Spring Data JPA with Spring Boot, Hibernate - Hands-on 1

# 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"  
• Select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"  
• Click Generate and download the project as zip  
• Extract the zip to Eclipse Workspace and import it.  
• Create a schema "ormlearn" in MySQL:  
> mysql -u root -p  
mysql> create schema ormlearn;

# Configuration in application properties

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

# Build the Project

Use the following command to build:  
mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456

# Main Method Logging

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);  
  
public static void main(String[] args) {  
 SpringApplication.run(OrmLearnApplication.class, args);  
 LOGGER.info("Inside main");  
}

# Project Folder Walkthrough

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 - XML configuration and dependency tree

# Country Table SQL

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 Entity Class

import javax.persistence.\*;  
  
@Entity  
@Table(name="country")  
public class Country {  
  
 @Id  
 @Column(name="code")  
 private String code;  
  
 @Column(name="name")  
 private String name;  
  
 // Getters and Setters  
  
 // toString()  
}

# CountryRepository Interface

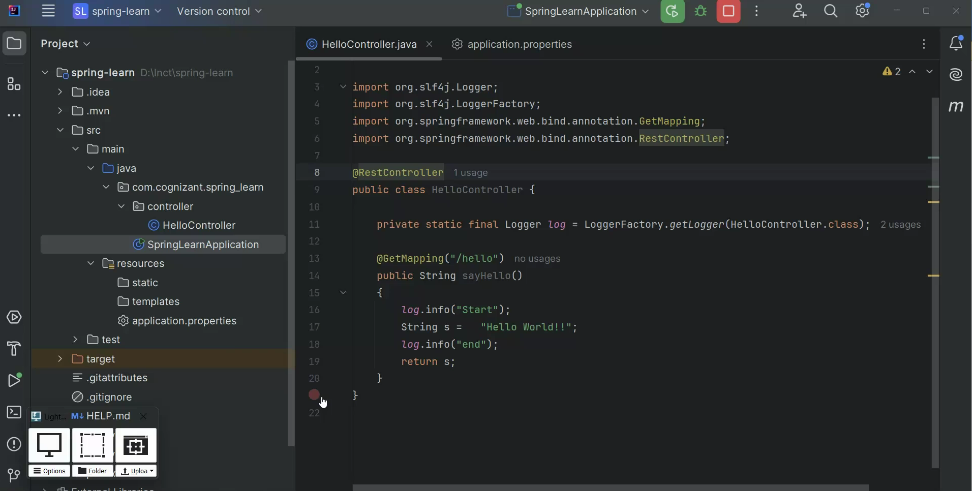
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
import com.cognizant.ormlearn.model.Country;  
  
@Repository  
public interface CountryRepository extends JpaRepository<Country, String> {}

# CountryService Class

import org.springframework.stereotype.Service;  
import javax.transaction.Transactional;  
  
@Service  
public class CountryService {  
  
 @Autowired  
 private CountryRepository countryRepository;  
  
 @Transactional  
 public List<Country> getAllCountries() {  
 return countryRepository.findAll();  
 }  
}

# OrmLearnApplication Testing

private static CountryService countryService;  
  
private static void testGetAllCountries() {  
 LOGGER.info("Start");  
 List<Country> countries = countryService.getAllCountries();  
 LOGGER.debug("countries={}", countries);  
 LOGGER.info("End");  
}  
  
ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);  
countryService = context.getBean(CountryService.class);  
testGetAllCountries();



# Difference between JPA, Hibernate, and Spring Data JPA

Java Persistence API (JPA)  
• Specification (JSR 338) for object-relational mapping.  
• No concrete implementation.  
  
Hibernate  
• Implementation of JPA.  
• Offers ORM features.  
  
Spring Data JPA  
• Abstraction over JPA/Hibernate.  
• Reduces boilerplate.  
• Manages transactions.

# Hibernate vs Spring Data JPA Code Comparison

Hibernate:  
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:  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}  
  
@Service  
public class EmployeeService {  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}