WEEK – 3

SPRING CORE AND MAVEN :

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 :

- **Set Up a Spring Project:**
  - Create a Maven project named **LibraryManagement**.
  - Add Spring Core dependencies in the **pom.xml** file.
- **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.
- **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**.
- **Run the Application:**
  - Create a main class to load the Spring context and test the configuration.

CODE :

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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.navya.library</groupId>
    <artifactId>LibraryManagement1</artifactId>
    <version>1.0-SNAPSHOT</version>

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

    <dependencies>
        <!-- Spring Context -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
```

```
        <version>5.3.32</version>
      </dependency>
   </dependencies>

   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.11.0</version>
            <configuration>
               <source>11</source>
               <target>11</target>
            </configuration>
         </plugin>
      </plugins>
   </build>
```

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">

   <!-- Repository Bean -->
   <bean id="bookRepository" class="com.navya.library.repository.BookRepository" />

   <!-- Service Bean and Dependency Injection -->
   <bean id="bookService" class="com.navya.library.service.BookService">
      <property name="bookRepository" ref="bookRepository" />
   </bean>
</beans>
```

BookRepository.java

```
package com.navya.library.repository;

public class BookRepository {
   public void saveBook(String title) {
      System.out.println(" BookRepository: Book saved → " + title);
   }
}
```

BookService.java

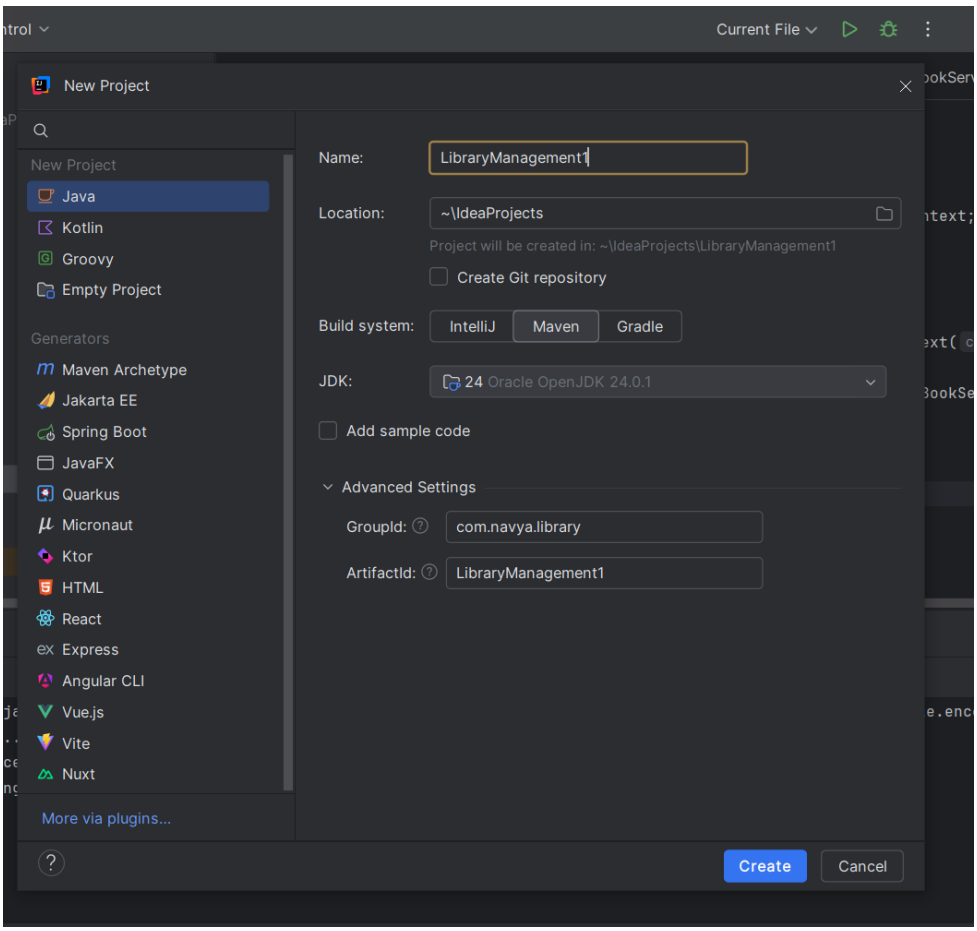
```
package com.navya.library.service;

import com.navya.library.repository.BookRepository;

public class BookService {
   private BookRepository bookRepository;

   // Setter method for Spring to inject BookRepository
```

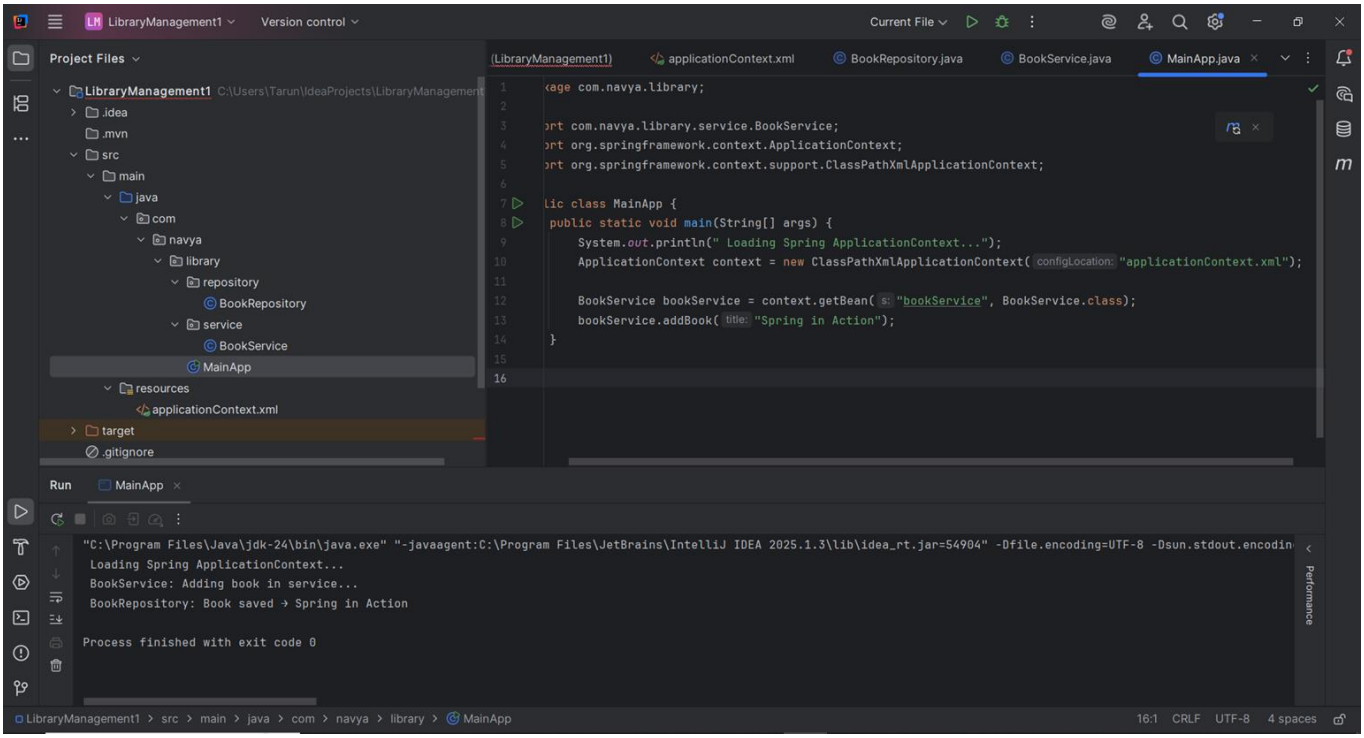
```
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void addBook(String title) {
        System.out.println(" BookService: Adding book in service...");
        bookRepository.saveBook(title);
    }
}
```

Main.java

```
package com.navya.library;

import com.navya.library.service.BookService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        System.out.println(" Loading Spring ApplicationContext...");
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

        BookService bookService = context.getBean("bookService", BookService.class);
        bookService.addBook("Spring in Action");
    }
}
```

OUTPUT :


New Project
New Project
Java
Kotlin
Groovy
Empty Project
Generators
Maven Archetype
Jakarta EE
Spring Boot
JavaFX
Quarkus
Micronaut
Ktor
HTML
React
Express
Angular CLI
Vue.js
Vite
Nuxt
More via plugins...
Name: LibraryManagement1
Location: ~\IdeaProjects
Project will be created in: ~\IdeaProjects\LibraryManagement1
Create Git repository
Build system: IntelliJ Maven Gradle
JDK: 24 Oracle OpenJDK 24.0.1
Add sample code
Advanced Settings
GroupId: com.navya.library
ArtifactId: LibraryManagement1
Create
Cancel


LibraryManagement1
Version control
Current File
Project Files
LibraryManagement1 C:\Users\Tarun\IdeaProjects\LibraryManagement
.idea
.mvn
src
main
java
com
navya
library
repository
BookRepository
service
BookService
MainApp
resources
applicationContext.xml
target
.gitignore
(LibraryManagement1)
applicationContext.xml
BookRepository.java
BookService.java
MainApp.java
age com.navya.library;
ort com.navya.library.service.BookService;
ort org.springframework.context.ApplicationContext;
ort org.springframework.context.support.ClassPathXmlApplicationContext;
lic class MainApp {
public static void main(String[] args) {
System.out.println(" Loading Spring ApplicationContext...");
ApplicationContext context = new ClassPathXmlApplicationContext(configLocation: "applicationContext.xml");
BookService bookService = context.getBean(s: "bookService", BookService.class);
bookService.addBook(title: "Spring in Action");
}
Run
MainApp
"C:\Program Files\Java\jdk-24\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2025.1.3\lib\idea_rt.jar=54904" -Dfile.encoding=UTF-8 -Dsun.stdout.encodin
Loading Spring ApplicationContext...
BookService: Adding book in service...
BookRepository: Book saved → Spring in Action
Process finished with exit code 0
Performance
LibraryManagement1 › src › main › java › com › navya › library › MainApp
16:1 CRLF UTF-8 4 spaces

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:

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

CODE :

applicationContext.xml

```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.navya.library.repository.BookRepository" />

   <bean id="bookService" class="com.navya.library.service.BookService">
      <property name="bookRepository" ref="bookRepository" />
   </bean>

</beans>
```

BookService.java

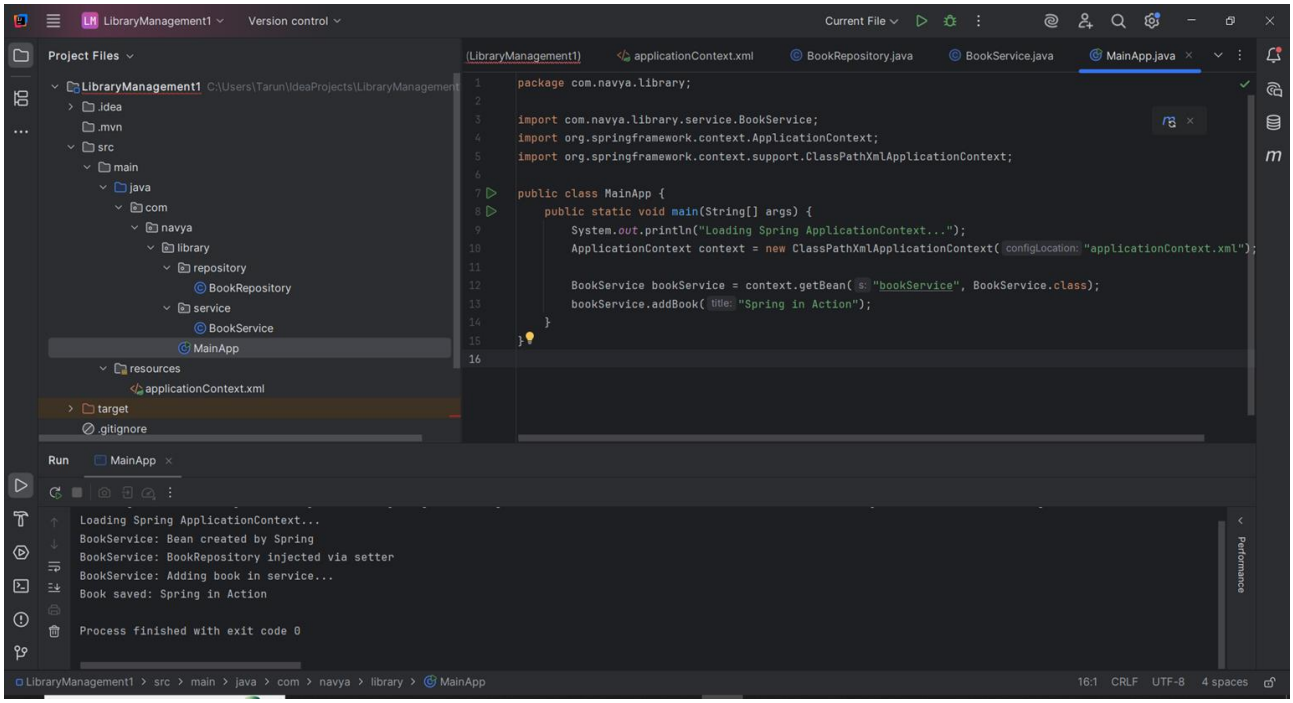
```java
package com.navya.library.service;

import com.navya.library.repository.BookRepository;

public class BookService {

   private BookRepository bookRepository;

   public BookService() {
      System.out.println("BookService: Bean created by Spring");
   }

   public void setBookRepository(BookRepository bookRepository) {
      System.out.println("BookService: BookRepository injected via setter");
      this.bookRepository = bookRepository;
   }

   public void addBook(String title) {
```

```
        System.out.println("BookService: Adding book in service...");
        bookRepository.saveBook(title);
    }
}
```

MainApp.java

```
package com.navya.library;

import com.navya.library.service.BookService;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        System.out.println("Loading Spring ApplicationContext...");
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

        BookService bookService = context.getBean("bookService", BookService.class);
        bookService.addBook("Spring in Action");
    }
}
```

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:

- **Create a New Maven Project:**
    - Create a new Maven project named **LibraryManagement**.
- **Add Spring Dependencies in pom.xml:**
    - Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
- **Configure Maven Plugins:**

Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

CODE :

pom.xml

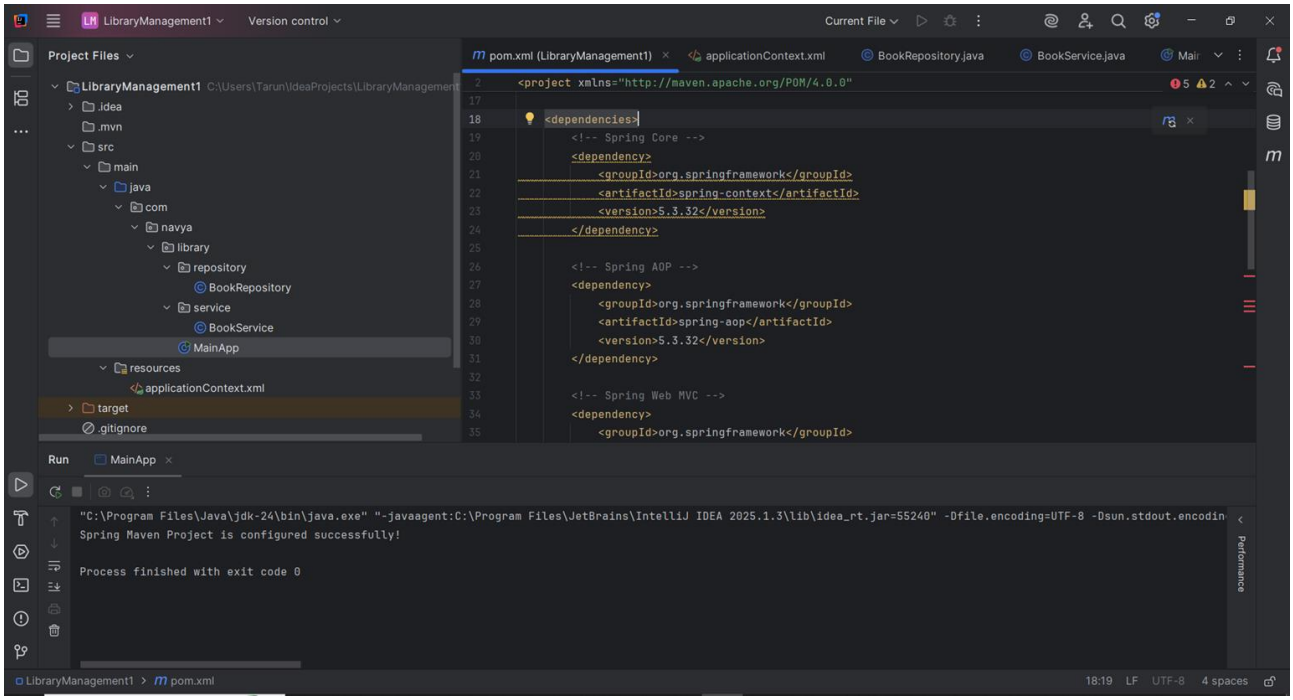
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.navya.library</groupId>
    <artifactId>LibraryManagement</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <!-- Spring Core -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.32</version>
        </dependency>

        <!-- Spring AOP -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>5.3.32</version>
        </dependency>

        <!-- Spring Web MVC -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>5.3.32</version>
        </dependency>
```

```
    <!-- JUnit for testing -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.10.0</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <!-- Maven Compiler Plugin -->
      <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>
```

MainApp.java

```
package com.navya.library;

public class MainApp {
    public static void main(String[] args) {
        System.out.println("Spring Maven Project is configured successfully!");
    }
}
```

OUTPUT :



SPRING DATA JPA WITH SPRINGBOOT, HIBERNATE :

Hands-on 4: Spring Data JPA – Quick Example

Create a MySQL database named ormlearn and a table Country with columns code and name.

- Insert sample data (e.g., 'IN', 'India') into the table.
- Use Spring Initializr to generate a project with dependencies: Spring Boot DevTools, Spring Data JPA, and MySQL Driver.
- Create Country entity class, CountryRepository, and CountryService.
- In the main class, fetch all countries from the database using countryService.getAllCountries() and display the result.
- Include proper configuration in application.properties.

CODE :

Country.java

```
package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import jakarta.persistence.Table;

@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 + "]";
    }
}
```

CountryService.java

```
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();
    }
}
```

CountryRepository.java

```
package com.cognizant.ormlearn.repository;

import org.springframework.data.jpa.repository.JpaRepository;
```

```
import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository
public interface CountryRepository extends JpaRepository<Country, String> {
}
```

OrmLearnApplication.java

```
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) {
		LOGGER.info("Inside main");

		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");
	}
}
```

application.properties

```
# Spring Framework and application log
logging.level.org.springframework=info
logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output
logging.level.org.hibernate.SQL=trace
logging.level.org.hibernate.type.descriptor.sql=trace

# Log pattern
logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-
25.25logger{25} %25M %4L %m%n
```

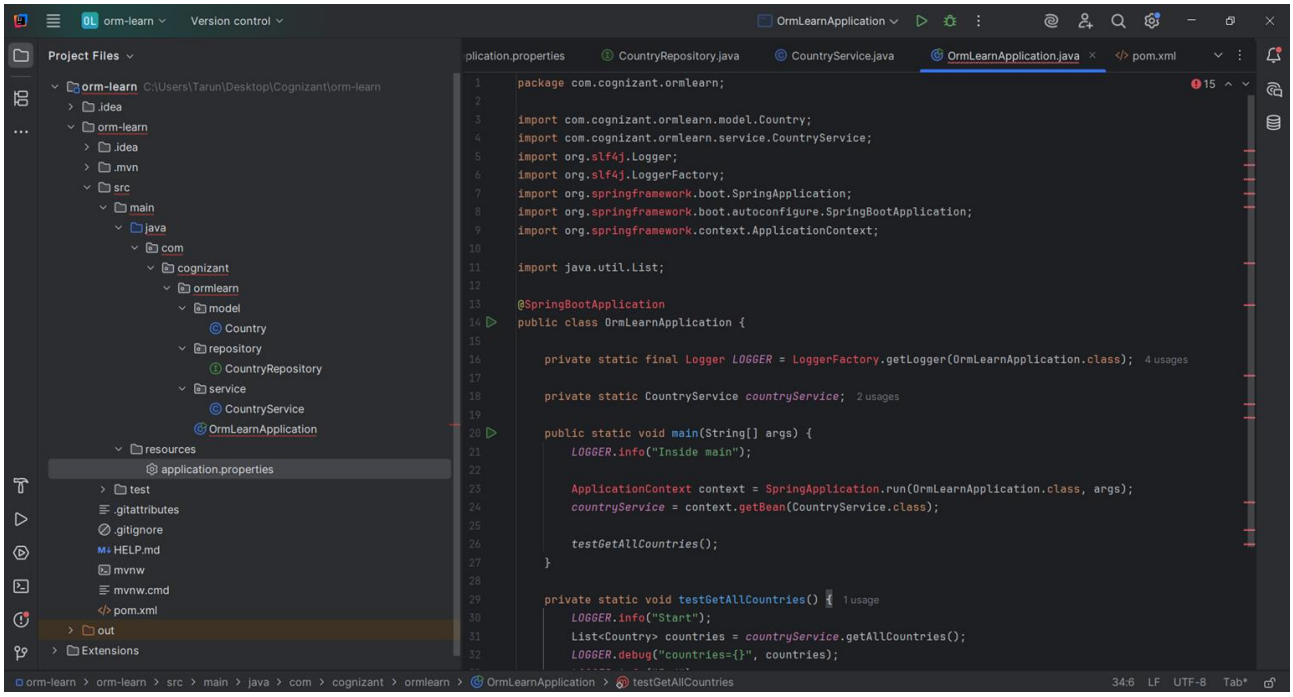
```
# Database configuration
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn
spring.datasource.username=root
spring.datasource.password=root

# Hibernate configuration
spring.jpa.hibernate.ddl-auto=validate
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect
```

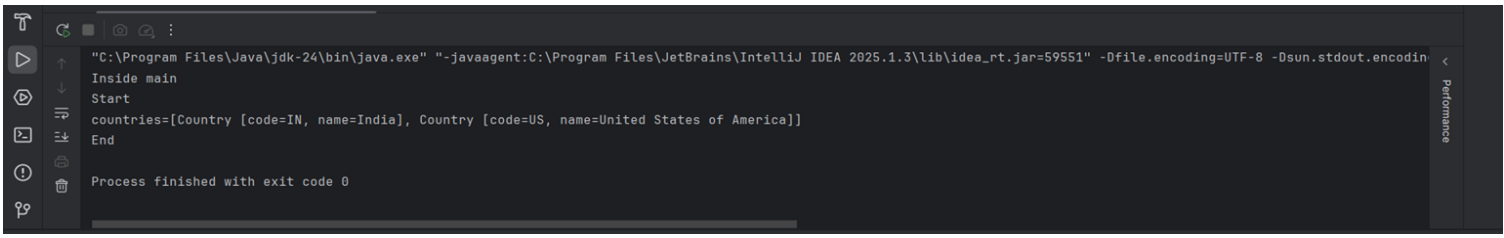
pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>

        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.2.4</version> <!-- This is STABLE and latest -->
                <relativePath/>
        </parent>

        <groupId>com.cognizant</groupId>
        <artifactId>orm-learn</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>orm-learn</name>
        <description>Demo project for Spring Data JPA and Hibernate</description>

        <properties>
                <java.version>17</java.version>
        </properties>

        <dependencies>
                <!-- Spring Boot JPA -->
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>

                <!-- MySQL Connector -->
                <dependency>
                        <groupId>com.mysql</groupId>
                        <artifactId>mysql-connector-j</artifactId>
                        <scope>runtime</scope>
                </dependency>

                <!-- Jakarta Persistence -->
                <dependency>
                        <groupId>jakarta.persistence</groupId>
                        <artifactId>jakarta.persistence-api</artifactId>
                        <version>3.1.0</version>
                </dependency>

                <!-- DevTools -->
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
```

```
                        <scope>runtime</scope>
                        <optional>true</optional>
                </dependency>

                <!-- Testing -->
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>
</project>
```

output :





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

Compare JPA, Hibernate, and Spring Data JPA. Provide code snippets to illustrate how the implementation differs between Hibernate and Spring Data JPA

<u>Differences</u> :

JPA (Java Persistence API)	Hibernate	Spring Data JPA
A specification that defines how to persist data in Java.	A popular ORM (Object Relational Mapping) tool and JPA implementation.	A Spring-based abstraction that builds on top of JPA, simplifying DB operations.
Specification only (needs implementation like Hibernate).	Concrete implementation of JPA.	Framework built on top of JPA and uses implementations like Hibernate underneath.
Manually managed (via **EntityManager**).	Manually handled using **Session** and **Transaction**.	Spring handles this automatically using annotations like **@Transaction**.

Requires writing more boilerplate code.	Slightly less, but still requires session handling.	Very minimal code; uses Spring-provided repositories.
When you want full control over database interactions.	When you prefer a mature ORM solution with powerful features.	When rapid development and less code is your priority (especially with Spring Boot).
Intermediate	Intermediate to Advanced	Beginner-friendly, especially if you're familiar with Spring.

Code Comparison: Hibernate vs Spring Data JPA (Adding an Employee) :

This section demonstrates how Hibernate *and* Spring Data JPA *differ in terms of boilerplate, simplicity, and abstraction — using a common example: Adding an employee to the database.*

Hibernate Code:

```
/* Method to CREATE an employee in the database */
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;
}
```

Explanation:

1. Manual session management, transactions, and exception handling.

2. More boilerplate code required.

Spring Data JPA Code:

EmployeeRepository.java

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

EmployeeService.java

```
@Autowired
private EmployeeRepository employeeRepository;

@Transactional
public void addEmployee(Employee employee) {
    employeeRepository.save(employee);
}
```

Explanation:

1. No need to manually open sessions or handle transactions.
2. Just use repository methods, thanks to Spring Data JPA abstraction.

Feature	Hibernate	Spring Data JPA
Boilerplate Code	High – needs session/transaction handling	Low – abstracted with repository
Simplicity	More complex setup	Simple and clean interface
Abstraction Level	Lower – developer handles persistence logic	Higher – Spring handles persistence internally
Ease of Use	Moderate	Very Easy
