retrieved bean: " + service.getClass().getSimpleName());  
 service.printMessage();  
 }  
}

**MessageService.java**

package com.example;  
  
public class MessageService {  
 public void printMessage() {  
 System.out.println("Hello from MessageService! Spring is working!");  
 }  
}

### **Spring Data JPA**

### **Difference between JPA, Hibernate, and Spring Data JPA**

#### **1. JPA (Java Persistence API)**

* It is a **specification** provided by Java for ORM (Object-Relational Mapping).
* JPA provides **standard APIs** for managing relational data in Java applications.
* **Does not provide implementation**, only guidelines.
* Needs a provider like **Hibernate** or **EclipseLink** to work.
* Focuses on **entity mapping**, **query language (JPQL)**, and **transactions**.
* Example annotations: @Entity, @Id, @GeneratedValue

#### **2. Hibernate**

* A popular **JPA implementation** and powerful ORM framework.
* Provides all features required by JPA **plus extras** like:
  + **Caching**
  + **Lazy loading**
  + **Batch processing**
* Supports both **JPA and native Hibernate APIs** (like Session).
* Has its own query language called **HQL** (Hibernate Query Language).
* Can be used **with or without Spring**.

#### **3. Spring Data JPA**

* A Spring project that **simplifies JPA** usage in Spring apps.
* **Builds on top of JPA and Hibernate**.
* Reduces boilerplate code by providing **pre-built repositories** like JpaRepository.
* Supports **query method names**, **custom JPQL**, and @Query annotations.
* Auto-implements **CRUD operations** and supports **pagination and sorting**.
* Ideal for **rapid development** of data access layers.