**Hands on 1**

**Spring Data JPA - Quick Example**   
  
A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer program

AI-generated content may be incorrect.

**Application.properties:**

# Application Properties

spring.application.name=om-lesm

# Database Configuration (MySQL)

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=live123live123

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate/JPA Configuration

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.jdbc.time\_zone=UTC

# Logging Configuration

logging.level.root=INFO

logging.level.org.springframework=INFO

logging.level.org.hibernate.SQL=DEBUG

logging.level.org.hibernate.type.descriptor.sql=TRACE

logging.level.com.cognizant.om.lesm=DEBUG

# Console log pattern (optimized for Java 17)

logging.pattern.console=%d{dd-MM-yyyy HH:mm:ss.SSS} %-5level [%-15.15thread] %-40.40logger{39} : %msg%n

# Spring Boot Specific

spring.devtools.restart.enabled=true

**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;

    // Getters and Setters

    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**

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> {

}

**countryService.java**

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import jakarta.transaction.Transactional;

@Service

public class CountryService {

    @Autowired

    private CountryRepository countryRepository;

    @Transactional

    public List<Country> getAllCountries() {

        return countryRepository.findAll();

    }

}

**OrmLearnApplication.java**

package com.cognizant.ormlearn;

import java.util.List;

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;

@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");

    }

}

**Build Output:**

A close-up of a computer code

AI-generated content may be incorrect.

**Run Output:**

A screenshot of a computer code

AI-generated content may be incorrect.

**Hands on 4**

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

| **Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (JSR 338) | Implementation | Framework/Abstraction |
| **Purpose** | Standard API for ORM in Java | ORM framework; implements JPA | Simplifies JPA-based persistence |
| **Implementation** | No implementation, only interfaces | Provides implementation of JPA | Built on top of JPA (uses JPA providers like Hibernate) |
| **Boilerplate** | Requires manual code | Reduces some boilerplate | Greatly reduces boilerplate (via repositories) |
| **Transaction Management** | Done manually or via container | Manual unless integrated with Spring | Handled by Spring |
| **Examples of Use** | EntityManager, Queries | Session, Transaction | JpaRepository, CrudRepository |

**Hibernate (Manual approach)**

**/\* 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;

   }

* Requires manual session handling.
* Manual transaction control.
* Exception handling and resource closing must be done explicitly.

**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);

  }

* No need to manage session or transaction.
* Just call save(), findById(), etc.
* Reduces up to 90% of the boilerplate code.