# Spring Framework Assignment – Digital Library Backend System

## Task 1: Setting Up a Spring Project Using Maven

Use Case: You are building a backend for a digital library system. To begin, create a Maven project named 'DigitalLibrarySystem'. Include essential Spring Core libraries for bean configuration and dependency management.

Step 1: Create the Maven Project

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.21</version>  
 </dependency>  
</dependencies>

Step 2: Create Configuration File

Inside src/main/resources, add a file named beanConfig.xml with the following content:

<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="catalogRepository" class="org.library.data.CatalogRepository"/>  
 <bean id="catalogService" class="org.library.business.CatalogService">  
 <property name="catalogRepository" ref="catalogRepository"/>  
 </bean>  
</beans>

Step 3: Create Repository and Service Classes

CatalogRepository.java

package org.library.data;  
  
public class CatalogRepository {  
 public void fetchCatalog() {  
 System.out.println("CatalogRepository: Retrieving book catalog...");  
 }  
}

CatalogService.java

package org.library.business;  
  
import org.library.data.CatalogRepository;  
  
public class CatalogService {  
 private CatalogRepository catalogRepository;  
  
 public void setCatalogRepository(CatalogRepository catalogRepository) {  
 this.catalogRepository = catalogRepository;  
 }  
  
 public void processCatalog() {  
 System.out.println("CatalogService: Processing catalog service...");  
 catalogRepository.fetchCatalog();  
 }  
}

Step 4: Execute and Verify

DigitalLibraryMain.java

import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
import org.library.business.CatalogService;  
  
public class DigitalLibraryMain {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("beanConfig.xml");  
 CatalogService service = (CatalogService) context.getBean("catalogService");  
 service.processCatalog();  
 }  
}

## Task 2: Managing Component Dependencies via Spring IoC

Update the XML configuration and ensure that CatalogService receives its dependency (CatalogRepository) using setter-based DI.

Updated CatalogService.java with Setter Injection:

package org.library.business;  
  
import org.library.data.CatalogRepository;  
  
public class CatalogService {  
 private CatalogRepository catalogRepository;  
  
 public void setCatalogRepository(CatalogRepository catalogRepository) {  
 this.catalogRepository = catalogRepository;  
 }  
  
 public void executeService() {  
 System.out.println("CatalogService: Executing service logic...");  
 catalogRepository.fetchCatalog();  
 }  
}

## Task 3: Configuring Maven Plugins and Multiple Spring Modules

Add the following to your pom.xml to include other Spring modules and set Java version:

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.21</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.21</version>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.21</version>  
 </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>