**Spring Data JPA with Spring Boot, Hibernate**

**Spring-data-jpa-handson**

**Spring Data JPA - Quick Example**

**//country.java**

package com.cognizant.orm\_learn.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

@Override

public String toString() {

return "Country{" +

"code='" + code + '\'' +

", name='" + name + '\'' +

'}';

}

}

**//CountryRepository.java**

package com.cognizant.orm\_learn.repository;

import com.cognizant.orm\_learn.model.Country;

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**// CountryService.java**

package com.cognizant.orm\_learn.service;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.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();

}

}

**//OmrLearnApplication.java**

package com.cognizant.orm\_learn;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.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) {

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.application.name=orm-learn

# 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

# Oracle DB config

spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

spring.datasource.url=jdbc:oracle:thin:@localhost:1521/XEPDB1

spring.datasource.username=ormlearn

spring.datasource.password=ormlearn

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

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

**//OmrLearnApplicationTest.java**

package com.cognizant.orm\_learn;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class OrmLearnApplicationTests {

@Test

void contextLoads() {

}

}

**// schema.sql**

-- SQL Schema for Country table

-- This file demonstrates DDL operations and will be executed automatically by Spring Boot

-- Drop table if exists (for clean setup)

DROP TABLE IF EXISTS countries;

-- Create countries table

CREATE TABLE countries (

country\_id BIGINT AUTO\_INCREMENT PRIMARY KEY,

country\_code VARCHAR(3) NOT NULL UNIQUE,

iso\_code VARCHAR(2) NOT NULL,

country\_name VARCHAR(100) NOT NULL,

created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

-- Create indexes for better query performance

CREATE INDEX idx\_country\_code ON countries(country\_code);

CREATE INDEX idx\_iso\_code ON countries(iso\_code);

CREATE INDEX idx\_country\_name ON countries(country\_name);

-- Insert some sample data (optional)

INSERT INTO countries (country\_code, iso\_code, country\_name) VALUES

('USA', 'US', 'United States'),

('GBR', 'GB', 'United Kingdom'),

('IND', 'IN', 'India'),

('DEU', 'DE', 'Germany'),

('FRA', 'FR', 'France'),

('JPN', 'JP', 'Japan'),

('CHN', 'CN', 'China'),

('CAN', 'CA', 'Canada'),

('AUS', 'AU', 'Australia'),

('BRA', 'BR', 'Brazil');

**//CountryServiceTest.java**

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.BeforeEach;

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

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.test.context.ActiveProfiles;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

/\*\*

\* Integration tests for CountryService

\*

\* This test class demonstrates:

\* - Spring Boot Test configuration

\* - Integration testing with H2 database

\* - JPA repository testing

\* - Service layer testing

\* - Transaction management in tests

\*/

@SpringBootTest

@ActiveProfiles("test")

@Transactional

public class CountryServiceTest {

@Autowired

private CountryService countryService;

@Autowired

private CountryRepository countryRepository;

@BeforeEach

void setUp() {

// Clean up database before each test

countryRepository.deleteAll();

}

@Test

void testCreateCountry() {

// Given

String countryCode = "TST";

String isoCode = "TS";

String countryName = "Test Country";

// When

Country createdCountry = countryService.createCountry(countryCode, isoCode, countryName);

// Then

assertNotNull(createdCountry);

assertNotNull(createdCountry.getId());

assertEquals(countryCode, createdCountry.getCountryCode());

assertEquals(isoCode, createdCountry.getIsoCode());

assertEquals(countryName, createdCountry.getCountryName());

assertNotNull(createdCountry.getCreatedAt());

assertNotNull(createdCountry.getUpdatedAt());

}

@Test

void testCreateDuplicateCountryCode() {

// Given

String countryCode = "DUP";

countryService.createCountry(countryCode, "DP", "Duplicate Country");

// When & Then

assertThrows(RuntimeException.class, () -> {

countryService.createCountry(countryCode, "D2", "Another Duplicate");

});

}

@Test

void testGetAllCountries() {

// Given

countryService.createCountry("US1", "U1", "Test Country 1");

countryService.createCountry("US2", "U2", "Test Country 2");

countryService.createCountry("US3", "U3", "Test Country 3");

// When

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

// Then

assertEquals(3, countries.size());

}

@Test

void testGetCountryById() {

// Given

Country createdCountry = countryService.createCountry("FND", "FN", "Find Country");

// When

Country foundCountry = countryService.getCountryById(createdCountry.getId());

// Then

assertNotNull(foundCountry);

assertEquals(createdCountry.getId(), foundCountry.getId());

assertEquals("FND", foundCountry.getCountryCode());

assertEquals("Find Country", foundCountry.getCountryName());

}

@Test

void testGetCountryByIdNotFound() {

// When & Then

assertThrows(RuntimeException.class, () -> {

countryService.getCountryById(999L);

});

}

@Test

void testUpdateCountry() {

// Given

Country country = countryService.createCountry("UPD", "UP", "Update Country");

country.setCountryName("Updated Country Name");

// When

Country updatedCountry = countryService.updateCountry(country);

// Then

assertEquals("Updated Country Name", updatedCountry.getCountryName());

assertEquals(country.getId(), updatedCountry.getId());

}

@Test

void testDeleteCountry() {

// Given

Country country = countryService.createCountry("DEL", "DL", "Delete Country");

Long countryId = country.getId();

// When

countryService.deleteCountry(countryId);

// Then

assertThrows(RuntimeException.class, () -> {

countryService.getCountryById(countryId);

});

}

@Test

void testFindByCountryCode() {

// Given

countryService.createCountry("SRC", "SR", "Search Country");

// When

Country foundCountry = countryService.findByCountryCode("SRC");

// Then

assertNotNull(foundCountry);

assertEquals("SRC", foundCountry.getCountryCode());

assertEquals("Search Country", foundCountry.getCountryName());

}

@Test

void testFindByCountryCodeNotFound() {

// When

Country foundCountry = countryService.findByCountryCode("NON");

// Then

assertNull(foundCountry);

}

@Test

void testFindByIsoCode() {

// Given

countryService.createCountry("ISO1", "IS", "ISO Country 1");

countryService.createCountry("ISO2", "IS", "ISO Country 2");

// When

List<Country> countries = countryService.findByIsoCode("IS");

// Then

assertEquals(2, countries.size());

}

@Test

void testFindByCountryNameContaining() {

// Given

countryService.createCountry("US1", "U1", "United States");

countryService.createCountry("UK1", "U2", "United Kingdom");

countryService.createCountry("FR1", "F1", "France");

// When

List<Country> countries = countryService.findByCountryNameContaining("United");

// Then

assertEquals(2, countries.size());

assertTrue(countries.stream().allMatch(c -> c.getCountryName().contains("United")));

}

@Test

void testGetCountryCount() {

// Given

countryService.createCountry("CNT1", "C1", "Count Country 1");

countryService.createCountry("CNT2", "C2", "Count Country 2");

// When

long count = countryService.getCountryCount();

// Then

assertEquals(2, count);

}

@Test

void testExistsByCountryCode() {

// Given

countryService.createCountry("EXT", "EX", "Exists Country");

// When & Then

assertTrue(countryService.existsByCountryCode("EXT"));

assertFalse(countryService.existsByCountryCode("NEX"));

}

@Test

void testFindAllCountryCodes() {

// Given

countryService.createCountry("CD1", "C1", "Code Country 1");

countryService.createCountry("CD2", "C2", "Code Country 2");

countryService.createCountry("CD3", "C3", "Code Country 3");

// When

List<String> codes = countryService.findAllCountryCodes();

// Then

assertEquals(3, codes.size());

assertTrue(codes.contains("CD1"));

assertTrue(codes.contains("CD2"));

assertTrue(codes.contains("CD3"));

}

@Test

void testFindCountriesWithLongNames() {

// Given

countryService.createCountry("SHT", "SH", "Short");

countryService.createCountry("LNG", "LG", "Very Long Country Name");

// When

List<Country> longNameCountries = countryService.findCountriesWithLongNames(10);

// Then

assertEquals(1, longNameCountries.size());

assertEquals("Very Long Country Name", longNameCountries.get(0).getCountryName());

}

}

**//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.cognizant</groupId>

<artifactId>spring-data-jpa-example</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>Spring Data JPA Quick Example</name>

<description>Spring Data JPA with Spring Boot and Hibernate Example</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.7.0</version>

<relativePath/>

</parent>

<properties>

<java.version>11</java.version>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

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

</properties>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot Starter Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database (for development and testing) -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- MySQL Connector (optional, for production) -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter Test -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<!-- Spring Boot Starter Validation -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!-- Spring Boot DevTools (optional, for development) -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Spring Boot Maven Plugin -->

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

<!-- Maven Compiler Plugin -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>11</source>

<target>11</target>

</configuration>

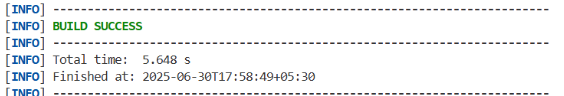
</plugin>

</plugins>

</build>

</project>

**OUTPUT:**

****

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

**//Employee.java**

package com.example;

import javax.persistence.\*;

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Integer id;

@Column(name = "first\_name", nullable = false)

private String firstName;

@Column(name = "last\_name", nullable = false)

private String lastName;

@Column(name = "email", unique = true, nullable = false)

private String email;

@Column(name = "salary")

private Double salary;

// Default constructor

public Employee() {}

// Constructor with parameters

public Employee(String firstName, String lastName, String email, Double salary) {

this.firstName = firstName;

this.lastName = lastName;

this.email = email;

this.salary = salary;

}

// Getters and Setters

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

public Double getSalary() {

return salary;

}

public void setSalary(Double salary) {

this.salary = salary;

}

@Override

public String toString() {

return "Employee{" +

"id=" + id +

", firstName='" + firstName + '\'' +

", lastName='" + lastName + '\'' +

", email='" + email + '\'' +

", salary=" + salary +

'}';

}

}

**//EmployeeDAO.java**

package com.example;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.util.List;

public class EmployeeDAO {

private static SessionFactory factory;

static {

try {

factory = new Configuration().configure().buildSessionFactory();

} catch (Throwable ex) {

System.err.println("Failed to create sessionFactory object." + ex);

throw new ExceptionInInitializerError(ex);

}

}

/\* 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;

}

/\* Method to READ all employees from the database \*/

public List<Employee> listEmployees() {

Session session = factory.openSession();

Transaction tx = null;

List<Employee> employees = null;

try {

tx = session.beginTransaction();

employees = session.createQuery("FROM Employee", Employee.class).list();

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employees;

}

/\* Method to UPDATE employee in the database \*/

public void updateEmployee(Integer employeeID, Double salary) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee employee = session.get(Employee.class, employeeID);

employee.setSalary(salary);

session.update(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

}

/\* Method to DELETE an employee from the database \*/

public void deleteEmployee(Integer employeeID) {

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Employee employee = session.get(Employee.class, employeeID);

session.delete(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

}

/\* Method to close the SessionFactory \*/

public static void shutdown() {

factory.close();

}

}

**//HibernateExample.java**

package com.example;

import java.util.List;

public class HibernateExample {

public static void main(String[] args) {

EmployeeDAO employeeDAO = new EmployeeDAO();

System.out.println("=== Hibernate Example ===");

System.out.println("Demonstrating manual session and transaction management\n");

// Create employees

System.out.println("1. Creating employees...");

Integer emp1Id = employeeDAO.addEmployee(new Employee("John", "Doe", "john.doe@example.com", 50000.0));

Integer emp2Id = employeeDAO.addEmployee(new Employee("Jane", "Smith", "jane.smith@example.com", 60000.0));

Integer emp3Id = employeeDAO.addEmployee(new Employee("Bob", "Johnson", "bob.johnson@example.com", 55000.0));

System.out.println("Created employee with ID: " + emp1Id);

System.out.println("Created employee with ID: " + emp2Id);

System.out.println("Created employee with ID: " + emp3Id);

// List all employees

System.out.println("\n2. Listing all employees:");

List<Employee> employees = employeeDAO.listEmployees();

for (Employee emp : employees) {

System.out.println(emp);

}

// Update an employee

System.out.println("\n3. Updating employee salary...");

employeeDAO.updateEmployee(emp1Id, 65000.0);

// List employees after update

System.out.println("\n4. Listing employees after update:");

employees = employeeDAO.listEmployees();

for (Employee emp : employees) {

System.out.println(emp);

}

// Delete an employee

System.out.println("\n5. Deleting employee...");

employeeDAO.deleteEmployee(emp2Id);

// List employees after deletion

System.out.println("\n6. Listing employees after deletion:");

employees = employeeDAO.listEmployees();

for (Employee emp : employees) {

System.out.println(emp);

}

System.out.println("\n=== Hibernate Example Completed ===");

System.out.println("Notice the manual session and transaction management required!");

// Shutdown

EmployeeDAO.shutdown();

}

}

**//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.example</groupId>

<artifactId>hibernate-example</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>11</maven.compiler.source>

<maven.compiler.target>11</maven.compiler.target>

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

<hibernate.version>5.6.15.Final</hibernate.version>

</properties>

<dependencies>

<!-- Hibernate Core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>${hibernate.version}</version>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<version>2.1.214</version>

</dependency>

<!-- JUnit for testing -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter</artifactId>

<version>5.9.0</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>11</source>

<target>11</target>

</configuration>

</plugin>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.example.HibernateExample</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**//EmployeeRepository.java**

package com.example;

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

import org.springframework.stereotype.Repository;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

// Custom query methods can be added here

// Spring Data JPA will automatically implement these methods

// Find employees by first name

java.util.List<Employee> findByFirstName(String firstName);

// Find employees by email

Employee findByEmail(String email);

// Find employees with salary greater than specified amount

java.util.List<Employee> findBySalaryGreaterThan(Double salary);

// Find employees by first name and last name

java.util.List<Employee> findByFirstNameAndLastName(String firstName, String lastName);

}

**//EmployeeService.java**

package com.example;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

@Transactional(readOnly = true)

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@Transactional(readOnly = true)

public Optional<Employee> getEmployeeById(Integer id) {

return employeeRepository.findById(id);

}

@Transactional

public Employee updateEmployee(Employee employee) {

return employeeRepository.save(employee);

}

@Transactional

public void deleteEmployee(Integer id) {

employeeRepository.deleteById(id);

}

@Transactional(readOnly = true)

public List<Employee> getEmployeesByFirstName(String firstName) {

return employeeRepository.findByFirstName(firstName);

}

@Transactional(readOnly = true)

public Employee getEmployeeByEmail(String email) {

return employeeRepository.findByEmail(email);

}

@Transactional(readOnly = true)

public List<Employee> getEmployeesWithSalaryGreaterThan(Double salary) {

return employeeRepository.findBySalaryGreaterThan(salary);

}

}

**//SpringDataJpaApplication.java**

package com.example;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.List;

@SpringBootApplication

public class SpringDataJpaApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

public static void main(String[] args) {

SpringApplication.run(SpringDataJpaApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

System.out.println("=== Spring Data JPA Example ===");

System.out.println("Demonstrating automatic transaction and session management\n");

// Create employees

System.out.println("1. Creating employees...");

Employee emp1 = new Employee("John", "Doe", "john.doe@example.com", 50000.0);

Employee emp2 = new Employee("Jane", "Smith", "jane.smith@example.com", 60000.0);

Employee emp3 = new Employee("Bob", "Johnson", "bob.johnson@example.com", 55000.0);

employeeService.addEmployee(emp1);

employeeService.addEmployee(emp2);

employeeService.addEmployee(emp3);

System.out.println("Employees created successfully!");

// List all employees

System.out.println("\n2. Listing all employees:");

List<Employee> employees = employeeService.getAllEmployees();

employees.forEach(System.out::println);

// Update an employee

System.out.println("\n3. Updating employee salary...");

if (!employees.isEmpty()) {

Employee firstEmployee = employees.get(0);

firstEmployee.setSalary(65000.0);

employeeService.updateEmployee(firstEmployee);

System.out.println("Updated employee: " + firstEmployee);

}

// List employees after update

System.out.println("\n4. Listing employees after update:");

employees = employeeService.getAllEmployees();

employees.forEach(System.out::println);

// Delete an employee

System.out.println("\n5. Deleting employee...");

if (employees.size() > 1) {

Employee employeeToDelete = employees.get(1);

employeeService.deleteEmployee(employeeToDelete.getId());

System.out.println("Deleted employee with ID: " + employeeToDelete.getId());

}

// List employees after deletion

System.out.println("\n6. Listing employees after deletion:");

employees = employeeService.getAllEmployees();

employees.forEach(System.out::println);

// Demonstrate custom queries

System.out.println("\n7. Demonstrating custom queries:");

List<Employee> highSalaryEmployees = employeeService.getEmployeesWithSalaryGreaterThan(50000.0);

System.out.println("Employees with salary > 50000:");

highSalaryEmployees.forEach(System.out::println);

System.out.println("\n=== Spring Data JPA Example Completed ===");

System.out.println("Notice how Spring Data JPA handled transactions and sessions automatically!");

}

}

**OUTPUT:**

