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

**1. Set Up a Spring Project:**

**->Create a Maven project named LibraryManagement.**

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement \

-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

cd LibraryManagement

### ->**Add Spring Core dependencies in the pom.xml file.**

<dependencies>  
 <!-- Spring Core Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.36</version>  
 </dependency>  
</dependencies>  
  
<build>  
 <plugins>  
 <!-- Allows mvn exec:java -->  
 <plugin>  
 <groupId>org.codehaus.mojo</groupId>  
 <artifactId>exec-maven-plugin</artifactId>  
 <version>3.1.0</version>  
 <configuration>  
 <mainClass>com.library.LibraryApp</mainClass>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

**2. Configure the Application Context:**

### **Create Required Classes**

3. Define Service and Repository Classes:

package com.library.repository;  
  
public class BookRepository {  
 public void getBooks() {  
 System.out.println("Fetching books from the repository...");  
 }  
}

#### **BookService.java**

java

CopyEdit

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 showBooks() {  
 bookRepository.getBooks();  
 }  
}

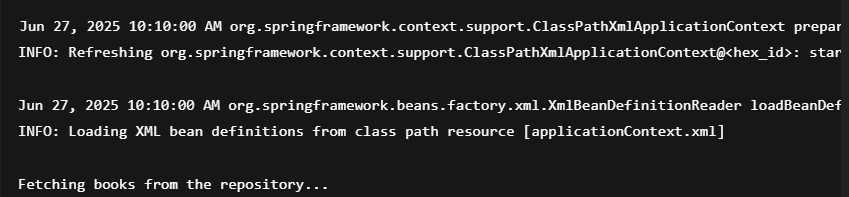
### **XML Configuration**

<?xml version="1.0" encoding="UTF-8"?>  
<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>

### **Main Class**

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

* **4. Run the Application:**



**Exercise 2: Implementing Dependency Injection**

1. Modify the XML Configuration:

**Update applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<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>">  
  
 <!-- Define BookRepository bean -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository" />  
  
 <!-- Define BookService bean and inject BookRepository using setter -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

**2. Update the BookService Class:**

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // Setter for Dependency Injection  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void showBooks() {  
 bookRepository.getBooks(); // Calls method from repository  
 }  
}

### **Verify BookRepository Class**

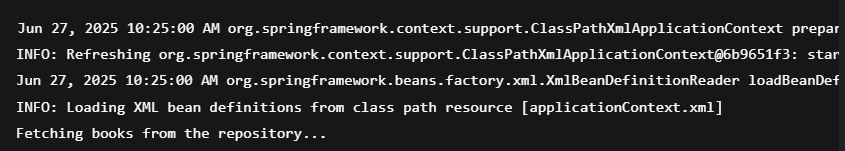
🗂package com.library.repository;  
  
public class BookRepository {  
 public void getBooks() {  
 System.out.println("Fetching books from the repository...");  
 }  
}

**3. Test the Configuration:Run Main Class**

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

mvn clean compile

mvn exec:java



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

1. Create a New Maven Project:

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2. Add Spring Dependencies in 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/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.31</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.31</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.31</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>

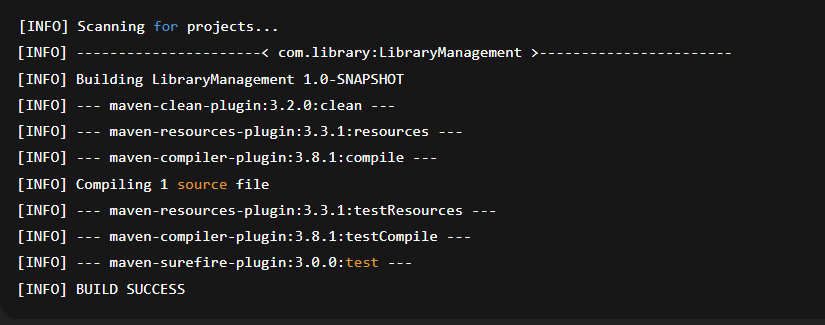
</configuration>

</plugin>

</plugins>

</build>

</project>



**Spring Data JPA - Quick Example**

### **MySQL Setup**

CREATE SCHEMA ormlearn;  
  
USE ormlearn;  
  
CREATE TABLE country (  
 code VARCHAR(2) PRIMARY KEY,  
 name VARCHAR(50)  
);  
  
INSERT INTO country VALUES ('IN', 'India');  
INSERT INTO country VALUES ('US', 'United States of America');

### **Update application.properties**

In src/main/resources/application.properties:

properties

CopyEdit

# Logging  
logging.level.org.springframework=info  
logging.level.com.cognizant=debug  
logging.level.org.hibernate.SQL=trace  
logging.level.org.hibernate.type.descriptor.sql=trace  
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n  
  
# DB Config  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn  
spring.datasource.username=root  
spring.datasource.password=root  
  
# Hibernate  
spring.jpa.hibernate.ddl-auto=validate  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

### **Create Entity Class**

package com.cognizant.ormlearn.model;  
  
import javax.persistence.\*;  
  
@Entity  
@Table(name = "country")  
public class Country {  
  
 @Id  
 @Column(name = "code")  
 private String code;  
  
 @Column(name = "name")  
 private String name;  
  
 public String getCode() { return code; }  
 public void setCode(String code) { this.code = code; }  
  
 public String getName() { return name; }  
 public void setName(String name) { this.name = name; }  
  
 @Override  
 public String toString() {  
 return "Country [code=" + code + ", name=" + name + "]";  
 }  
}

### **Create Repository**

package com.cognizant.ormlearn.repository;  
  
import com.cognizant.ormlearn.model.Country;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository  
public interface CountryRepository extends JpaRepository<Country, String> {  
}

### **Create Service**

package com.cognizant.ormlearn.service;  
  
import com.cognizant.ormlearn.model.Country;  
import com.cognizant.ormlearn.repository.CountryRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
import java.util.List;  
  
@Service  
public class CountryService {  
  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

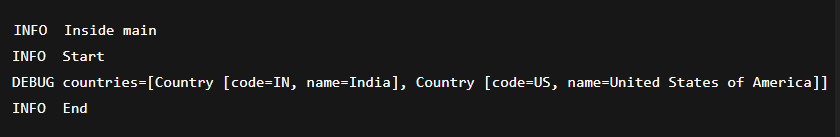
### **Update Main Class**

OrmLearnApplication.java:

java

CopyEdit

package com.cognizant.ormlearn;  
  
import com.cognizant.ormlearn.model.Country;  
import com.cognizant.ormlearn.service.CountryService;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
  
import java.util.List;  
  
@SpringBootApplication  
public class OrmLearnApplication {  
  
 private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
 private static CountryService countryService;  
  
 public static void main(String[] args) {  
 ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
 countryService = context.getBean(CountryService.class);  
 testGetAllCountries();  
 }  
  
 private static void testGetAllCountries() {  
 LOGGER.info("Start");  
 List<Country> countries = countryService.getAllCountries();  
 LOGGER.debug("countries={}", countries);  
 LOGGER.info("End");  
 }  
}



**Difference between JPA, Hibernate and Spring Data JPA**   
   
**Java Persistence API (JPA)**

* JPA is a **Java Specification (JSR 338)** for data persistence.
* It defines standard interfaces and annotations for **mapping Java objects to relational databases**.
* **Does not contain an implementation**.
* You still need an implementation like **Hibernate**, **EclipseLink**, or **OpenJPA** to use it.

### **Hibernate**

* Hibernate is a **popular ORM (Object-Relational Mapping)** tool.
* It is **an implementation of JPA**.
* It adds additional features **beyond the JPA specification**.
* Provides APIs like Session, Transaction for direct control.

### **Spring Data JPA**

* Built on top of JPA (and usually uses Hibernate as default).
* **Does not implement JPA** but provides **abstraction and helper APIs**.
* **Reduces boilerplate** (like writing DAO code).
* Manages transaction and entity management automatically.
* You can perform queries with just interfaces and annotations.

## **Code Comparison: Hibernate vs Spring Data JPA**

### **Using Hibernate (manual session & transaction handling):**

java

CopyEdit

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 (HibernateException e) {  
 if (tx != null) tx.rollback();  
 e.printStackTrace();  
 } finally {  
 session.close();  
 }  
  
 return employeeID;  
}

### **Using Spring Data JPA (less code, built-in abstraction):**

**EmployeeRepository.java**

java

CopyEdit

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
}

**EmployeeService.java**

java

CopyEdit

@Service  
public class EmployeeService {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}