Spring Data JPA Hands On

**Software Prerequisites**

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

**1. Project Setup using Spring Initializr**

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

**2. MySQL Database Setup**

1. Open MySQL client:
2. mysql -u root -p
3. Create schema:
4. CREATE SCHEMA ormlearn;

**3. Configure Application Properties**

# 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

# Log format

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

# Database

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=valluri

# Hibernate

spring.jpa.hibernate.ddl-auto=update

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

**4. Build the Project**

**5. Main Application Logging**

package com.cognizant.ormlearn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class OrmLearnApplication {

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

public static void main(String[] args) {

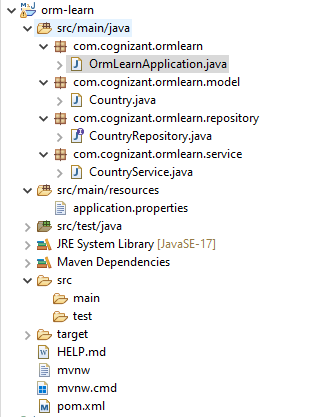
ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

}

}

**6. Project Structure Overview**

****

**7. Understanding Key Annotations**

|  |  |
| --- | --- |
| **Annotation** | **Purpose** |
| @SpringBootApplication | Enables auto-configuration, component scanning, and extra Spring Boot features |
| @Entity | Marks a class as a JPA entity to map to a database table |
| @Table | Specifies the mapped database table name |
| @Id | Defines the primary key field |
| @Column | Maps a field to a database column |
| @Repository | Marks a class/interface as a Spring Data repository |
| @Service | Marks a service layer class |
| @Transactional | Manages transaction boundaries |

**8. Database Table Creation: country**

create schema ormlearn;

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');

**9. Persistence Class: Country**

**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 = "co\_code")

**private** String code;

@Column(name = "co\_name")

**private** String name;

// Getter for code

**public** String getCode() {

**return** code;

}

// Setter for code

**public** **void** setCode(String code) {

**this**.code = code;

}

// Getter for name

**public** String getName() {

**return** name;

}

// Setter for name

**public** **void** setName(String name) {

**this**.name = name;

}

// toString method for printing

@Override

**public** String toString() {

**return** "Country [code=" + code + ", name=" + name + "]";

}

}

**10. Repository Interface: CountryRepository**

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> {

}

**11. Service Class: CountryService**

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

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();

}

}

**12. Testing Data Retrieval in OrmLearnApplication**

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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import org.springframework.context.ApplicationContext;

@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);

LOGGER.info("Inside main");

// Get the service bean from context

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

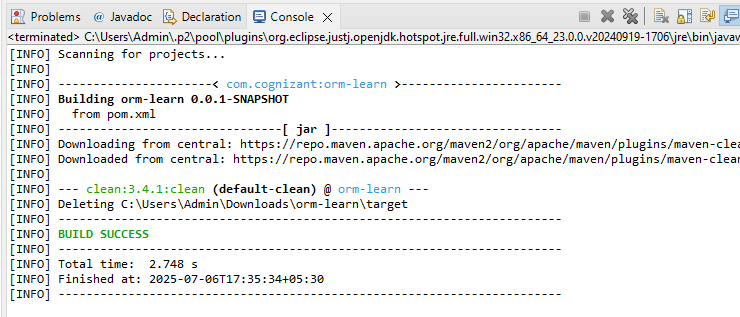
List<Country> countries = countryService.getAllCountries();

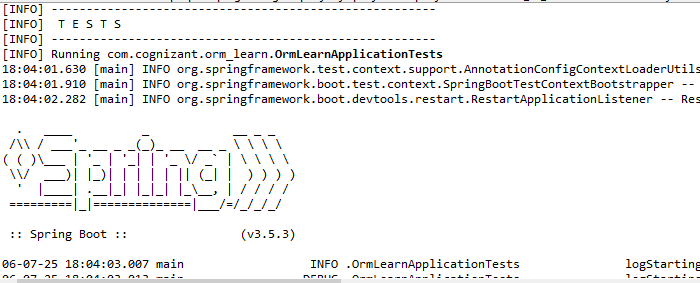
LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

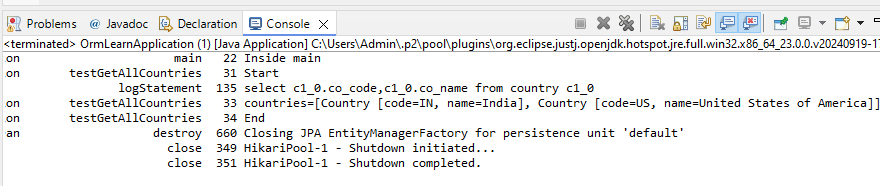
}







Output:



**Difference between JPA, Hibernate and Spring Data JPA**

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

* **What it is:**  
  JPA is a **specification** (defined by Java standard JSR 338). It defines a standard way to map Java objects to relational database tables and perform CRUD (Create, Read, Update, Delete) operations.
* **Important:** JPA **does NOT provide any implementation** — it's just an interface and a set of rules (annotations and APIs) to follow.
* **Purpose:** It standardizes ORM behavior so you can write your code once and switch between different implementations without changing your business logic.

**2. Hibernate**

* **What it is:**  
  Hibernate is an **implementation** of the JPA specification. It's an ORM (Object-Relational Mapping) tool.
* **Role:** It provides the actual mechanisms to persist Java objects to the database using SQL behind the scenes.
* **Extra features:** Hibernate also offers many features beyond the JPA spec, like advanced caching, lazy loading, and more configuration options.
* **Usage:** If you use Hibernate directly (without Spring Data JPA), you must manage things like opening/closing database sessions and transactions manually.

**3. Spring Data JPA**

* **What it is:**  
  Spring Data JPA is a **Spring framework module** that builds on top of JPA implementations (like Hibernate) to make development easier.
* **Role:** It provides an abstraction that eliminates the need to write repetitive code (called boilerplate code), like opening sessions or writing common queries.
* **Key features:**
  + Automatically implements repository interfaces (JpaRepository) based on method names.
  + Manages transactions for you via @Transactional.
  + Integrates seamlessly with Spring’s Dependency Injection.
* **Important:** Spring Data JPA does NOT implement JPA itself; it uses an underlying JPA provider like Hibernate.

