Hands on 1: Spring Data JPA - Quick Example

MODEL

```
package com.example.country.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;

   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 + "]";
   }
}
```

REPOSITORY

```
package com.example.country.repository;

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

```
import org.springframework.stereotype.Repository;

import com.example.country.model.country;

@Repository
public interface countryRepo extends JpaRepository<country, String> {
}
```

SERVICE

```
package com.example.country.service;

import java.util.List;

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

import com.example.country.model.country;
import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service
public class countryService {

   @Autowired
   private countryRepo countryRepository;

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

MAIN CLASS

```
package com.example.country;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.example.country.model.country;
import com.example.country.service.countryService;

import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationContext;
```

```
@SpringBootApplication
public class CountryApplication {

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

   public static void main(String[] args) {
      ApplicationContext context = SpringApplication.run(CountryApplication.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");
   }
}
```

Hands on 2: Hibernate XML Config implementation walk through Object to Relational Database Mapping (ORM) in Hibernate XML

Entity Class – Java POJO

Employee.java

```
public class Employee {
    private int id;
    private String firstName;
    private String lastName;
    private int salary;

    // Getter and Setter for id
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    // Getter and Setter for firstName
    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    // Getter and Setter for lastName
    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    // Getter and Setter for salary
    public int getSalary() {
        return salary;
    }

    public void setSalary(int salary) {
        this.salary = salary;
    }
}
```

Hibernate Mapping XML – Employee.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="Employee" table="EMPLOYEE">
      <id name="id" type="int">
         <column name="id"/>
         <generator class="native"/>
      </id>
      <property name="firstName" column="first_name" type="string"/>
      <property name="lastName" column="last_name" type="string"/>
      <property name="salary" column="salary" type="int"/>
   </class>
</hibernate-mapping>
```

End-to-End Hibernate Operations

SessionFactory

- Built once during application startup.
- It holds metadata and manages sessions.
- Created using hibernate.cfg.xml.

```
SessionFactory factory = new Configuration()
        .configure("hibernate.cfg.xml")
        .addResource("Employee.hbm.xml")
        .buildSessionFactory();
```

Session

- Think of it as a connection with the database.
- Created by SessionFactory.

```
Session session = factory.openSession();
```

Transaction

- Used to group DB operations as a single unit (ACID).
- Needed for save, delete, etc.

```
Transaction tx = session.beginTransaction();
```

beginTransaction()

- Starts a new transaction.
- Required before making changes to the DB.

commit()

- Saves all DB changes permanently.

```
tx.commit();
```

rollback()

- Cancels the current transaction (undoes changes).

```
tx.rollback();
```

session.save(Object)

- Saves an object into the DB.

```
Employee emp = new Employee();
emp.setFirstName("John");
emp.setLastName("Doe");
emp.setSalary(5000);
session.save(emp);
```

session.createQuery().list()

- Used for fetching multiple records.

```
List<Employee> list = session.createQuery("FROM Employee").list();
for (Employee e : list) {
   System.out.println(e.getFirstName());
}
```

session.get(Class, id)

- Fetches a record by primary key.
- Returns null if not found.

```
Employee emp = session.get(Employee.class, 1);
```

session.delete(Object)

- Deletes the given entity object.

```
Employee emp = session.get(Employee.class, 1);
session.delete(emp);
```

Hands on 3: Hibernate Annotation Config implementation walk through Object to Relational Mapping (ORM) in Persistence Class Employee

Employee.java – Persistence Class (POJO)

```
import jakarta.persistence.*;

@Entity
@Table(name = "EMPLOYEE")
public class Employee {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private int id;

   @Column(name = "first_name")
   private String firstName;

   @Column(name = "last_name")
   private String lastName;

   @Column(name = "salary")
   private int salary;

   // Getters and setters...
}
```

Annotation	Purpose
@Entity	Marks the class as a Hibernate entity (i.e., table mapped to a class).
@Table	Specifies the DB table name for the entity.
@Id	Identifies the primary key field.
@GeneratedValue	Auto-generates primary key values (based on strategy).
@Column	Maps a field to a DB column and allows specifying the column name.

Hibernate Configuration File (hibernate.cfg.xml)

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
   "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
  <session-factory>

    <!-- Database connection settings -->
    <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>

    <!-- SQL dialect -->
    <property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

    <!-- Show executed SQL -->
    <property name="hibernate.show_sql">true</property>

    <!-- Update the schema automatically -->
    <property name="hibernate.hbm2ddl.auto">update</property>

    <!-- Register annotated class -->
    <mapping class="com.example.Employee"/>

  </session-factory>
</hibernate-configuration>
```

hibernate.connection.driver_class – JDBC driver class (for MySQL: com.mysql.cj.jdbc.Driver)
hibernate.connection.url – JDBC URL of the database
hibernate.connection.username – Database username
hibernate.connection.password – Database password
hibernate.dialect – SQL dialect to use (for MySQL: MySQLDialect or MySQL8Dialect)
hibernate.show_sql – Shows SQL statements in console (true/false)
hibernate.hbm2ddl.auto – Auto schema action (create, update, validate, none)
mapping class – Registers the entity class with Hibernate

Hands on 4: Difference between JPA, Hibernate and Spring Data JPA

JPA (Java Persistence API)

- It's a **specification**, not a framework.
- Defines how Java objects should be mapped to relational databases.
- Provides standard annotations like @Entity, @Id, @OneToMany.
- Needs an implementation (like Hibernate) to work.
- Does not provide any actual persistence code.

Hibernate

- It's a **framework** and a **JPA implementation**.
- Provides the actual code that works behind the JPA interfaces.
- Can work with or without JPA.
- Offers additional features like HQL, caching, and lazy loading.
- More powerful but requires more manual coding than Spring Data JPA.

Spring Data JPA

- It's a **Spring project** that builds on top of JPA.
- Makes database access easier using **repository interfaces**.
- Removes boilerplate—no need to use EntityManager directly.
- Supports auto query generation from method names.
- Ideal for rapid development and cleaner code.

In short:
JPA is the rulebook,
Hibernate follows and implements that rulebook,
Spring Data JPA uses both to simplify your life.

Hands on 5: Implement services for managing Country

MODEL

```
package com.example.country.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;

    // 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 + "]";
    }
}
```

REPOSITORY

```
package com.example.country.repository;

import java.util.List;

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

import com.example.country.model.country;

@Repository
public interface countryRepo extends JpaRepository<country, String> {
    List<country> findByNameContainingIgnoreCase(String name);
```

}

SERVICE

```
package com.example.country.service;

import java.util.List;
import java.util.Optional;

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

import com.example.country.model.country;
import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service
public class countryService {

    @Autowired
    private countryRepo countryRepository;

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

    @Transactional
    public country findCountryByCode(String code) throws Exception {
        Optional<country> result = countryRepository.findById(code);
        if (result.isPresent()) return result.get();
        else throw new Exception("Country not found for code: " + code);
    }

    @Transactional
    public void addCountry(country country) {
        countryRepository.save(country);
    }

    @Transactional
    public void updateCountry(String code, String newName) throws Exception {
        country country = findCountryByCode(code);
        country.setName(newName);
        countryRepository.save(country);
    }

    @Transactional
    public void deleteCountry(String code) {
```

```
        countryRepository.deleteById(code);
    }

    @Transactional
    public List<country> searchCountriesByName(String name) {
        return countryRepository.findByNameContainingIgnoreCase(name);
    }
}
```

MAIN_CLASS

```
package com.example.country;

import java.util.List;

import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

import com.example.country.model.country;
import com.example.country.service.countryService;

@SpringBootApplication
public class CountryApplication {

    private static countryService countryService;

    private static final org.slf4j.Logger LOGGER =
LoggerFactory.getLogger(CountryApplication.class);

    public static void main(String[] args) throws Exception {
        ConfigurableApplicationContext context =
SpringApplication.run(CountryApplication.class, args);
        LOGGER.info("Inside main");

        countryService = context.getBean(countryService.class);

        testGetAllCountries();
        testFindCountryByCode();
        testAddCountry();
        testUpdateCountry();
        testDeleteCountry();
        testSearchCountriesByName();
    }

    private static void testGetAllCountries() {
        LOGGER.info("Start getAll");
```

```
        List<country> list = countryService.getAllCountries();
        LOGGER.debug("countries={}", list);
        LOGGER.info("End getAll");
    }

    private static void testFindCountryByCode() throws Exception {
        LOGGER.info("Start findByCode");
        country c = countryService.findCountryByCode("IN");
        LOGGER.debug("country={}", c);
        LOGGER.info("End findByCode");
    }

    private static void testAddCountry() {
        LOGGER.info("Start add");
        country c = new country();
        c.setCode("ZZ");
        c.setName("Zootopia");
        countryService.addCountry(c);
        LOGGER.info("End add");
    }

    private static void testUpdateCountry() throws Exception {
        LOGGER.info("Start update");
        countryService.updateCountry("ZZ", "Zootopia Updated");
        LOGGER.info("End update");
    }

    private static void testDeleteCountry() {
        LOGGER.info("Start delete");
        countryService.deleteCountry("ZZ");
        LOGGER.info("End delete");
    }

    private static void testSearchCountriesByName() {
        LOGGER.info("Start search");
        List<country> results = countryService.searchCountriesByName("land");
        LOGGER.debug("Matching countries={}", results);
        LOGGER.info("End search");
    }
}
```

Hands on 6: Find a country based on country code

MODEL

```
package com.example.country.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;

    // 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 + "]";
    }
}
```

REPOSITORY

```
package com.example.country.repository;

import java.util.List;

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

import com.example.country.model.country;
```

```
@Repository
public interface countryRepo extends JpaRepository<country, String> {
   List<country> findByNameContainingIgnoreCase(String name);
}
```

SERVICE

```
package com.example.country.service;

import java.util.List;
import java.util.Optional;

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

import com.example.country.model.country;
import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service
public class countryService {

   @Autowired
   private countryRepo countryRepository;

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

   @Transactional
   public country findCountryByCode(String code) throws Exception {
      Optional<country> result = countryRepository.findById(code);
      if (result.isPresent()) return result.get();
      else throw new Exception("Country not found for code: " + code);
   }

   @Transactional
   public void addCountry(country country) {
      countryRepository.save(country);
   }

   @Transactional
   public void updateCountry(String code, String newName) throws Exception {
      country country = findCountryByCode(code);
      country.setName(newName);
      countryRepository.save(country);
   }
```

```
    @Transactional
    public void deleteCountry(String code) {
        countryRepository.deleteById(code);
    }

    @Transactional
    public List<country> searchCountriesByName(String name) {
        return countryRepository.findByNameContainingIgnoreCase(name);
    }
}
```

MAIN_CLASS

```
package com.example.country;

import java.util.List;

import org.slf4j.LoggerFactory;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ConfigurableApplicationContext;

import com.example.country.model.country;
import com.example.country.service.countryService;

@SpringBootApplication
public class CountryApplication {

    private static countryService countryService;

    private static final org.slf4j.Logger LOGGER =
LoggerFactory.getLogger(CountryApplication.class);

    public static void main(String[] args) throws Exception {
        ConfigurableApplicationContext context =
SpringApplication.run(CountryApplication.class, args);
        LOGGER.info("Inside main");

        countryService = context.getBean(countryService.class);

        testGetAllCountries();
        testFindCountryByCode();
        testAddCountry();
        testUpdateCountry();
        testDeleteCountry();
        testSearchCountriesByName();
    }
```

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

    private static void testFindCountryByCode() throws Exception {
        LOGGER.info("Start findByCode");
        country c = countryService.findCountryByCode("IN");
        LOGGER.debug("country={}", c);
        LOGGER.info("End findByCode");
    }

    private static void testAddCountry() {
        LOGGER.info("Start add");
        country c = new country();
        c.setCode("ZZ");
        c.setName("Zootopia");
        countryService.addCountry(c);
        LOGGER.info("End add");
    }

    private static void testUpdateCountry() throws Exception {
        LOGGER.info("Start update");
        countryService.updateCountry("ZZ", "Zootopia Updated");
        LOGGER.info("End update");
    }

    private static void testDeleteCountry() {
        LOGGER.info("Start delete");
        countryService.deleteCountry("ZZ");
        LOGGER.info("End delete");
    }

    private static void testSearchCountriesByName() {
        LOGGER.info("Start search");
        List<country> results = countryService.searchCountriesByName("land");
        LOGGER.debug("Matching countries={}", results);
        LOGGER.info("End search");
    }
}
```