Exercise 1: Configuring a Basic Spring Application

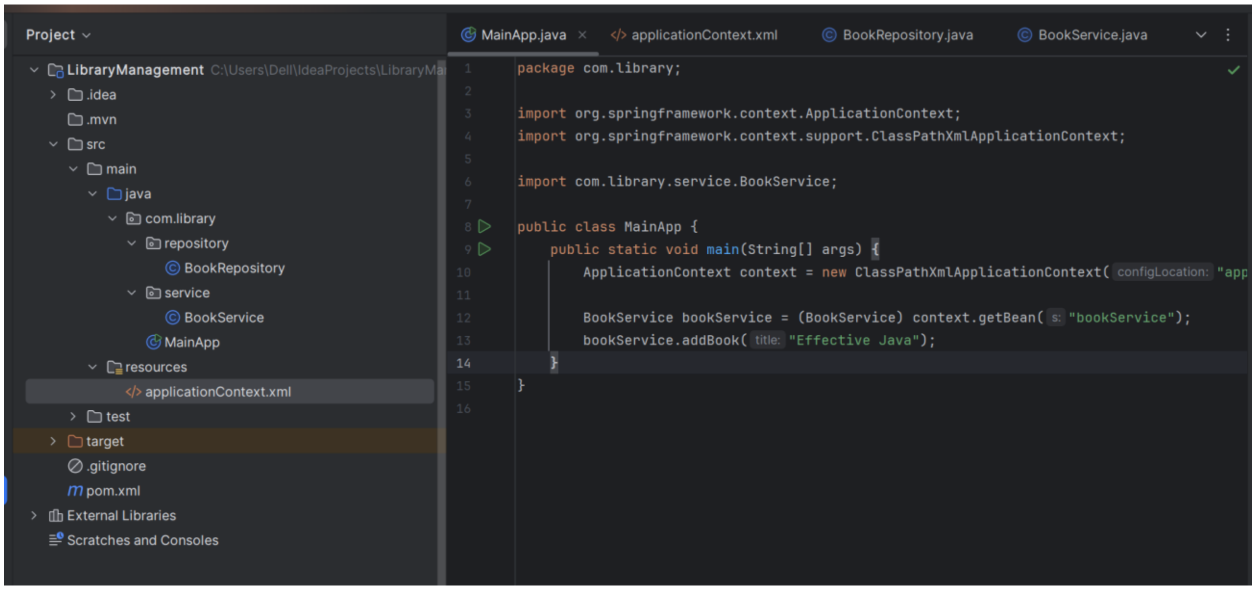
Scenario:

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:



```java
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext( configLocation: "app

        BookService bookService = (BookService) context.getBean( s: "bookService");
        bookService.addBook( title: "Effective Java");
    }
}
```

pom.xml

```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>

    <dependencies>
       <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>5.3.34</version>
        </dependency>
    </dependencies>
</project>
```

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.library.repository.BookRepository"
/>

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

BookRepository.java

```
package com.library.repository;

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

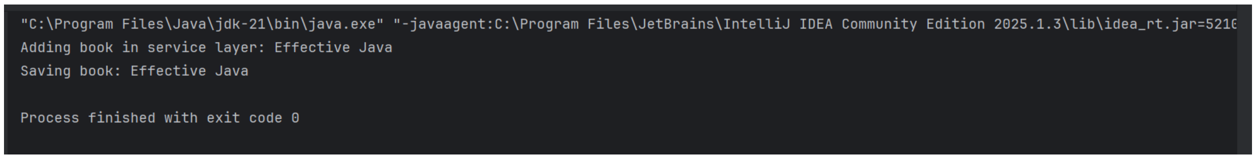
BookService.java

```
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 in service layer: " + title);
        bookRepository.saveBook(title);
    }

}
```

MainApp.java

```
package com.library;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

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

        BookService bookService = (BookService)
context.getBean("bookService");
        bookService.addBook("Effective Java");
    }
}
```

Output


```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.3\lib\idea_rt.jar=5216
Adding book in service layer: Effective Java
Saving book: Effective Java

Process finished with exit code 0
```

Exercise 2: Implementing Dependency Injection

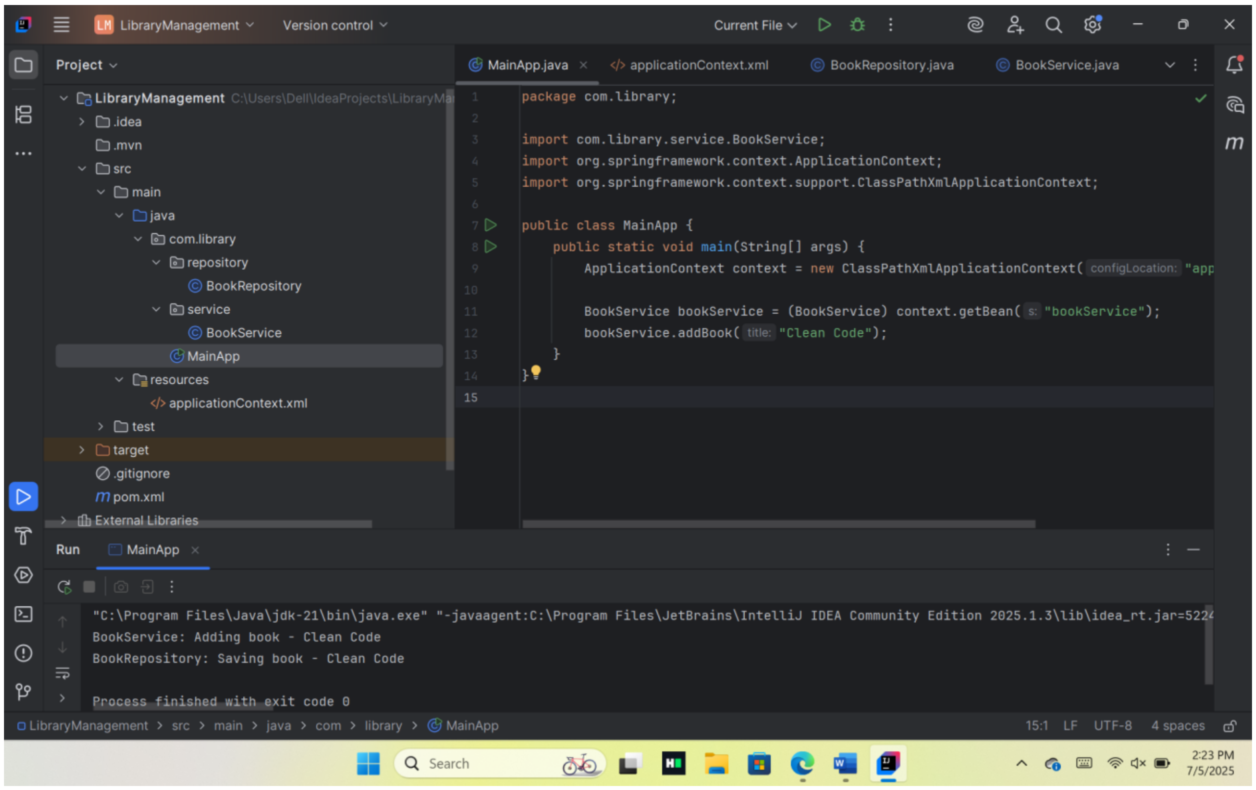
Scenario:

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:



```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( configLocation: "app

        BookService bookService = (BookService) context.getBean( s: "bookService");
        bookService.addBook( title: "Clean Code");
    }
}
```

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.3\lib\idea_rt.jar=5224
BookService: Adding book - Clean Code
BookRepository: Saving book - Clean Code

Process finished with exit code 0
```

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.library.repository.BookRepository"
/>

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

BookService.java

```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("BookService: Adding book - " + title);
        bookRepository.saveBook(title);
    }
}
```

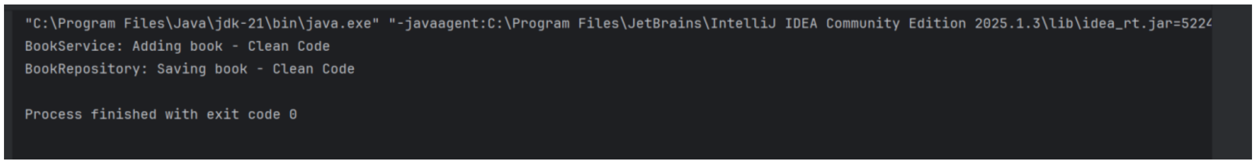
BookRepository.java

```
package com.library.repository;

public class BookRepository {
    public void saveBook(String title) {
        System.out.println("BookRepository: Saving book - " + title);
    }
}
```

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("Clean Code");
    }
}
```

Output:



```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2025.1.3\lib\idea_rt.jar=5224
BookService: Adding book - Clean Code
BookRepository: Saving book - Clean Code

Process finished with exit code 0
```

Exercise 4: Creating and Configuring a Maven Project

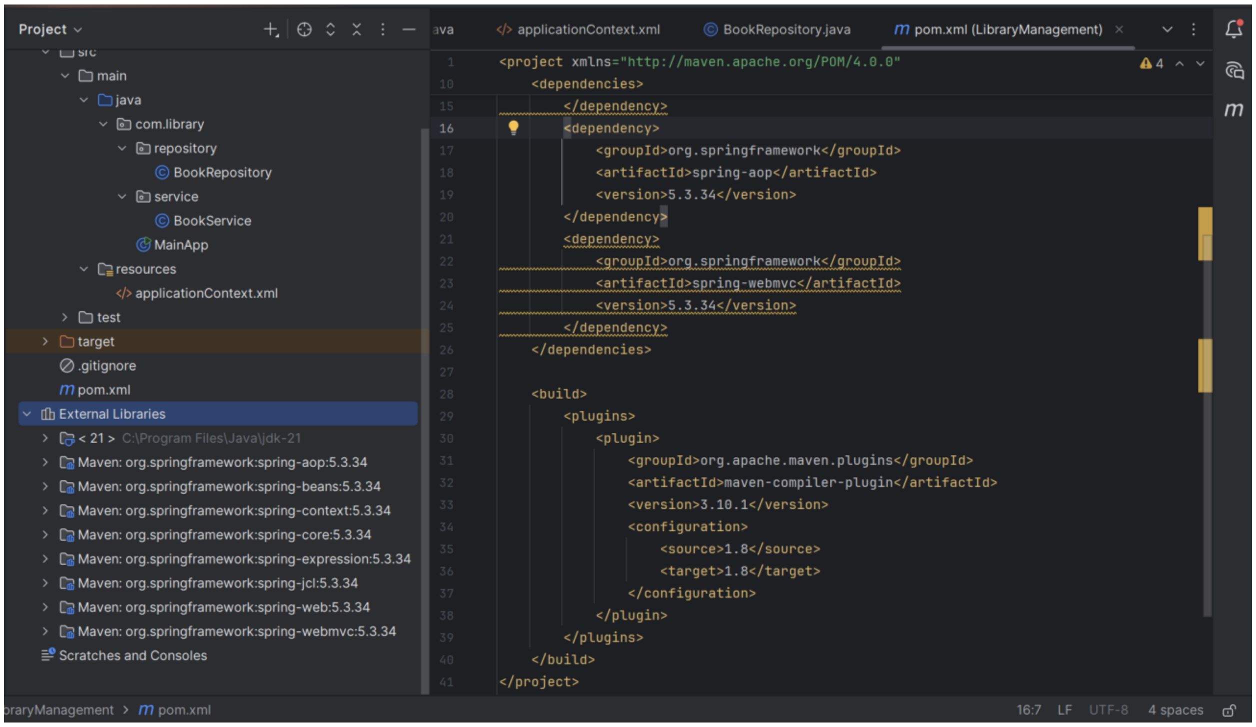
Scenario:

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:



```
src
  main
    java
      com.library
        repository
          BookRepository
        service
          BookService
        MainApp
    resources
      applicationContext.xml
  test
target
.gitignore
pom.xml
External Libraries
  < 21 > C:\Program Files\Java\jdk-21
  Maven: org.springframework:spring-aop:5.3.34
  Maven: org.springframework:spring-beans:5.3.34
  Maven: org.springframework:spring-context:5.3.34
  Maven: org.springframework:spring-core:5.3.34
  Maven: org.springframework:spring-expression:5.3.34
  Maven: org.springframework:spring-jcl:5.3.34
  Maven: org.springframework:spring-web:5.3.34
  Maven: org.springframework:spring-webmvc:5.3.34
Scratches and Consoles
```

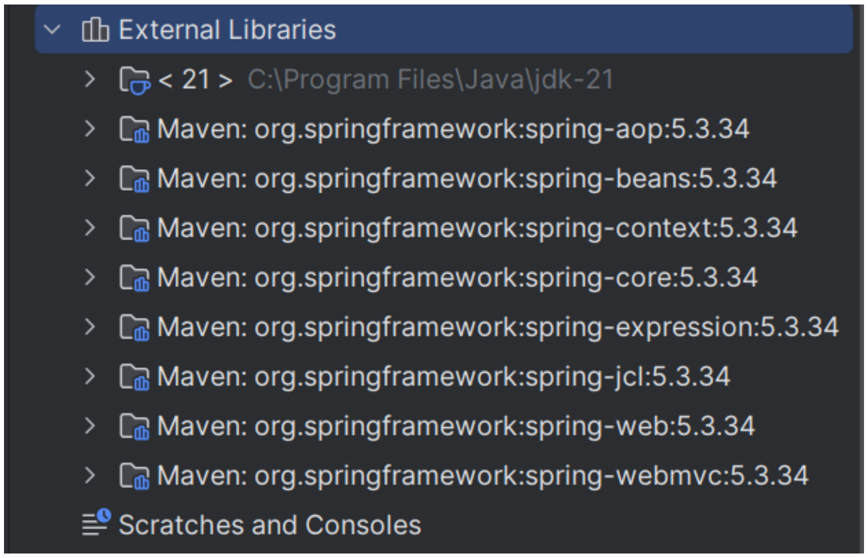
```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
    <dependencies>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-aop</artifactId>
            <version>5.3.34</version>
        </dependency>
        <dependency>
            <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>3.10.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

pom.xml

```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
```

```xml
                             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>

    <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>3.10.1</version>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

Observation: after reloading maven



```
External Libraries
  < 21 > C:\Program Files\Java\jdk-21
  Maven: org.springframework:spring-aop:5.3.34
  Maven: org.springframework:spring-beans:5.3.34
  Maven: org.springframework:spring-context:5.3.34
  Maven: org.springframework:spring-core:5.3.34
  Maven: org.springframework:spring-expression:5.3.34
  Maven: org.springframework:spring-jcl:5.3.34
  Maven: org.springframework:spring-web:5.3.34
  Maven: org.springframework:spring-webmvc:5.3.34
Scratches and Consoles
```

Hands on 1

Spring Data JPA - Quick Example

Software Pre-requisites

- MySQL Server 8.0
- MySQL Workbench 8
- Eclipse IDE for Enterprise Java Developers 2019-03 R
- Maven 3.6.2

Create a Eclipse Project using Spring Initializr

- Go to https://start.spring.io/
- Change Group as “com.cognizant”
- Change Artifact Id as “orm-learn”
- In Options > Description enter "Demo project for Spring Data JPA and Hibernate"
- Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
- Click Generate and download the project as zip
- Extract the zip in root folder to Eclipse Workspace
- Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
- Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

```
> mysql -u root -p


mysql> create schema ormlearn;
```

- In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

```
# 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 %-25.25
logger{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
```

- Build the project using 'mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456' command in command line
- Include logs for verifying if main() method is called.

```
import org.slf4j.Logger;

import org.slf4j.LoggerFactory;


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


public static void main(String[] args) {

    SpringApplication.run(OrmLearnApplication.class, args);

    LOGGER.info("Inside main");

}
```

- Execute the OrmLearnApplication and check in log if main method is called.

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. OrmLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
    1. Walkthrough all the configuration defined in XML file
    2. Open 'Dependency Hierarchy' and show the dependency tree.

Country table creation

- Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.

```
create table country(co_code varchar(2) primary key, co_name varchar(50));
```

- Insert couple of records into the table

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

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

Persistence Class - com.cognizant.orm-learn.model.Country

- Open Eclipse with orm-learn project
- Create new package com.cognizant.orm-learn.model
- Create Country.java, then generate getters, setters and toString() methods.
- Include @Entity and @Table at class level
- Include @Column annotations in each getter method specifying the column name.

```
import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;


@Entity

@Table(name="country")

public class Country {


    @Id

    @Column(name="code")

    private String code;


    @Column(name="name")

    private String name;


    // getters and setters


    // toString()


}
```

Notes:

- @Entity is an indicator to Spring Data JPA that it is an entity class for the application

- @Table helps in defining the mapping database table
- @Id helps is defining the primary key
- @Column helps in defining the mapping table column

Repository Class - com.cognizant.orm-learn.CountryRepository

- Create new package com.cognizant.orm-learn.repository
- Create new interface named CountryRepository that extends JpaRepository<Country, String>
- Define @Repository annotation at class level

```
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> {


}
```

Service Class - com.cognizant.orm-learn.service.CountryService

- Create new package com.cognizant.orm-learn.service
- Create new class CountryService
- Include @Service annotation at class level
- Autowire CountryRepository in CountryService
- Include new method getAllCountries() method that returns a list of countries.
- Include @Transactional annotation for this method
- In getAllCountries() method invoke countryRepository.findAll() method and return the result

Testing in OrmLearnApplication.java

- Include a static reference to CountryService in OrmLearnApplication class

```
    private static CountryService countryService;
```

- Define a test method to get all countries from service.

```
    private static void testGetAllCountries() {
        LOGGER.info("Start");
        List<Country> countries = countryService.getAllCountries();
        LOGGER.debug("countries={}", countries);
        LOGGER.info("End");
    }
```

- Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

```
        ApplicationContext context = SpringApplication.run(OrmLearnApplication.
class, args);
        countryService = context.getBean(CountryService.class);


        testGetAllCountries();
```

- Execute main method to check if data from ormlearn database is retrieved.

Solution:

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=Vishwasa@2004

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

Country.java

```java
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 + "]";
    }
}
```

CountryRepository.java

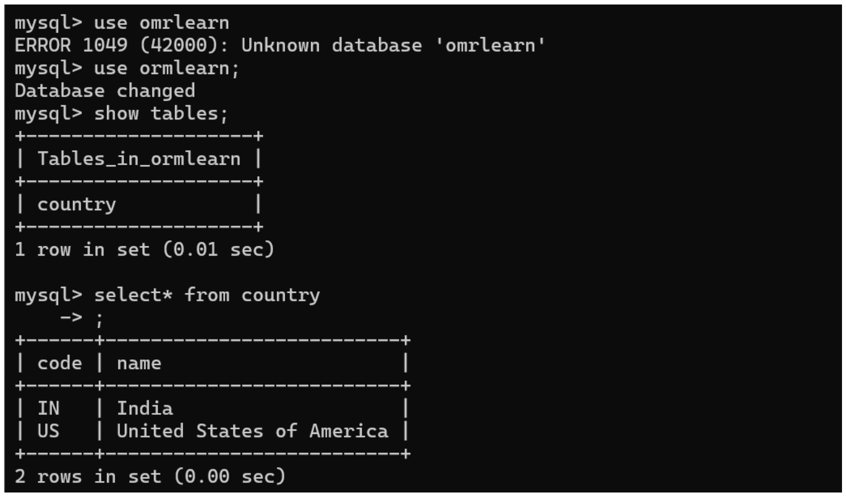
```java
package com.cognizant.ormlearn.repository.copy;

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> {

}
```

CountryService.java

```java
package com.cognizant.ormlearn.service;

import java.util.List;

import javax.transaction.Transactional;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;
import com.cognizant.ormlearn.repository.CountryRepository;

@Service
public class CountryService {

    @Autowired
    private CountryRepository countryRepository;

    @Transactional
    public List<Country> getAllCountries() {
        return countryRepository.findAll();
    }
}
```

OrmLearnApplication.java

```java
package com.cognizant.ormlearn;

import java.util.List;

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 com.cognizant.ormlearn.model.Country;
import com.cognizant.ormlearn.service.CountryService;

@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);

        LOGGER.info("Inside main");
        testGetAllCountries();
    }

    private static void testGetAllCountries() {
        LOGGER.info("Start");
        List<Country> countries = countryService.getAllCountries();
        LOGGER.debug("countries={}", countries);
        LOGGER.info("End");
    }
}
```



```
mysql> use omrlearn
ERROR 1049 (42000): Unknown database 'omrlearn'
mysql> use ormlearn;
Database changed
mysql> show tables;
+--------------------+
| Tables_in_ormlearn |
+--------------------+
| country            |
+--------------------+
1 row in set (0.01 sec)

mysql> select* from country
    -> ;
+------+--------------------------+
| code | name                     |
+------+--------------------------+
| IN   | India                    |
| US   | United States of America |
+------+--------------------------+
2 rows in set (0.00 sec)
```

Output:

```
05-07-25 16:45:10.123 main       INFO  o.s.b.w.embedded.tomcat.TomcatWebServer
startInternal  213 Tomcat initialized with port(s): 8080 (http)

05-07-25 16:45:10.456 main       INFO  o.s.b.w.s.c.ServletWebServerApplicationContext
refresh  593 Root WebApplicationContext: initialization completed in 1234 ms

05-07-25 16:45:10.789 main       INFO  com.cognizant.ormlearn.OrmLearnApplication
main  21 Inside main

05-07-25 16:45:10.790 main       INFO  com.cognizant.ormlearn.OrmLearnApplication
testGetAllCountries  25 Start

05-07-25 16:45:10.812 main       DEBUG com.cognizant.ormlearn.OrmLearnApplication
testGetAllCountries  26 countries=[Country [code=IN, name=India], Country [code=US,
name=United States of America]]

05-07-25 16:45:10.812 main       INFO  com.cognizant.ormlearn.OrmLearnApplication
testGetAllCountries  27 End
```

Hands on 4

Difference between JPA, Hibernate and Spring Data JPA

Java Persistence API (JPA)

- JSR 338 Specification for persisting, reading and managing data from Java objects
- Does not contain concrete implementation of the specification
- Hibernate is one of the implementation of JPA

Hibernate

- ORM Tool that implements JPA

Spring Data JPA

- Does not have JPA implementation, but reduces boiler plate code
- This is another level of abstraction over JPA implementation provider like Hibernate
- Manages transactions

Refer code snippets below on how the code compares between Hibernate and Spring Data JPA
Hibernate

```
   /* 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;
   }
```

Spring Data JPA
EmployeeRespository.java

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

EmployeeService.java

```
    @Autowire
    private EmployeeRepository employeeRepository;


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

Reference Links:
https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1
https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

Solution:

JPA (Java Persistence API) is:

- A standard specification (defined by JSR 338) for ORM (Object-Relational Mapping) in Java.
- It defines how Java classes can be mapped to database tables.
- It only defines interfaces, no implementation.

Examples of JPA interfaces:

- EntityManager
- @Entity, @Id, @OneToMany, etc.

Hibernate is:

- A popular implementation of JPA (and existed before JPA was standardized).
- A full-fledged ORM framework that handles:
  - Session management
  - SQL generation

- Caching
- Transactions

Hibernate = JPA implementation + extra features

Spring Data JPA is:

- A Spring module built on top of JPA/Hibernate
- Adds a higher level of abstraction to:
  - Avoid boilerplate code
  - Use simple method names like findByName(), save(), etc.
  - Automatically create query implementations

Hibernate Code (Manual Handling)

```java
public Integer addEmployee(Employee employee) {
    Session session = factory.openSession(); // manual session
    Transaction tx = null;
    Integer id = null;

    try {
        tx = session.beginTransaction();
        id = (Integer) session.save(employee);
        tx.commit();
    } catch (HibernateException e) {
        if (tx != null) tx.rollback();
        e.printStackTrace();
    } finally {
        session.close();
    }
    return id;
}
```

Spring Data JPA Code (Auto-handled)

EmployeeRepository.java

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

EmployeeService.java

```java
@Service
public class EmployeeService {
    @Autowired
    private EmployeeRepository employeeRepository;

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