WEEK 3:  
MODEULE 1: Spring Core Maven

**Exercise 1: Configuring a Basic Spring Application**  
  
package com.example;

public class BookRepository {

public BookRepository() {

System.out.println("This is BookRepository");

}

}  
  
package com.example;

public class BookService {

public BookService() {

System.out.println("This is BookService");

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 2: Implementing Dependency Injection**  
  
package com.example;

public class BookRepository {

public BookRepository() {

System.out.println("BookRepository Created");

}

@Override

public String toString(){

return "BookRepository Injected";

}

}  
package com.example;

import com.example.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

System.out.println("Application Started");

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

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

service.printBookRepository();

}

}

A screenshot of a computer program

AI-generated content may be incorrect.  
  
**Exercise 4: Creating and Configuring a Maven Project**  
  
package com.example;

import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

public class AppTest

extends TestCase

{

public AppTest( String testName )

{

super( testName );

}

public static Test suite()

{

return new TestSuite( AppTest.class );

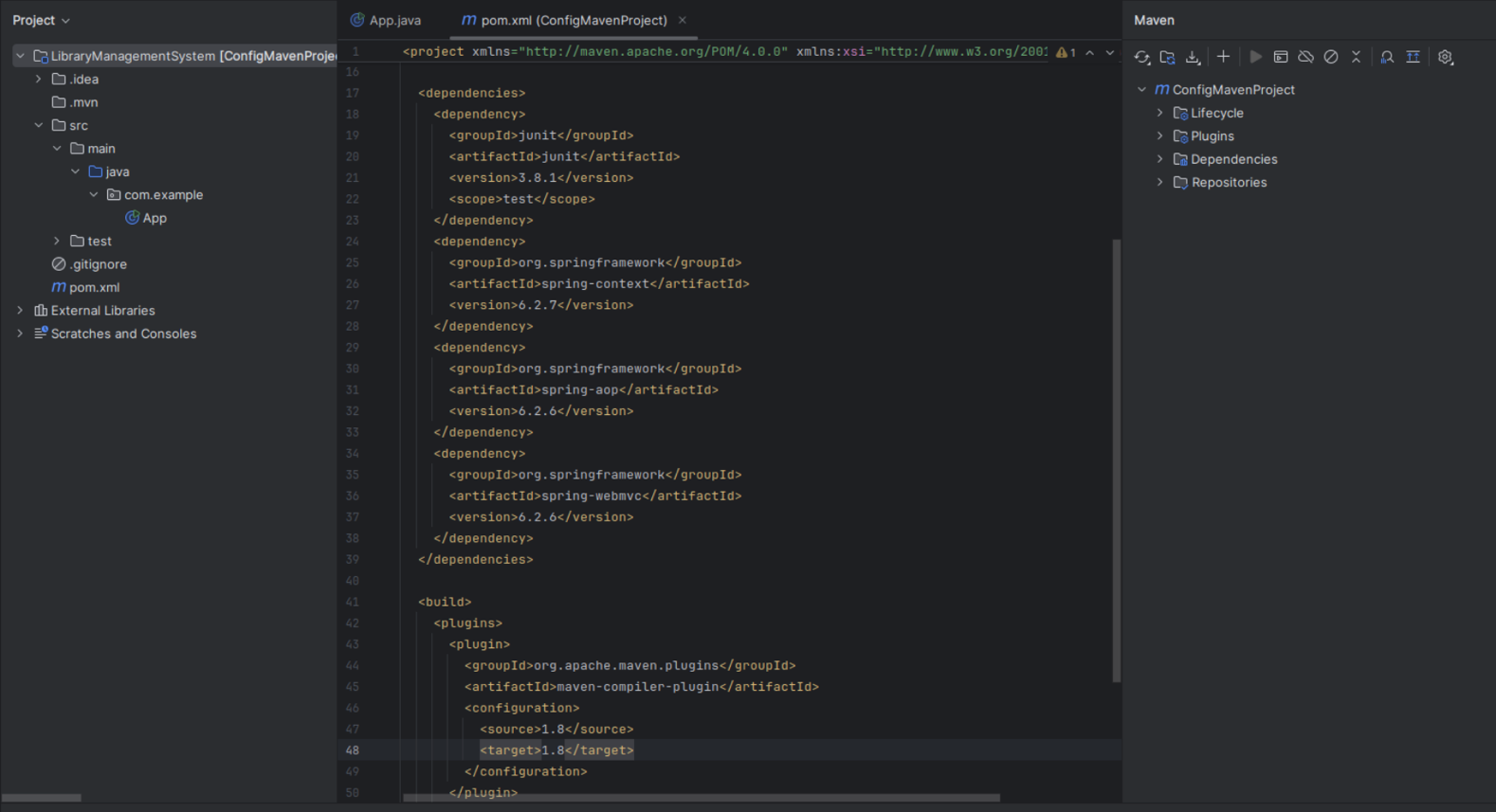
}

public void testApp()

{

assertTrue( true );

}

}  
  
  
**Exercise 5: Configuring the Spring IoC Container**  
  
package com.example;

import java.util.Arrays;

import java.util.List;

public class BookRepository {

public BookRepository() {

System.out.println("BookRepository Created");

}

public List<String> getBooks() {

return Arrays.asList("Java Books", "Spring Books", "Mern Books");

}

}

A computer screen shot of a program

AI-generated content may be incorrect.  
**Exercise 7: Implementing Constructor and Setter Injection**  
  
package com.example;

import com.example.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args )

{

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

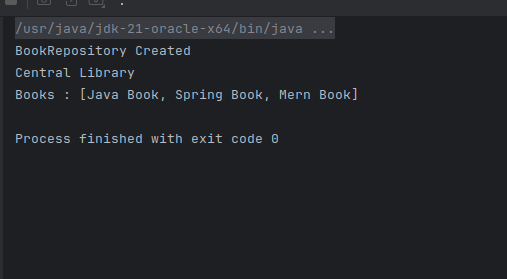
BookService bookService = context.getBean(BookService.class);

System.out.println(bookService.getLibraryName());

bookService.getBooks();

}

}



**Exercise 9: Creating a Spring Boot Application**  
A screenshot of a computer

AI-generated content may be incorrect.  
A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

MODULE 2: Spring Data JPA with Spring Boot, Hibernate

**Spring Data JPA - Quick Example**

A close up of a text

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Difference Between JPA, Hibernate, and Spring Data JPA**

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

JPA is just a **specification** (a set of rules) for working with databases using Java objects.

It defines key concepts like:

@Entity – marks a class as a database table.

@Id – marks the primary key.

EntityManager – used to interact with the database.

JPA **does not contain actual code**—you need a library (like Hibernate) to use it.

**2) Hibernate**

Hibernate is a **framework** that **implements JPA** and adds extra features.

It provides the real functionality to connect Java objects with database tables.

Hibernate also includes helpful features like:

Caching

Lazy loading

Query optimization

If you use Hibernate directly, you manage sessions and transactions yourself (unless you use Spring).

**3) Spring Data JPA**

Spring Data JPA is a **part of the Spring Framework**.

It builds **on top of JPA** to make working with databases easier.

You just define **interfaces** (like EmployeeRepository), and Spring handles the database code for you.

It works smoothly with Spring Boot and handles transactions automatically.

**Code Comparison  
  
Using Hibernate (Manual Handling)**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

} finally {

session.close();

}

return employeeID;

}

**Using Spring Data JPA**

**Employee Repository**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**Employee Service**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Implement services for managing Country**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Add a new Country

A screenshot of a computer

AI-generated content may be incorrect.