**Week 3**

**Mandatory Hands-On**

**-Abinaya B(Superset ID: 6388062)**

**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.

**Steps:**

1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
3. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.
   * Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**SOLUTION:**

**Step 1:**

1. **Create Maven Project**

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

A screenshot of a computer

AI-generated content may be incorrect.

1. **pom.xml – Add Spring Core Dependency**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

1. **Step 2: Configure the Application Context**

**Create 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

https://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>

A screenshot of a computer program

AI-generated content may be incorrect.

1. **Step 3: Define Service and Repository Classes**
2. **BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Book saved: " + title);

}

}

1. **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 title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

A screenshot of a computer

AI-generated content may be incorrect.

**Step 4: Run the Application**

**Main Class**

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");

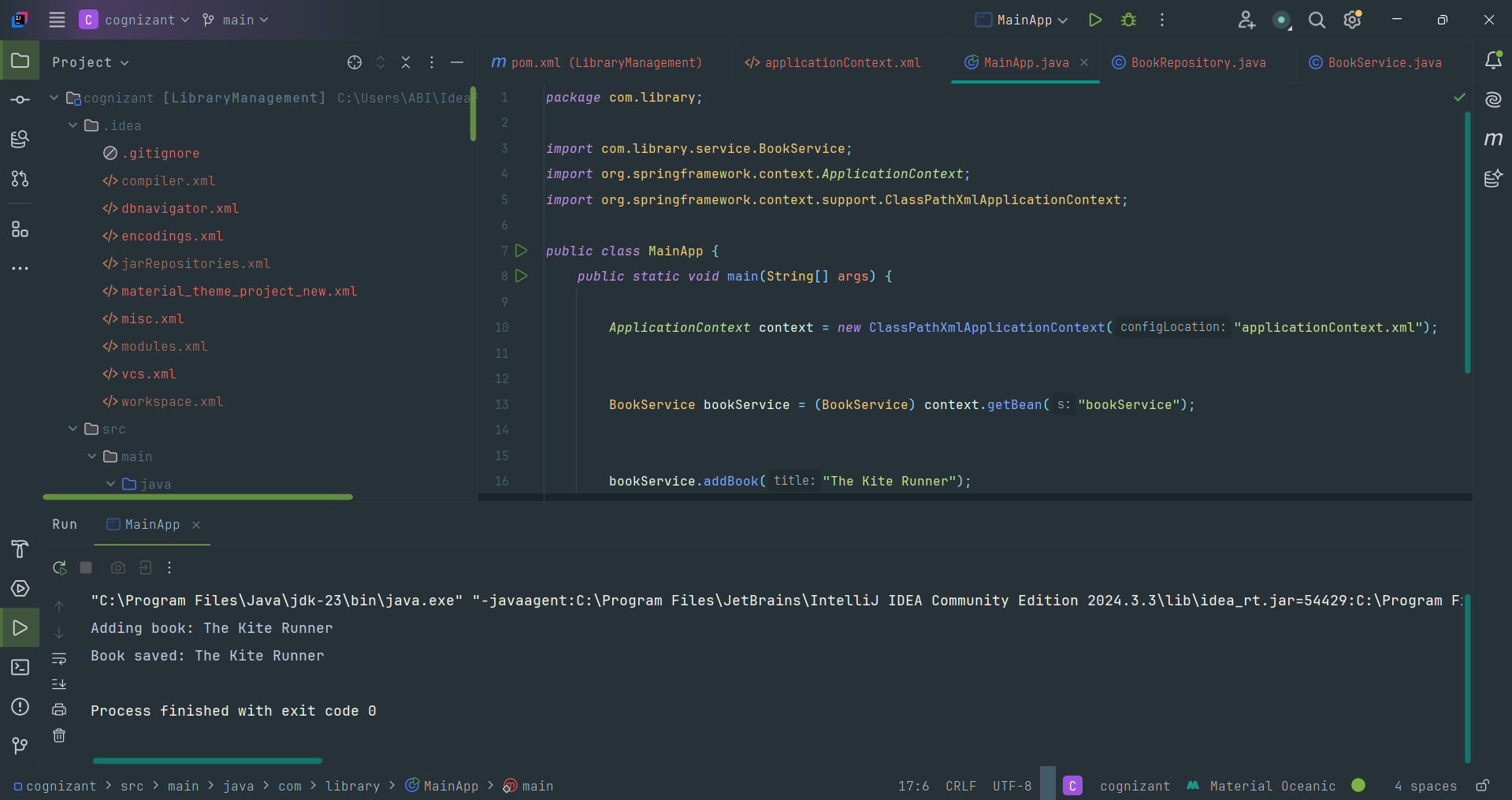
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Kite Runner");

}

}

**OUTPUT:**



**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.

**Steps:**

1. **Modify the XML Configuration:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

**SOLUTION:**

**Step 1: Modify applicationContext.xml**

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

A screen shot of a computer

AI-generated content may be incorrect.

**Step 2: Update 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 title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**Step 3: Test It Using 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");

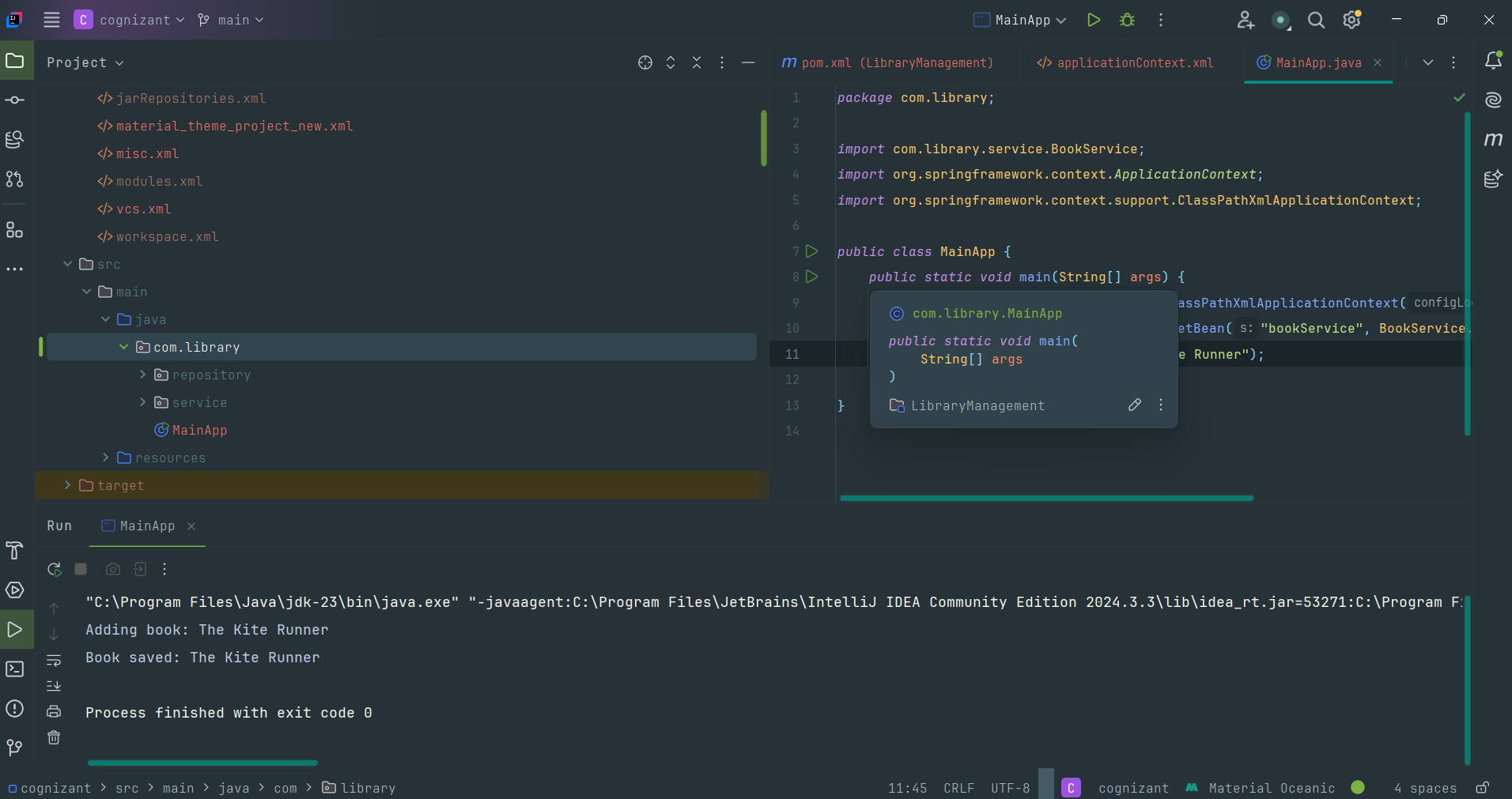
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Kite Runner ");

}

}

**OUTPUT:**



**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.

**Steps:**

1. **Create a New Maven Project:**
   * Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**SOLUTION:**

**Step 1: Create a New Maven Project**

LibraryManagement/

├── pom.xml

├── src/

│ ├── main/

│ │ ├── java/

│ │ └── resources/

│ └── test/

│ └── java/

A screenshot of a computer program

AI-generated content may be incorrect.

**Step 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>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</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>

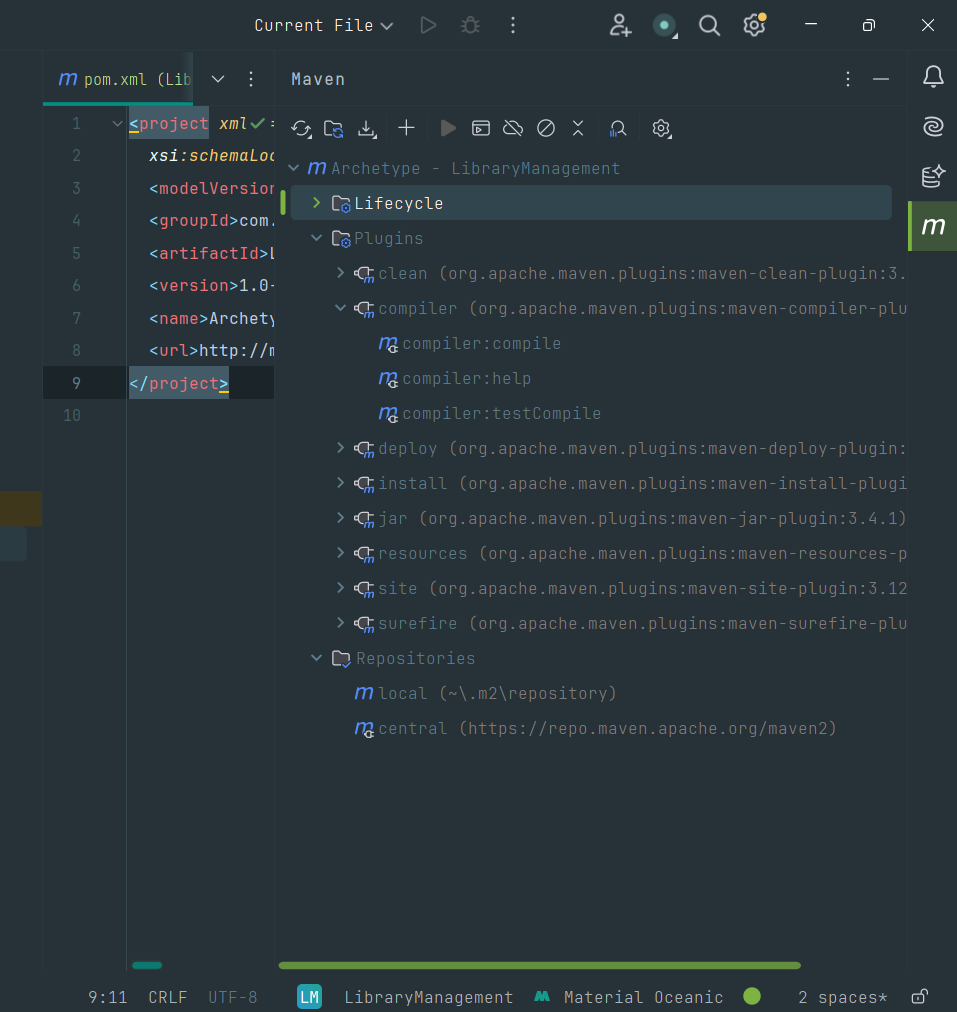
</build>

</project>

**Step 3: Reload the Maven Project**

* We will get the following after reloading:
* A fully configured Maven project
* Spring Context (for beans and DI)
* Spring AOP (for aspect programming if needed later)
* Spring Web MVC (to add a controller layer or web app)
* Maven Compiler Plugin for Java 1.8

**PLUGINS:**



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

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**

Create a main class to load the Spring context and test the configuration.

**SOLUTION :**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.

<?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>

1. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.

**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("Adding book: " + bookName);

bookRepository.save(bookName);

}

}

**BookRepository.java;**

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

System.out.println("Saving book: " + bookName);

}

}

**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("Pride and Prejudice");

}

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

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

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**SOLUTION:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.

**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">

<bean id="bookRepository" class="com.library.repository.BookRepository" />

<bean id="bookService" class="com.library.service.BookService">

<constructor-arg value="LibraryService V1" />

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

1. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

public BookService(String serviceName) {

this.serviceName = serviceName;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println("[" + serviceName + "] Adding book: " + bookName);

bookRepository.save(bookName);

}

}

**BookRepository.java:**

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

System.out.println("Saving book: " + bookName);

}

}

**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 Midnight Library");  
 }  
}

1. **Test the Injection:**

Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Spring Data JPA with Spring Boot, Hibernate**

**Hands on 1 - Spring Data JPA - Quick Example**   
**Project Setup Using Spring Initializr**

1. Open <https://start.spring.io/>
2. Configure the following:
   * **Group**: com.cognizant
   * **Artifact**: orm-learn
   * **Description**: Demo project for Spring Data JPA and Hibernate
3. Add dependencies:
   * Spring Boot DevTools
   * Spring Data JPA
   * MySQL Driver
4. Click **Generate** to download the project ZIP.
5. Extract the ZIP into your Eclipse workspace directory.
6. Import in Eclipse:
   * File > Import > Maven > Existing Maven Projects > Browse (select extracted folder) > Finish

**MySQL Setup**

1. Open MySQL terminal: mysql -u root -p
2. Create schema:create schema ormlearn;

**application.properties:**

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

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

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

OrmLearnApplication.java:

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

}

**Country Table Creation:**

create table country(

co\_code varchar(2) primary key,

co\_name varchar(50)

);

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Country.java:**

@Entity

@Table(name="country")

public class Country {

@Id

@Column(name="code")

private String code;

@Column(name="name")

private String name;

}

**CountryRepository.java:**

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

CountryService.java:

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java:**

@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");

}

}

**OUTPUT:**

A black screen with white text

AI-generated content may be incorrect.

**Hands on 4 - Difference between JPA, Hibernate and Spring Data JPA**

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

| **Feature** | **Description** |
| --- | --- |
| Type | **Specification (JSR 338)** |
| Purpose | Defines standard API for object-relational mapping (ORM) in Java |
| Implementation | Does **not** provide implementation—it’s just an API |
| Usage | Used through implementations like Hibernate, EclipseLink, etc. |
| Example | @Entity, @Id, @Table, etc. |

JPA is the blueprint. Think of it as a contract that says how Java objects should interact with relational databases.

**2. Hibernate**

| **Feature** | **Description** |
| --- | --- |
| Type | **ORM framework** |
| Purpose | Implements JPA for data persistence |
| Capabilities | Provides additional features beyond JPA (e.g., caching, lazy loading) |
| Responsibility | Manages database sessions, transactions, SQL generation, and mapping |

Hibernate is one of the most popular **implementations of JPA**.

**3. Spring Data JPA**

| **Feature** | **Description** |
| --- | --- |
| Type | **Abstraction layer over JPA** |
| Purpose | Reduces boilerplate code for data access |
| Implementation | Does **not** implement JPA; instead uses JPA provider like Hibernate |
| Features | Provides repository interfaces, automatic query generation, transaction management |
| Benefits | Minimal configuration, CRUD without writing SQL, better integration with Spring Boot |

Spring Data JPA allows developers to focus only on **business logic** without writing repetitive data access code.

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

**Hibernate Example**

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;

}

**What’s happening?**

* Manual session and transaction handling
* Explicit error handling and rollback
* Boilerplate code

**Spring Data JPA Example**

**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);

}

}

**What’s happening?**

* No session/transaction management needed
* save() handles both insert and update
* Minimal code with the same functionality

**Summary**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Framework | Abstraction/Framework |
| Provides | API only | JPA implementation + enhancements | Simplified data access using repositories |
| Boilerplate | Moderate | High | Low |
| Transaction Mgmt | Manual (via implementation) | Manual or automatic | Automatic |
| Common Usage | With Hibernate, EclipseLink | Standalone or via Spring | Inside Spring Boot Projects |

**Hands on 5-Implement services for managing Country**   
Populate country Table

**Clear existing data**

DELETE FROM country;

**Insert sample data**:

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(100)

);

**Find a Country by Code:**

public Country findCountryByCode(String code) {

return countryRepository.findById(code).orElseThrow(() -> new CountryNotFoundException("Country not found"));

}

**Add a New Country:**

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

**Update Country:**

@Transactional

public void updateCountry(Country country) {

if (!countryRepository.existsById(country.getCode())) {

throw new CountryNotFoundException("Country not found");

}

countryRepository.save(country);

}

**Delete Country:**

@Transactional

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

Find Countries by Partial Name:

**CountryRepository.java:**

List<Country> findByNameContainingIgnoreCase(String partialName);

CountryService.java:

public List<Country> searchCountriesByName(String partialName) {

return countryRepository.findByNameContainingIgnoreCase(partialName);

}

**OrmLearnApplication.java:**

private static void testFindCountryByCode() {

Country country = countryService.findCountryByCode("IN");

LOGGER.debug("Country: {}", country);

}

private static void testAddCountry() {

Country country = new Country();

country.setCode("ZZ");

country.setName("Zootopia");

countryService.addCountry(country);

}

private static void testUpdateCountry() {

Country country = new Country();

country.setCode("ZZ");

country.setName("Zootopia - Updated");

countryService.updateCountry(country);

}

private static void testDeleteCountry() {

countryService.deleteCountry("ZZ");

}

private static void testSearchByName() {

List<Country> countries = countryService.searchCountriesByName("an");

countries.forEach(c -> LOGGER.debug("Matched Country: {}", c));

}

**OUTPUT:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Hands on 6 - Find a country based on country code** :

**Create the Exception Class:**

package com.cognizant.springlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

**CountryService.java:**

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country code \"" + countryCode + "\" not found");

}

return result.get();

}

}

**OrmLearnApplication.java:**

@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);

testGetCountryByCode();

}

private static void testGetCountryByCode() {

LOGGER.info("Start");

try {

Country country = countryService.findCountryByCode("IN");

LOGGER.debug("Country: {}", country);

assert "India".equals(country.getName()) : "Country name mismatch";

} catch (CountryNotFoundException e) {

LOGGER.error("Exception: {}", e.getMessage());

}

LOGGER.info("End");

}

}

**OUTPUT:**

A black screen with white text

AI-generated content may be incorrect.

**Hands on 7-Add a new country** 

**CountryService.java:**

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country code \"" + countryCode + "\" not found");

}

return result.get();

}

}

**OrmLearnApplication.java**

private static void testAddCountry() {

LOGGER.info("Start");

Country newCountry = new Country();

newCountry.setCode("ZZ");

newCountry.setName("Zootopia");

countryService.addCountry(newCountry);

try {

Country fetchedCountry = countryService.findCountryByCode("ZZ");

LOGGER.debug("Added Country: {}", fetchedCountry);

assert "Zootopia".equals(fetchedCountry.getName()) : "Country name doesn't match!";

} catch (CountryNotFoundException e) {

LOGGER.error("Exception occurred while verifying new country: {}", e.getMessage());

}

LOGGER.info("End");

}

**main():**

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

testAddCountry();

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.