**Objectives**

* Explain the need and benefit of ORM
  + ORM (Object-Relational Mapping), makes it easier to develop code that interacts with database, abstracts the database system, transactionality
    - ORM Pros and Cons - https://blog.bitsrc.io/what-is-an-orm-and-why-you-should-use-it-b2b6f75f5e2a
    - What is ORM? - https://en.wikipedia.org/wiki/Object-relational\_mapping

**Answer:**

**Need and Benefit of ORM**

**Object-Relational Mapping (ORM)** is a programming technique that allows developers to interact with a **relational database** using **object-oriented code**. Instead of writing SQL queries directly, ORM frameworks (like Hibernate or JPA) map database tables to Java classes and rows to objects.

**Need for ORM:**

* **Bridges the Object-Relational Impedance Mismatch**: Databases are relational, while Java uses objects. ORM helps connect these two models seamlessly.
* **Reduces Boilerplate Code**: Eliminates repetitive JDBC code like result set handling and manual mapping.
* **Improves Productivity**: Developers work with familiar objects and let the ORM handle SQL and database interaction.
* **Supports Maintainability**: Centralized entity definitions make the codebase cleaner and easier to update.
* **Manages Transactions**: ORM tools handle transaction boundaries and rollbacks easily.

**Benefits of ORM:**

1. **Automatic SQL Generation**  
   ORM can generate complex SQL queries behind the scenes.
2. **Code Reusability and Clean Structure**  
   Entities can be reused across services and layers.
3. **Database Abstraction**  
   Easily switch between different databases with minimal code change.
4. **Caching and Performance Optimization**  
   Built-in caching mechanisms improve performance.
5. **Relationship Handling**  
   ORM simplifies one-to-many, many-to-many, and other complex relationships.

* Demonstrate the need and benefit of Spring Data JPA
  + Evolution of ORM solutions, Hibernate XML Configuration, Hibernate Annotation Configuration, Spring Data JPA, Hibernate benefits, open source, light weight, database independent query
    - With H2 in memory database - https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa/
    - With MySQL - https://www.mkyong.com/spring-boot/spring-boot-spring-data-jpa-mysql-example/
    - XML Configuration Example -https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm
    - Hibernate Configuration Example -https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm

**Answer:**

### ****Need and Benefit of Spring Data JPA****

#### ****Evolution of ORM Solutions:****

1. **Hibernate with XML Configuration**
   * Early Hibernate projects required verbose XML files (hibernate.cfg.xml) for mapping classes to tables.
   * Tedious and error-prone.
2. **Hibernate with Annotations**
   * Simplified configuration by using Java annotations like @Entity, @Table, etc.
   * Still required manual DAO/Repository layer coding.
3. **Spring Data JPA**
   * A powerful abstraction built on top of JPA (and often Hibernate) within the Spring ecosystem.
   * Automates repository implementation.
   * Integrates seamlessly with Spring Boot.
   * Eliminates the need for writing most of the data access logic manually.

### ****Need for Spring Data JPA:****

* Manually writing CRUD operations for each entity is repetitive.
* Reducing boilerplate and error-prone DAO logic is essential for scalability.
* Need to support **database-agnostic development** with **minimal SQL**.

### ****Benefits of Spring Data JPA:****

1. **Auto-Generated Repository Methods**
   * Example: findByEmail(String email) auto-generates SQL to query based on method name.
2. **Integrated with Spring Boot**
   * Easily integrates with in-memory databases like H2 and relational databases like MySQL.
3. **Support for Pagination, Sorting, and Query DSL**
   * Built-in support for paginated and sorted results using Pageable and Sort.
4. **Minimal Configuration**
   * Spring Boot auto-configures JPA and Hibernate based on application properties.
5. **Database Independent Queries**
   * Uses JPQL and method name queries that are abstracted from SQL dialects.
6. **Open Source & Community Driven**
   * Actively maintained, well-documented, and highly compatible with various databases.

### ****Example with Spring Boot + Spring Data JPA + H2:****

# application.properties

spring.datasource.url=jdbc:h2:mem:testdb

spring.jpa.hibernate.ddl-auto=update

@Entity

public class Product {

@Id @GeneratedValue

private Long id;

private String name;

private double price;

}

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByName(String name);

}

* Explain about core objects of hibernate framework
  + Session Factory, Session, Transaction Factory, Transaction, Connection Provider
    - Hibernate Architecture Reference - https://www.tutorialspoint.com/hibernate/hibernate\_architecture.htm

**Answer:**

**Core Objects of Hibernate Framework**

Hibernate is a lightweight, open-source ORM framework that maps Java objects to relational database tables. Its core functionality is built around a few key components:

**1. SessionFactory**

* A **thread-safe**, heavyweight object used to create Session objects.
* Created once per application and configured via hibernate.cfg.xml or annotations.
* Internally uses database configuration settings, mappings, and caching.

**Example:**

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

**2. Session**

* A lightweight, non-thread-safe object used to **interact with the database**.
* Represents a single unit of work.
* Used to **save, update, delete, and fetch** persistent objects.

**Example:**

Session session = factory.openSession();

**3. TransactionFactory *(Advanced Use)***

* Responsible for creating Transaction instances.
* Used internally by Hibernate for transaction management strategy.

**4. Transaction**

* Manages the atomic unit of work.
* Supports **begin**, **commit**, and **rollback** operations.
* Ensures data integrity and consistency.

**Example:**

Transaction tx = session.beginTransaction();

// ... perform operations ...

tx.commit();

**5. ConnectionProvider**

* A low-level interface responsible for providing **database connections** to Hibernate.
* Can be implemented to use custom connection pools (e.g., HikariCP, C3P0).
* Hibernate uses it internally to obtain JDBC connections.

**Summary Diagram (Conceptual):**

Application

↓

SessionFactory (1 per app)

↓

Session (1 per unit of work)

↓

Transaction (Optional)

↓

Database via JDBC Connection (from ConnectionProvider)

* Explain ORM implementation with Hibernate XML Configuration and Annotation Configuration
  + XML Configuration - persistence class, mapping xml, configuration xml, loading hibernate configuration xml file; Annotation Configuration - persistence class, @Entity, @Table, @Id, @Column, hibernate configuration xml file Loading hibernate configuration and interacting with database get the session factory, open session, begin transaction, commit transaction, close session
    - XML Configuration Example - https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm
    - Hibernate Configuration Example - https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm

**Answer:**

### ****ORM Implementation with Hibernate****

Hibernate supports two major ways of mapping Java classes to database tables:

## 1. ****XML Configuration-Based Implementation****

### Components:

1. **Persistence Class (POJO):**
   * A simple Java class with private fields and public getters/setters.

public class Employee {

private int id;

private String name;

private double salary;

// Getters and Setters

}

1. **Mapping XML (Employee.hbm.xml):**
   * Maps class fields to table columns.

<hibernate-mapping>

<class name="Employee" table="employee">

<id name="id" column="id" />

<property name="name" column="name" />

<property name="salary" column="salary" />

</class>

</hibernate-mapping>

1. **Configuration XML (hibernate.cfg.xml):**
   * Contains database connection details and references the mapping file.

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<mapping resource="Employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

1. **Database Interaction Flow:**

Configuration cfg = new Configuration().configure();

SessionFactory factory = cfg.buildSessionFactory();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Employee emp = new Employee();

emp.setId(1);

emp.setName("John");

emp.setSalary(5000);

session.save(emp);

tx.commit();

session.close();

## 2. ****Annotation Configuration-Based Implementation****

### Components:

1. **Annotated Persistence Class:**

@Entity

@Table(name = "employee")

public class Employee {

@Id

private int id;

@Column(name = "name")

private String name;

@Column(name = "salary")

private double salary;

// Getters and Setters

}

1. **Hibernate Configuration (hibernate.cfg.xml):**
   * Same as XML config, but instead of mapping files, add annotated class.

<hibernate-configuration>

<session-factory>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<mapping class="com.example.Employee"/>

</session-factory>

</hibernate-configuration>

1. **Database Interaction Flow (Same as XML):**

Configuration cfg = new Configuration().configure();

SessionFactory factory = cfg.buildSessionFactory();

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

Employee emp = new Employee();

emp.setId(2);

emp.setName("Alice");

emp.setSalary(6000);

session.save(emp);

tx.commit();

session.close();

* Explain the difference between Java Persistence API, Hibernate and Spring Data JPA
  + JPA (Java Persistence API), JPA is a specification (JSR 338), JPA does not have implementation, Hibernate is one of the implementation for JPA, Hibernate is a ORM tool, Spring Data JPA is an abstraction above Hibernate to remove boiler plate code when persisting data using Hibernate.
    - Difference between Spring Data JPA and Hibernate - https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1
    - Intro to JPA - https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html

**Answer:**

**Difference Between Java Persistence API (JPA), Hibernate, and Spring Data JPA**

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (JSR 338) | Implementation (ORM framework) | Spring abstraction over JPA |
| **Role** | Defines the standard for ORM in Java | Implements JPA + adds extra features | Simplifies data access using JPA/Hibernate |
| **Provided By** | Oracle/Java EE | Red Hat | Spring Framework |
| **Implementation?** | No implementation | Implements JPA (and more) | Uses JPA implementation internally |
| **Boilerplate Code** | Still required | Less, but still some DAO logic | Minimal — auto-generates CRUD and query methods |
| **Example Usage** | @Entity, @Id, EntityManager | SessionFactory, Session, @Entity | JpaRepository, @Entity, method-based queries |

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

* A **standard specification** for object-relational mapping in Java.
* Defines annotations like @Entity, @Table, @Id.
* Provides APIs like EntityManager for persisting objects.
* **Does not** provide its own implementation — needs a provider like Hibernate or EclipseLink.

**Example:**

@Entity

public class Student {

@Id

private int id;

private String name;

}

**2. Hibernate**

* A **popular ORM tool** and **JPA implementation**.
* Provides additional features beyond JPA like:
  + Lazy loading
  + Second-level caching
  + HQL (Hibernate Query Language)
* Can be used with or without JPA.

**Example (Native Hibernate):**

Session session = sessionFactory.openSession();

session.beginTransaction();

session.save(student);

session.getTransaction().commit();

session.close();

**3. Spring Data JPA**

* A **Spring abstraction layer** over JPA.
* Simplifies repository creation by removing boilerplate DAO code.
* Uses Hibernate (or any JPA provider) under the hood.
* Provides out-of-the-box CRUD, pagination, sorting, and method query generation.

**Example:**

public interface StudentRepository extends JpaRepository<Student, Integer> {

List<Student> findByName(String name);

}

**Summary:**

**JPA** is the **standard**, **Hibernate** is a **JPA implementation**, and **Spring Data JPA** is a **Spring-based abstraction** that uses JPA (typically Hibernate) to make data access even easier.

* Demonstrate implementation of DML using Spring Data JPA on a single database table
  + Hibernate log configuration and ddl-auto configuration, JpaRepsitory.findById(), defining Query Methods, JpaRespository.save(), JpaRepository.deleteById()
    - Spring Data JPA Ref Repository methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.core-concepts
    - Query methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#repositories.query-methods

**Answer:**

**Demonstrate Implementation of DML Using Spring Data JPA**

We'll use a Product table to show how to:

* Save a new record
* Fetch a record by ID
* Update a record
* Delete a record

**1. Setup:**

# Database connection (H2 example)

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA settings

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

**2. Entity Class:**

import jakarta.persistence.\*;

@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private double price;

// Getters and Setters

}

**3. Repository Interface:**

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

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByName(String name); // Custom query method

}

**4. Service or Runner Logic:**

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import java.util.Optional;

@Component

public class ProductDemoRunner implements CommandLineRunner {

private final ProductRepository repository;

public ProductDemoRunner(ProductRepository repository) {

this.repository = repository;

}

@Override

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

// INSERT

Product product = new Product();

product.setName("Laptop");

product.setPrice(85000);

repository.save(product); // Save to DB

// SELECT

Optional<Product> result = repository.findById(product.getId());

result.ifPresent(p -> System.out.println("Fetched: " + p.getName()));

// UPDATE

if (result.isPresent()) {

Product updated = result.get();

updated.setPrice(80000);

repository.save(updated); // Save updates

}

// DELETE

repository.deleteById(product.getId());

}

}

**5. Hibernate Console Output (with logs enabled)**

Hibernate: insert into product (id, name, price) values (null, ?, ?)

Hibernate: select product0\_.id as id1\_0\_0\_, ...

Hibernate: update product set price=? where id=?

Hibernate: delete from product where id=?

**Hands on 1**

**Spring Data JPA - Quick Example**   
  
**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"
* Click on menu and select "Spring Boot DevTools", "Spring Data JPA" and "MySQL Driver"
* Click Generate and download the project as zip
* Extract the zip in root folder to Eclipse Workspace
* Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
* Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.

> mysql -u root -p

mysql> create schema ormlearn;

* In orm-learn Eclipse project, open src/main/resources/application.properties and include the below database and log configuration.

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

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 using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
* Include logs for verifying if main() method is called.

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");

}

* Execute the OrmLearnApplication and check in log if main method is called.

SME to walk through the following aspects related to the project created:

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
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

**Country table creation**

* Create a new table country with columns for code and name. For sample, let us insert one country with values 'IN' and 'India' in this table.

create table country(co\_code varchar(2) primary key, co\_name varchar(50));

* Insert couple of records into the table

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Persistence Class - com.cognizant.orm-learn.model.Country**

* Open Eclipse with orm-learn project
* Create new package com.cognizant.orm-learn.model
* Create Country.java, then generate getters, setters and toString() methods.
* Include @Entity and @Table at class level
* Include @Column annotations in each getter method specifying the column name.

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.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

  // toString()

}

*Notes:*

* @Entity is an indicator to Spring Data JPA that it is an entity class for the application
* @Table helps in defining the mapping database table
* @Id helps is defining the primary key
* @Column helps in defining the mapping table column

**Repository Class - com.cognizant.orm-learn.CountryRepository**

* Create new package com.cognizant.orm-learn.repository
* Create new interface named CountryRepository that extends JpaRepository<Country, String>
* Define @Repository annotation at class level

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> {

}

**Service Class - com.cognizant.orm-learn.service.CountryService**

* Create new package com.cognizant.orm-learn.service
* Create new class CountryService
* Include @Service annotation at class level
* Autowire CountryRepository in CountryService
* Include new method getAllCountries() method that returns a list of countries.
* Include @Transactional annotation for this method
* In getAllCountries() method invoke countryRepository.findAll() method and return the result

**Testing in OrmLearnApplication.java**

* Include a static reference to CountryService in OrmLearnApplication class

private static CountryService countryService;

* Define a test method to get all countries from service.

    private static void testGetAllCountries() {

        LOGGER.info("Start");

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

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

        LOGGER.info("End");

    }

* Modify SpringApplication.run() invocation to set the application context and the CountryService reference from the application context.

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

        countryService = context.getBean(CountryService.class);

        testGetAllCountries();

* Execute main method to check if data from ormlearn database is retrieved.

**Code:**

**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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>1.0</version>

<name>SpringDataJpaExample</name>

<description>Demo project for Spring Data JPA and Hibernate&quot;</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.oracle.database.jdbc</groupId>

<artifactId>ojdbc11</artifactId>

<version>23.2.0</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.\*;

*@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 + "]";

}

}

**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 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();

}

}

**OrmLearnApplication.java**

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 org.springframework.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@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");

        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.datasource.url=jdbc:oracle:thin:@//localhost:1521/freepdb1

spring.datasource.username=system

spring.datasource.password="$rutiSLD#07@"

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

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect

logging.level.org.hibernate.SQL=debug

logging.level.com.cognizant=debug

**SQL Table & Query**

CREATE TABLE country (

  co\_code VARCHAR2(2) PRIMARY KEY,

  co\_name VARCHAR2(50)

);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

COMMIT;

**Output:**

****

**Hands on 2**

**Hibernate XML Config implementation walk through**   
  
SME to provide explanation on the sample Hibernate implementation available in the link below:  
https://www.tutorialspoint.com/hibernate/hibernate\_examples.htm  
  
Explanation Topics

* Explain how object to relational database mapping done in hibernate xml configuration file
* Explain about following aspects of implementing the end to end operations in Hibernate:
  + SessionFactory
  + Session
  + Transaction
  + beginTransaction()
  + commit()
  + rollback()
  + session.save()
  + session.createQuery().list()
  + session.get()
  + session.delete()

**Answer:**

## ****Object to Relational Mapping in Hibernate XML Configuration****

In Hibernate, object-relational mapping (ORM) using **XML configuration** means that Java classes (POJOs) are mapped to database tables using .hbm.xml mapping files.

### Example:

<hibernate-mapping>

<class name="Employee" table="EMPLOYEE">

<id name="id" column="ID">

<generator class="native"/>

</id>

<property name="firstName" column="FIRST\_NAME"/>

<property name="lastName" column="LAST\_NAME"/>

<property name="salary" column="SALARY"/>

</class>

</hibernate-mapping>

### Breakdown:

| **XML Element** | **Purpose** |
| --- | --- |
| <class> | Maps a Java class (Employee) to a DB table (EMPLOYEE) |
| <id> | Specifies the primary key field |
| <property> | Maps Java fields to DB columns |

## 2. ****End-to-End Hibernate Workflow Components****

Hibernate operations typically follow a lifecycle:

### a) ****SessionFactory****

* A thread-safe factory for Session objects.
* Created once during application startup from hibernate.cfg.xml.

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

### b) ****Session****

* Represents a single unit of work with the database.
* Used to perform CRUD operations.

Session session = factory.openSession();

### c) ****Transaction****

* Used to group multiple operations into a single atomic unit.
* Ensures either all changes are committed or none are (ACID).

### d) beginTransaction()

* Starts a transaction.

Transaction tx = session.beginTransaction();

### e) commit()

* Finalizes the transaction, making changes permanent.

tx.commit();

### f) rollback()

* Used when an exception occurs to undo all operations in the current transaction.

tx.rollback();

## 3. ****CRUD Operations in Hibernate****

### session.save()

* Saves a new object to the database and returns the identifier.

Employee emp = new Employee("John", "Doe", 50000);

session.save(emp);

### session.get()

* Retrieves an object by primary key. Returns null if not found.

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

### session.createQuery().list()

* Executes an HQL (Hibernate Query Language) query to fetch multiple results.

List<Employee> list = session.createQuery("FROM Employee").list();

### session.delete()

* Deletes an existing object.

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

session.delete(emp);

## Summary Flow Example

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

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

// Create

Employee emp = new Employee("John", "Doe", 50000);

session.save(emp);

// Read

Employee emp2 = session.get(Employee.class, emp.getId());

// Update

emp2.setSalary(55000);

session.update(emp2);

// Delete

session.delete(emp2);

tx.commit();

} catch (Exception e) {

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

e.printStackTrace();

} finally {

session.close();

}

**Hands on 3**

**Hibernate Annotation Config implementation walk through**   
  
SME to provide explanation on the sample Hibernate implementation available in the link below:  
https://www.tutorialspoint.com/hibernate/hibernate\_annotations.htm  
  
Explanation Topics

* Explain how object to relational database mapping done in persistence class file Employee
* Explain about following aspects of implementing the end to end operations in Hibernate:
  + @Entity
  + @Table
  + @Id
  + @GeneratedValue
  + @Column
  + Hibernate Configuration (hibernate.cfg.xml)
    - Dialect
    - Driver
    - Connection URL
    - Username
    - Password

**Answer:**

## 1. ****Object to Relational Mapping in the**** Employee ****Class (Annotations)****

In **annotation-based configuration**, object-relational mapping (ORM) is done using **Java annotations inside the persistence class**, removing the need for separate .hbm.xml files.

### Sample Employee.java:

import javax.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;

// Constructors, getters, setters, toString()

}

## Explanation of Annotations:

| **Annotation** | **Purpose** |
| --- | --- |
| @Entity | Marks this class as a persistent entity mapped to a database table. |
| @Table(name="EMPLOYEE") | Specifies the database table name. |
| @Id | Denotes the primary key field. |
| @GeneratedValue | Specifies how the primary key is generated (e.g., AUTO, IDENTITY, SEQUENCE). |
| @Column | Maps a Java field to a specific column in the table. |

## 2. ****Hibernate Configuration File (****hibernate.cfg.xml****)****

Although we use annotations for mapping, we still need a configuration file to define:

* Database connection
* Hibernate settings
* Classes to be mapped

### Sample 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>

<!-- JDBC Driver -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<!-- Database URL -->

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>

<!-- DB Credentials -->

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<!-- Dialect for DB -->

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

<!-- Show SQL in console -->

<property name="show\_sql">true</property>

<!-- Auto update schema -->

<property name="hbm2ddl.auto">update</property>

<!-- Register annotated class -->

<mapping class="com.example.Employee"/>

</session-factory>

</hibernate-configuration>

## Explanation of Config Properties:

| **Property** | **Purpose** |
| --- | --- |
| hibernate.dialect | Specifies DB-specific SQL syntax support (e.g., MySQLDialect, Oracle12cDialect) |
| hibernate.connection.driver\_class | The JDBC driver for your database |
| hibernate.connection.url | JDBC URL pointing to the database |
| hibernate.connection.username/password | Credentials used to log in |
| show\_sql | Logs executed SQL statements to the console |
| hbm2ddl.auto | Controls schema creation: validate, update, create, create-drop |
| <mapping class="..."> | Registers your entity class |

## 3. Hibernate End-to-End Flow (With Annotations)

public class HibernateDemo {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Employee.class)

.buildSessionFactory();

Session session = factory.openSession();

try {

Transaction tx = session.beginTransaction();

// Create and Save

Employee emp = new Employee("John", "Doe", 50000);

session.save(emp);

// Read

Employee e = session.get(Employee.class, emp.getId());

// Update

e.setSalary(60000);

session.update(e);

// Delete

session.delete(e);

tx.commit();

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

factory.close();

}

}

}

**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA**   
  
Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

Hibernate

* ORM Tool that implements JPA

Spring Data JPA

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

**Refer code snippets below on how the code compares between Hibernate and Spring Data JPA  
Hibernate**

   /\* 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;

   }

**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);

  }

​​​​​​​   
  
**Reference Links:**   
<https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1>   
<https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html>

**Answer:**

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

### What is JPA?

* **JPA (Java Persistence API)** is a **specification (JSR 338)** defined by Java EE for ORM (Object-Relational Mapping).
* It provides standard annotations and APIs for mapping Java objects to relational database tables.
* JPA itself **does not provide an implementation** — it only defines **interfaces and annotations** like @Entity, @Table, @Id, etc.

### Key Features:

* Abstracts data persistence logic from implementation.
* Defines a standardized way to manage entity lifecycle, querying (JPQL), and transactions.
* Requires a **provider** (like Hibernate, EclipseLink, OpenJPA) to actually perform persistence.

## ****2. Hibernate****

### What is Hibernate?

* Hibernate is an **ORM framework** and **the most popular implementation of JPA**.
* It offers additional features beyond JPA, such as **caching, batch processing, lazy/eager loading**, and native SQL support.

### Hibernate vs JPA:

| **Aspect** | **JPA** | **Hibernate** |
| --- | --- | --- |
| Type | Specification | Implementation |
| Ownership | Java EE/Jakarta EE | Independent project |
| Query Language | JPQL | HQL (JPQL-compatible + enhancements) |
| Features | Basic ORM API | Advanced ORM + Hibernate-specific features |

### Hibernate Can Be Used:

* Using **XML-based configuration**
* Using **annotation-based configuration**
* With or without Spring

## ****3. Spring Data JPA****

### What is Spring Data JPA?

* Spring Data JPA is **not a JPA implementation**.
* It is a **Spring module** that builds on top of JPA (via providers like Hibernate) and **eliminates boilerplate code** for common operations (e.g., CRUD).

### Key Advantages:

* Reduces the need to write DAO classes manually.
* Offers a powerful **Repository abstraction**.
* Supports **query derivation** from method names.
* **Integrates easily with Spring Boot** and manages transactions automatically.

## ****4. Code Comparison: Hibernate vs Spring Data JPA****

### Using Hibernate (manual session and transaction handling):

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;

}

You must manually open session, handle transactions, commit/rollback, close session.

### Using Spring Data JPA (cleaner, declarative):

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

java

CopyEdit

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

You only define the repository interface, and Spring handles all session and transaction details internally.

**Hands on 5**

**Implement services for managing Country**   
  
An application requires for features to be implemented with regards to country. These features needs to be supported by implementing them as service using Spring Data JPA.

* Find a country based on country code
* Add new country
* Update country
* Delete country
* Find list of countries matching a partial country name

Before starting the implementation of the above features, there are few configuration and data population that needs to be incorporated. Please refer each topic below and implement the same.   
  
**Explanation for Hibernate table creation configuration**

* Moreover the ddl-auto defines how hibernate behaves if a specific table or column is not present in the database.
  + create - drops existing tables data and structure, then creates new tables
  + validate - check if the table and columns exist or not, throws an exception if a matching table or column is not found
  + update - if a table does not exists, it creates a new table; if a column does not exists, it creates a new column
  + create-drop - creates the table, once all operations are completed, the table is dropped

# Hibernate ddl auto (create, create-drop, update, validate)

spring.jpa.hibernate.ddl-auto=validate

Populate country table

* Delete all the records in Country table and then use the below script to create the actual list of all countries in our world.

insert into country (co\_code, co\_name) values ("AF", "Afghanistan");

insert into country (co\_code, co\_name) values ("AL", "Albania");

insert into country (co\_code, co\_name) values ("DZ", "Algeria");

insert into country (co\_code, co\_name) values ("AS", "American Samoa");

insert into country (co\_code, co\_name) values ("AD", "Andorra");

insert into country (co\_code, co\_name) values ("AO", "Angola");

insert into country (co\_code, co\_name) values ("AI", "Anguilla");

insert into country (co\_code, co\_name) values ("AQ", "Antarctica");

insert into country (co\_code, co\_name) values ("AG", "Antigua and Barbuda");

insert into country (co\_code, co\_name) values ("AR", "Argentina");

insert into country (co\_code, co\_name) values ("AM", "Armenia");

insert into country (co\_code, co\_name) values ("AW", "Aruba");

insert into country (co\_code, co\_name) values ("AU", "Australia");

insert into country (co\_code, co\_name) values ("AT", "Austria");

insert into country (co\_code, co\_name) values ("AZ", "Azerbaijan");

insert into country (co\_code, co\_name) values ("BS", "Bahamas");

insert into country (co\_code, co\_name) values ("BH", "Bahrain");

insert into country (co\_code, co\_name) values ("BD", "Bangladesh");

insert into country (co\_code, co\_name) values ("BB", "Barbados");

insert into country (co\_code, co\_name) values ("BY", "Belarus");

insert into country (co\_code, co\_name) values ("BE", "Belgium");

insert into country (co\_code, co\_name) values ("BZ", "Belize");

insert into country (co\_code, co\_name) values ("BJ", "Benin");

insert into country (co\_code, co\_name) values ("BM", "Bermuda");

insert into country (co\_code, co\_name) values ("BT", "Bhutan");

insert into country (co\_code, co\_name) values ("BO", "Bolivia, Plurinational State of");

insert into country (co\_code, co\_name) values ("BQ", "Bonaire, Sint Eustatius and Saba");

insert into country (co\_code, co\_name) values ("BA", "Bosnia and Herzegovina");

insert into country (co\_code, co\_name) values ("BW", "Botswana");

insert into country (co\_code, co\_name) values ("BV", "Bouvet Island");

insert into country (co\_code, co\_name) values ("BR", "Brazil");

insert into country (co\_code, co\_name) values ("IO", "British Indian Ocean Territory");

insert into country (co\_code, co\_name) values ("BN", "Brunei Darussalam");

insert into country (co\_code, co\_name) values ("BG", "Bulgaria");

insert into country (co\_code, co\_name) values ("BF", "Burkina Faso");

insert into country (co\_code, co\_name) values ("BI", "Burundi");

insert into country (co\_code, co\_name) values ("KH", "Cambodia");

insert into country (co\_code, co\_name) values ("CM", "Cameroon");

insert into country (co\_code, co\_name) values ("CA", "Canada");

insert into country (co\_code, co\_name) values ("CV", "Cape Verde");

insert into country (co\_code, co\_name) values ("KY", "Cayman Islands");

insert into country (co\_code, co\_name) values ("CF", "Central African Republic");

insert into country (co\_code, co\_name) values ("TD", "Chad");

insert into country (co\_code, co\_name) values ("CL", "Chile");

insert into country (co\_code, co\_name) values ("CN", "China");

insert into country (co\_code, co\_name) values ("CX", "Christmas Island");

insert into country (co\_code, co\_name) values ("CC", "Cocos (Keeling) Islands");

insert into country (co\_code, co\_name) values ("CO", "Colombia");

insert into country (co\_code, co\_name) values ("KM", "Comoros");

insert into country (co\_code, co\_name) values ("CG", "Congo");

insert into country (co\_code, co\_name) values ("CD", "Congo, the Democratic Republic of the");

insert into country (co\_code, co\_name) values ("CK", "Cook Islands");

insert into country (co\_code, co\_name) values ("CR", "Costa Rica");

insert into country (co\_code, co\_name) values ("HR", "Croatia");

insert into country (co\_code, co\_name) values ("CU", "Cuba");

insert into country (co\_code, co\_name) values ("CW", "Curaçao");

insert into country (co\_code, co\_name) values ("CY", "Cyprus");

insert into country (co\_code, co\_name) values ("CZ", "Czech Republic");

insert into country (co\_code, co\_name) values ("CI", "Côte d'Ivoire");

insert into country (co\_code, co\_name) values ("DK", "Denmark");

insert into country (co\_code, co\_name) values ("DJ", "Djibouti");

insert into country (co\_code, co\_name) values ("DM", "Dominica");

insert into country (co\_code, co\_name) values ("DO", "Dominican Republic");

insert into country (co\_code, co\_name) values ("EC", "Ecuador");

insert into country (co\_code, co\_name) values ("EG", "Egypt");

insert into country (co\_code, co\_name) values ("SV", "El Salvador");

insert into country (co\_code, co\_name) values ("GQ", "Equatorial Guinea");

insert into country (co\_code, co\_name) values ("ER", "Eritrea");

insert into country (co\_code, co\_name) values ("EE", "Estonia");

insert into country (co\_code, co\_name) values ("ET", "Ethiopia");

insert into country (co\_code, co\_name) values ("FK", "Falkland Islands (Malvinas)");

insert into country (co\_code, co\_name) values ("FO", "Faroe Islands");

insert into country (co\_code, co\_name) values ("FJ", "Fiji");

insert into country (co\_code, co\_name) values ("FI", "Finland");

insert into country (co\_code, co\_name) values ("FR", "France");

insert into country (co\_code, co\_name) values ("GF", "French Guiana");

insert into country (co\_code, co\_name) values ("PF", "French Polynesia");

insert into country (co\_code, co\_name) values ("TF", "French Southern Territories");

insert into country (co\_code, co\_name) values ("GA", "Gabon");

insert into country (co\_code, co\_name) values ("GM", "Gambia");

insert into country (co\_code, co\_name) values ("GE", "Georgia");

insert into country (co\_code, co\_name) values ("DE", "Germany");

insert into country (co\_code, co\_name) values ("GH", "Ghana");

insert into country (co\_code, co\_name) values ("GI", "Gibraltar");

insert into country (co\_code, co\_name) values ("GR", "Greece");

insert into country (co\_code, co\_name) values ("GL", "Greenland");

insert into country (co\_code, co\_name) values ("GD", "Grenada");

insert into country (co\_code, co\_name) values ("GP", "Guadeloupe");

insert into country (co\_code, co\_name) values ("GU", "Guam");

insert into country (co\_code, co\_name) values ("GT", "Guatemala");

insert into country (co\_code, co\_name) values ("GG", "Guernsey");

insert into country (co\_code, co\_name) values ("GN", "Guinea");

insert into country (co\_code, co\_name) values ("GW", "Guinea-Bissau");

insert into country (co\_code, co\_name) values ("GY", "Guyana");

insert into country (co\_code, co\_name) values ("HT", "Haiti");

insert into country (co\_code, co\_name) values ("HM", "Heard Island and McDonald Islands");

insert into country (co\_code, co\_name) values ("VA", "Holy See (Vatican City State)");

insert into country (co\_code, co\_name) values ("HN", "Honduras");

insert into country (co\_code, co\_name) values ("HK", "Hong Kong");

insert into country (co\_code, co\_name) values ("HU", "Hungary");

insert into country (co\_code, co\_name) values ("IS", "Iceland");

insert into country (co\_code, co\_name) values ("IN", "India");

insert into country (co\_code, co\_name) values ("ID", "Indonesia");

insert into country (co\_code, co\_name) values ("IR", "Iran, Islamic Republic of");

insert into country (co\_code, co\_name) values ("IQ", "Iraq");

insert into country (co\_code, co\_name) values ("IE", "Ireland");

insert into country (co\_code, co\_name) values ("IM", "Isle of Man");

insert into country (co\_code, co\_name) values ("IL", "Israel");

insert into country (co\_code, co\_name) values ("IT", "Italy");

insert into country (co\_code, co\_name) values ("JM", "Jamaica");

insert into country (co\_code, co\_name) values ("JP", "Japan");

insert into country (co\_code, co\_name) values ("JE", "Jersey");

insert into country (co\_code, co\_name) values ("JO", "Jordan");

insert into country (co\_code, co\_name) values ("KZ", "Kazakhstan");

insert into country (co\_code, co\_name) values ("KE", "Kenya");

insert into country (co\_code, co\_name) values ("KI", "Kiribati");

insert into country (co\_code, co\_name) values ("KP", "Democratic People's Republic of Korea");

insert into country (co\_code, co\_name) values ("KR", "Republic of Korea");

insert into country (co\_code, co\_name) values ("KW", "Kuwait");

insert into country (co\_code, co\_name) values ("KG", "Kyrgyzstan");

insert into country (co\_code, co\_name) values ("LA", "Lao People's Democratic Republic");

insert into country (co\_code, co\_name) values ("LV", "Latvia");

insert into country (co\_code, co\_name) values ("LB", "Lebanon");

insert into country (co\_code, co\_name) values ("LS", "Lesotho");

insert into country (co\_code, co\_name) values ("LR", "Liberia");

insert into country (co\_code, co\_name) values ("LY", "Libya");

insert into country (co\_code, co\_name) values ("LI", "Liechtenstein");

insert into country (co\_code, co\_name) values ("LT", "Lithuania");

insert into country (co\_code, co\_name) values ("LU", "Luxembourg");

insert into country (co\_code, co\_name) values ("MO", "Macao");

insert into country (co\_code, co\_name) values ("MK", "Macedonia, the Former Yugoslav Republic of");

insert into country (co\_code, co\_name) values ("MG", "Madagascar");

insert into country (co\_code, co\_name) values ("MW", "Malawi");

insert into country (co\_code, co\_name) values ("MY", "Malaysia");

insert into country (co\_code, co\_name) values ("MV", "Maldives");

insert into country (co\_code, co\_name) values ("ML", "Mali");

insert into country (co\_code, co\_name) values ("MT", "Malta");

insert into country (co\_code, co\_name) values ("MH", "Marshall Islands");

insert into country (co\_code, co\_name) values ("MQ", "Martinique");

insert into country (co\_code, co\_name) values ("MR", "Mauritania");

insert into country (co\_code, co\_name) values ("MU", "Mauritius");

insert into country (co\_code, co\_name) values ("YT", "Mayotte");

insert into country (co\_code, co\_name) values ("MX", "Mexico");

insert into country (co\_code, co\_name) values ("FM", "Micronesia, Federated States of");

insert into country (co\_code, co\_name) values ("MD", "Moldova, Republic of");

insert into country (co\_code, co\_name) values ("MC", "Monaco");

insert into country (co\_code, co\_name) values ("MN", "Mongolia");

insert into country (co\_code, co\_name) values ("ME", "Montenegro");

insert into country (co\_code, co\_name) values ("MS", "Montserrat");

insert into country (co\_code, co\_name) values ("MA", "Morocco");

insert into country (co\_code, co\_name) values ("MZ", "Mozambique");

insert into country (co\_code, co\_name) values ("MM", "Myanmar");

insert into country (co\_code, co\_name) values ("NA", "Namibia");

insert into country (co\_code, co\_name) values ("NR", "Nauru");

insert into country (co\_code, co\_name) values ("NP", "Nepal");

insert into country (co\_code, co\_name) values ("NL", "Netherlands");

insert into country (co\_code, co\_name) values ("NC", "New Caledonia");

insert into country (co\_code, co\_name) values ("NZ", "New Zealand");

insert into country (co\_code, co\_name) values ("NI", "Nicaragua");

insert into country (co\_code, co\_name) values ("NE", "Niger");

insert into country (co\_code, co\_name) values ("NG", "Nigeria");

insert into country (co\_code, co\_name) values ("NU", "Niue");

insert into country (co\_code, co\_name) values ("NF", "Norfolk Island");

insert into country (co\_code, co\_name) values ("MP", "Northern Mariana Islands");

insert into country (co\_code, co\_name) values ("NO", "Norway");

insert into country (co\_code, co\_name) values ("OM", "Oman");

insert into country (co\_code, co\_name) values ("PK", "Pakistan");

insert into country (co\_code, co\_name) values ("PW", "Palau");

insert into country (co\_code, co\_name) values ("PS", "Palestine, State of");

insert into country (co\_code, co\_name) values ("PA", "Panama");

insert into country (co\_code, co\_name) values ("PG", "Papua New Guinea");

insert into country (co\_code, co\_name) values ("PY", "Paraguay");

insert into country (co\_code, co\_name) values ("PE", "Peru");

insert into country (co\_code, co\_name) values ("PH", "Philippines");

insert into country (co\_code, co\_name) values ("PN", "Pitcairn");

insert into country (co\_code, co\_name) values ("PL", "Poland");

insert into country (co\_code, co\_name) values ("PT", "Portugal");

insert into country (co\_code, co\_name) values ("PR", "Puerto Rico");

insert into country (co\_code, co\_name) values ("QA", "Qatar");

insert into country (co\_code, co\_name) values ("RO", "Romania");

insert into country (co\_code, co\_name) values ("RU", "Russian Federation");

insert into country (co\_code, co\_name) values ("RW", "Rwanda");

insert into country (co\_code, co\_name) values ("RE", "Réunion");

insert into country (co\_code, co\_name) values ("BL", "Saint Barthélemy");

insert into country (co\_code, co\_name) values ("SH", "Saint Helena, Ascension and Tristan da Cunha");

insert into country (co\_code, co\_name) values ("KN", "Saint Kitts and Nevis");

insert into country (co\_code, co\_name) values ("LC", "Saint Lucia");

insert into country (co\_code, co\_name) values ("MF", "Saint Martin (French part)");

insert into country (co\_code, co\_name) values ("PM", "Saint Pierre and Miquelon");

insert into country (co\_code, co\_name) values ("VC", "Saint Vincent and the Grenadines");

insert into country (co\_code, co\_name) values ("WS", "Samoa");

insert into country (co\_code, co\_name) values ("SM", "San Marino");

insert into country (co\_code, co\_name) values ("ST", "Sao Tome and Principe");

insert into country (co\_code, co\_name) values ("SA", "Saudi Arabia");

insert into country (co\_code, co\_name) values ("SN", "Senegal");

insert into country (co\_code, co\_name) values ("RS", "Serbia");

insert into country (co\_code, co\_name) values ("SC", "Seychelles");

insert into country (co\_code, co\_name) values ("SL", "Sierra Leone");

insert into country (co\_code, co\_name) values ("SG", "Singapore");

insert into country (co\_code, co\_name) values ("SX", "Sint Maarten (Dutch part)");

insert into country (co\_code, co\_name) values ("SK", "Slovakia");

insert into country (co\_code, co\_name) values ("SI", "Slovenia");

insert into country (co\_code, co\_name) values ("SB", "Solomon Islands");

insert into country (co\_code, co\_name) values ("SO", "Somalia");

insert into country (co\_code, co\_name) values ("ZA", "South Africa");

insert into country (co\_code, co\_name) values ("GS", "South Georgia and the South Sandwich Islands");

insert into country (co\_code, co\_name) values ("SS", "South Sudan");

insert into country (co\_code, co\_name) values ("ES", "Spain");

insert into country (co\_code, co\_name) values ("LK", "Sri Lanka");

insert into country (co\_code, co\_name) values ("SD", "Sudan");

insert into country (co\_code, co\_name) values ("SR", "Suriname");

insert into country (co\_code, co\_name) values ("SJ", "Svalbard and Jan Mayen");

insert into country (co\_code, co\_name) values ("SZ", "Swaziland");

insert into country (co\_code, co\_name) values ("SE", "Sweden");

insert into country (co\_code, co\_name) values ("CH", "Switzerland");

insert into country (co\_code, co\_name) values ("SY", "Syrian Arab Republic");

insert into country (co\_code, co\_name) values ("TW", "Taiwan, Province of China");

insert into country (co\_code, co\_name) values ("TJ", "Tajikistan");

insert into country (co\_code, co\_name) values ("TZ", "Tanzania, United Republic of");

insert into country (co\_code, co\_name) values ("TH", "Thailand");

insert into country (co\_code, co\_name) values ("TL", "Timor-Leste");

insert into country (co\_code, co\_name) values ("TG", "Togo");

insert into country (co\_code, co\_name) values ("TK", "Tokelau");

insert into country (co\_code, co\_name) values ("TO", "Tonga");

insert into country (co\_code, co\_name) values ("TT", "Trinidad and Tobago");

insert into country (co\_code, co\_name) values ("TN", "Tunisia");

insert into country (co\_code, co\_name) values ("TR", "Turkey");

insert into country (co\_code, co\_name) values ("TM", "Turkmenistan");

insert into country (co\_code, co\_name) values ("TC", "Turks and Caicos Islands");

insert into country (co\_code, co\_name) values ("TV", "Tuvalu");

insert into country (co\_code, co\_name) values ("UG", "Uganda");

insert into country (co\_code, co\_name) values ("UA", "Ukraine");

insert into country (co\_code, co\_name) values ("AE", "United Arab Emirates");

insert into country (co\_code, co\_name) values ("GB", "United Kingdom");

insert into country (co\_code, co\_name) values ("US", "United States");

insert into country (co\_code, co\_name) values ("UM", "United States Minor Outlying Islands");

insert into country (co\_code, co\_name) values ("UY", "Uruguay");

insert into country (co\_code, co\_name) values ("UZ", "Uzbekistan");

insert into country (co\_code, co\_name) values ("VU", "Vanuatu");

insert into country (co\_code, co\_name) values ("VE", "Venezuela, Bolivarian Republic of");

insert into country (co\_code, co\_name) values ("VN", "Viet Nam");

insert into country (co\_code, co\_name) values ("VG", "Virgin Islands, British");

insert into country (co\_code, co\_name) values ("VI", "Virgin Islands, U.S.");

insert into country (co\_code, co\_name) values ("WF", "Wallis and Futuna");

insert into country (co\_code, co\_name) values ("EH", "Western Sahara");

insert into country (co\_code, co\_name) values ("YE", "Yemen");

insert into country (co\_code, co\_name) values ("ZM", "Zambia");

insert into country (co\_code, co\_name) values ("ZW", "Zimbabwe");

insert into country (co\_code, co\_name) values ("AX", "Åland Islands");

Refer subsequent hands on exercises to implement the features related to country.

**Code:**

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

*@Entity*

*@Table*(name = "country")

public class Country {

*@Id*

private String code;

private String name;

public Country() {}

public Country(String code, String name) {

this.code = code;

this.name = 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 java.util.List;

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

import com.cognizant.ormlearn.model.Country;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String keyword);

}

### CountryService.java

package com.cognizant.ormlearn.service;

import java.util.List;

import java.util.Optional;

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;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public Country findCountryByCode(String code) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(code);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + code);

}

return result.get();

}

*@Transactional*

public void addCountry(Country country) {

countryRepository.save(country);

}

*@Transactional*

public void updateCountry(String code, String name) throws CountryNotFoundException {

Country country = findCountryByCode(code);

country.setName(name);

countryRepository.save(country);

}

*@Transactional*

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

*@Transactional*

public List<Country> findCountriesByPartialName(String keyword) {

return countryRepository.findByNameContainingIgnoreCase(keyword);

}

}

### CountryNotFoundException.java

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

**OrmLearnApplication.java**

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 org.springframework.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@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);

try {

*testCountryService*();

} catch (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

}

private static void testCountryService() throws CountryNotFoundException {

***LOGGER***.info("=== Start Test ===");

Country in = *countryService*.findCountryByCode("IN");

***LOGGER***.debug("Fetched: {}", in);

*countryService*.addCountry(new Country("ZZ", "Zootopia"));

***LOGGER***.debug("Country ZZ added.");

*countryService*.updateCountry("ZZ", "Zootopian Republic");

***LOGGER***.debug("Country ZZ updated.");

*countryService*.deleteCountry("ZZ");

***LOGGER***.debug("Country ZZ deleted.");

List<Country> matched = *countryService*.findCountriesByPartialName("land");

matched.forEach(c -> ***LOGGER***.debug("Matched Country: {}", c));

***LOGGER***.info("=== End Test ===");

}

}

**application.properties**

spring.datasource.url=jdbc:oracle:thin:@//localhost:1521/freepdb1

spring.datasource.username=system

spring.datasource.password="$rutiSLD#07@"

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

spring.jpa.hibernate.ddl-auto=validate

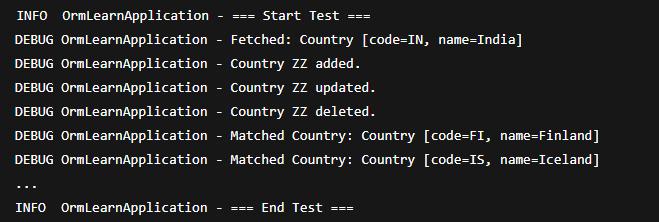
spring.jpa.database-platform=org.hibernate.dialect.Oracle12cDialect

logging.level.org.hibernate.SQL=debug

logging.level.com.cognizant=debug

**Output:**

1. **Find**: "IN"
2. **Add**: "ZZ - Zootopia"
3. **Update**: "ZZ" → "Zootopian Republic"
4. **Delete**: "ZZ"
5. **Partial Name**: Search by "land"

****

**Hands on 6**

**Find a country based on country code** 

* Create new exception class CountryNotFoundException in com.cognizant.spring-learn.service.exception
* Create new method findCountryByCode() in CountryService with @Transactional annotation
* In findCountryByCode() method, perform the following steps:
  + Method signature

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException

* Get the country based on findById() built in method

Optional<Country> result = countryRepository.findById(countryCode);

* From the result, check if a country is found. If not found, throw CountryNotFoundException

if (!result.isPresent())

* Use get() method to return the country fetched.

Country country = result.get();

* Include new test method in OrmLearnApplication to find a country based on country code and compare the country name to check if it is valid.

    private static void getAllCountriesTest() {

        LOGGER.info("Start");

        Country country = countryService.findCountryByCode("IN");

  LOGGER.debug("Country:{}", country);

        LOGGER.info("End");

    }

* Invoke the above method in main() method and test it.

**NOTE:** SME to explain the importance of @Transactional annotation. Spring takes care of creating the Hibernate session and manages the transactionality when executing the service method.

**Code:**

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

*@Entity*

*@Table*(name = "country")

public class Country {

*@Id*

private String code;

private String name;

public Country() {}

public Country(String code, String name) {

this.code = code;

this.name = 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 com.cognizant.ormlearn.model.Country;

public interface CountryRepository extends JpaRepository<Country, String> {}

**CountryNotFoundException.java**

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.Optional;

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;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found");

}

return result.get();

}

}

**OrmLearnApplication.java**

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;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@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);

try {

*getCountryByCodeTest*();

} catch (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

}

private static void getCountryByCodeTest() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country country = *countryService*.findCountryByCode("IN");

***LOGGER***.debug("Country:{}", country);

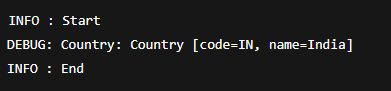
***LOGGER***.info("End");

}

}

**Output:**

**Input: IN**

****

**Input: ZZ**

****

**Hands on 7**

**Add a new country** 

* Create new method in CountryService.

@Transactional

public void addCountry(Country country)

* Invoke save() method of repository to get the country added.

countryRepository.save(country)

* Include new testAddCountry() method in OrmLearnApplication. Perform steps below:
  + Create new instance of country with a new code and name
  + Call countryService.addCountry() passing the country created in the previous step.
  + Invoke countryService.findCountryByCode() passing the same code used when adding a new country
  + Check in the database if the country is added

**Code:**

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.Optional;

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;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found");

}

return result.get();

}

*@Transactional*

public void addCountry(Country country) {

countryRepository.save(country);

}

}

**OrmLearnApplication.java**

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;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@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);

try {

*testAddCountry*();

} catch (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

}

private static void getCountryByCodeTest() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country country = *countryService*.findCountryByCode("IN");

***LOGGER***.debug("Country:{}", country);

***LOGGER***.info("End");

}

private static void testAddCountry() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country newCountry = new Country("NP", "Nepal");

*countryService*.addCountry(newCountry);

Country added = *countryService*.findCountryByCode("NP");

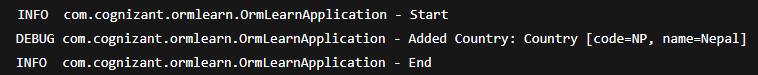
***LOGGER***.debug("Added Country: {}", added);

***LOGGER***.info("End");

}

}

**Output:**

****

**Hands on 8**

**Update a country based on code** 

* Create a new method updateCountry() in CountryService with parameters code and name. Annotate this method with @Transactional. Implement following steps in this method.
  + Get the reference of the country using findById() method in repository
  + In the country reference obtained, update the name of country using setter method
  + Call countryRepository.save() method to update the name
* Include new test method in OrmLearnApplication, which invokes updateCountry() method in CountryService passing a country's code and different name for the country.
* Check in database table if name is modified.

**Code:**

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.Optional;

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;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found");

}

return result.get();

}

*@Transactional*

public void addCountry(Country country) {

countryRepository.save(country);

}

// ✅ Hands-on 8: Update country

*@Transactional*

public void updateCountry(String code, String newName) throws CountryNotFoundException {

Optional<Country> optionalCountry = countryRepository.findById(code);

if (!optionalCountry.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + code);

}

Country country = optionalCountry.get();

country.setName(newName); // update the name

countryRepository.save(country); // save the updated entity

}

}

**OrmLearnApplication.java**

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;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@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);

try {

// testAddCountry(); // From Hands-on 7

// getCountryByCodeTest(); // From Hands-on 6

*testUpdateCountry*(); // ✅ Hands-on 8

} catch (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

}

private static void getCountryByCodeTest() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country country = *countryService*.findCountryByCode("IN");

***LOGGER***.debug("Country:{}", country);

***LOGGER***.info("End");

}

private static void testAddCountry() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country newCountry = new Country("NP", "Nepal");

*countryService*.addCountry(newCountry);

Country added = *countryService*.findCountryByCode("NP");

***LOGGER***.debug("Added Country: {}", added);

***LOGGER***.info("End");

}

// ✅ Hands-on 8 test

private static void testUpdateCountry() throws CountryNotFoundException {

***LOGGER***.info("Start");

*countryService*.updateCountry("NP", "Federal Democratic Republic of Nepal");

Country updated = *countryService*.findCountryByCode("NP");

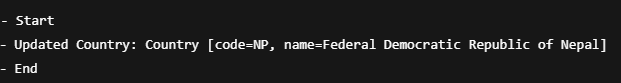
***LOGGER***.debug("Updated Country: {}", updated);

***LOGGER***.info("End");

}

}

**Output:**

****

**Hands on 9**

**Delete a country based on code** 

* Create new method deleteCountry() in CountryService. Annotate this method with @Transactional.
* In deleteCountry() method call deleteById() method of repository.
* Include new test method in OrmLearnApplication with following steps
  + Call the delete method based on the country code during the add country hands on
* Check in database if the country is deleted

**Code:**

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.Optional;

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;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found");

}

return result.get();

}

*@Transactional*

public void addCountry(Country country) {

countryRepository.save(country);

}

*@Transactional*

public void updateCountry(String code, String newName) throws CountryNotFoundException {

Optional<Country> optionalCountry = countryRepository.findById(code);

if (!optionalCountry.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + code);

}

Country country = optionalCountry.get();

country.setName(newName);

countryRepository.save(country);

}

// ✅ Hands-on 9: Delete country

*@Transactional*

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

}

**OrmLearnApplication.java**

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;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.service.exception.CountryNotFoundException;

*@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);

try {

// getCountryByCodeTest(); // Hands-on 6

// testAddCountry(); // Hands-on 7

// testUpdateCountry(); // Hands-on 8

*testDeleteCountry*();

} catch (CountryNotFoundException e) {

***LOGGER***.error("Exception: {}", e.getMessage());

}

}

private static void getCountryByCodeTest() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country country = *countryService*.findCountryByCode("IN");

***LOGGER***.debug("Country: {}", country);

***LOGGER***.info("End");

}

private static void testAddCountry() throws CountryNotFoundException {

***LOGGER***.info("Start");

Country newCountry = new Country("NP", "Nepal");

*countryService*.addCountry(newCountry);

Country added = *countryService*.findCountryByCode("NP");

***LOGGER***.debug("Added Country: {}", added);

***LOGGER***.info("End");

}

private static void testUpdateCountry() throws CountryNotFoundException {

***LOGGER***.info("Start");

*countryService*.updateCountry("NP", "Federal Democratic Republic of Nepal");

Country updated = *countryService*.findCountryByCode("NP");

***LOGGER***.debug("Updated Country: {}", updated);

***LOGGER***.info("End");

}

private static void testDeleteCountry() {

***LOGGER***.info("Start");

*countryService*.deleteCountry("NP");

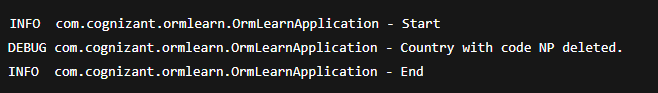
***LOGGER***.debug("Country with code NP deleted.");

***LOGGER***.info("End");

}

}

**Output:**

****