**Hands on 1**

**Spring Data JPA - Quick Example**

**src/main/jacva**

**package:com.cognizant.ormlearn**

**OrmLearnApllication.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) {

// Start Spring Boot and load all beans

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

// Get the CountryService bean (Spring will inject it)

*countryService* = context.getBean(CountryService.class);

***LOGGER***.info("Inside main");

*testGetAllCountries*(); // test method

}

private static void testGetAllCountries() {

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

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

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

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

}

}

**Package: com.cognizant.ormlearn.model**

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

*@Entity*

*@Table*(name = "country")

public class Country {

*@Id*

*@Column*(name = "co\_code")

private String code;

*@Column*(name = "co\_name")

private String name;

// Getters and setters

*@Override*

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**Package: com.cognizant.ormlearn.repository**

**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> {

}

**Package:com.cognizant.ormlearn.service**

**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();

}

}

**Package: src/main/resources**

**application.properties**

spring.application.name=orm-learn

# --- Database connection ---

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 settings ---

spring.jpa.hibernate.ddl-auto=validate

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

# --- Logging ---

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

**pom.xml**

spring.application.name=orm-learn

# --- Database connection ---

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 settings ---

spring.jpa.hibernate.ddl-auto=validate

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

# --- Logging ---

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

**MySQL:**

CREATE DATABASE ormlearn;

USE ormlearn;

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

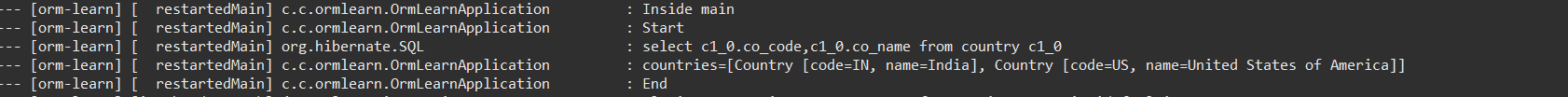
co\_name VARCHAR(50)

);

INSERT INTO country VALUES

('IN', 'India'),

('US', 'United States of America');

**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**

* 1. **• Explain about following aspects of implementing the end to end operations in Hibernate: o SessionFactory**
  2. **o Session**
  3. **o Transaction**
  4. **o beginTransaction()**
  5. **o commit()**
  6. **o rollback()**
  7. **o session.save()**
  8. **o session.createQuery().list()**
  9. **o session.get()**
  10. **o session.delete()**

**Answers:**

**Explain how object to relational database mapping done in hibernate xml configuration file**

Hibernate uses XML configuration files to map Java classes (objects) to relational database tables. This is called ORM (Object-Relational Mapping).

There are two key XML files:

1. hibernate.cfg.xml – The main Hibernate configuration file
2. <ClassName>.hbm.xml – Class-specific mapping file

**Example:**

**hibernate.cfg.xml**

<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>

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

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

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

</session-factory>

</hibernate-configuration>

**Employee.hbm.xml**

<hibernate-mapping>

<class name="com.example.Employee" table="employee">

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

<generator class="increment"/>

</id>

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

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

</class>

</hibernate-mapping>

**Explanation of Core Hibernate Components and Methods**

**1. SessionFactory**

* **A heavyweight object that is created once during application startup.**
* **It is used to create Session objects.**
* **Configuration is loaded from hibernate.cfg.xml.**

**2. Session**

* **A lightweight object used to interact with the database.**
* **All CRUD operations are performed using the Session object.**

**3. Transaction**

* **Used to group one or more operations into a single unit of work.**
* **Ensures atomicity (either all succeed or none do).**

**4. beginTransaction()**

* **Starts a new database transaction.**

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

**5. commit()**

* **Commits the current transaction to the database (makes changes permanent).**

**tx.commit();**

**6. rollback()**

* **Reverts the changes if something goes wrong during the transaction.**

**tx.rollback();**

**7. session.save(Object)**

* **Persists a new object into the database.**

**Employee emp = new Employee();**

**emp.setName("John");**

**emp.setSalary(50000);**

**session.save(emp);**

**8. session.createQuery().list()**

* **Runs HQL (Hibernate Query Language) to fetch data.**

**List<Employee> employees = session.createQuery("from Employee").list();**

**9. session.get(Class, id)**

* **Fetches a record by primary key.**

**Employee e = session.get(Employee.class, 1);**

**10. session.delete(Object)**

* **Deletes an object from the database.**

**session.delete(e);**

**Hands-On:3**

**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:**

**o @Entity**

* + **o @Table**
  + **o @Id**
  + **o @GeneratedValue**
  + **o @Column**
  + **o Hibernate Configuration (hibernate.cfg.xml) ▪ Dialect**
  + **▪ Driver**
  + **▪ Connection URL**
  + **▪ Username**
  + **▪ Password**

**Answers:**

In annotation-based Hibernate, the **Java class itself** (e.g., Employee) is used for ORM mapping, replacing the need for .hbm.xml files.

Hibernate uses **JPA (Java Persistence API)** annotations directly on the class and its fields.

**Example:**

**Employee.java**

import jakarta.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "name")

private String name;

@Column(name = "salary")

private double salary;

// Getters and setters

}

**Explanation of Annotations:**

**@Entity**

* Marks the class as a persistent entity (i.e., maps to a DB table).
* Required for Hibernate to manage the class as a table

**@Table(name = "employee")**

* Maps the class to a specific **table name** in the database.
* Optional if the table name matches the class name.

**@Id**

* Marks a field as the **primary key**.

**@GeneratedValue**

* Tells Hibernate to **automatically generate** the primary key value.
* Common strategies:
  + AUTO
  + IDENTITY (auto-increment in MySQL)
  + SEQUENCE
  + TABLE

**@Column(name = "column\_name")**

* Maps a Java field to a specific **column** in the table.
* Optional if field name matches column name.

**Hibernate Configuration (hibernate.cfg.xml)**

Even with annotations, you still need to use the hibernate.cfg.xml file to configure the database connection and dialect

**Sample hibernate.cfg.xml**

<hibernate-configuration>

<session-factory>

<!-- JDBC Properties -->

<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>

<!-- Hibernate Properties -->

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

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

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

<!-- Add annotated class -->

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

</session-factory>

</hibernate-configuration>

|  |  |
| --- | --- |
| **hibernate.dialect** | Tells Hibernate which SQL dialect to use (e.g., MySQLDialect, OracleDialect) |

|  |  |
| --- | --- |
| **hibernate.connection.driver\_class** | JDBC driver class for your database (e.g., com.mysql.cj.jdbc.Driver) |

|  |  |
| --- | --- |
| **hibernate.connection.url** | Full JDBC URL to connect to your database |

|  |  |
| --- | --- |
| **hibernate.connection.username** | Username used to connect to the DB |

|  |  |
| --- | --- |
| **hibernate.connection.password** | Password used to connect to the DB |

**Hands-On 4:**

**🔹 Introduction**

In Java, accessing and managing data in relational databases is simplified using various persistence technologies. The three main tools in this space are:

* Java Persistence API (JPA) — Specification
* Hibernate — Implementation of JPA
* Spring Data JPA — Abstraction over JPA that reduces boilerplate

**🔹 Java Persistence API (JPA)**

* JPA is a standard specification defined by JSR 338.
* It provides APIs to manage relational data in Java applications.
* JPA is just an interface and does not include an implementation.
* Developers need to use an implementation like Hibernate, EclipseLink, etc.

🧩 Think of JPA as a "contract" that others (like Hibernate) must implement.

**🔹 Hibernate**

* Hibernate is a popular ORM (Object Relational Mapping) tool.
* It provides the actual implementation of JPA.
* Hibernate includes both:
  + JPA features (via javax.persistence / jakarta.persistence)
  + Extra features not covered by JPA (e.g., caching, custom queries)
* Requires manual session management and boilerplate code**.**

**🔹 Spring Data JPA**

* Spring Data JPA is part of the Spring Framework.
* It does not implement JPA itself but provides an abstraction layer over JPA.
* It uses JPA providers like Hibernate underneath.
* Automatically handles:
  + Repository interfaces
  + Common queries
  + Transactions
* Minimizes the need for boilerplate code.

**🔍 Summary Table: JPA vs Hibernate vs Spring Data JPA**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | **Specification (API only)** | **ORM Tool, JPA implementation** | **Abstraction over JPA (Framework helper)** |
| **Provides Implementation** | **❌ No** | **✅ Yes** | **❌ No (uses Hibernate internally)** |
| **Boilerplate Code** | **Medium** | **High** | **Low** |
| **Transactions** | **Manual or Annotation-Based** | **Manual** | **Managed automatically** |
| **Ease of Use** | **Moderate** | **Moderate** | **Easy** |

**🔸 Code Comparison**

**✅ Hibernate Example**

public Integer addEmployee(Employee employee) {

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee); // Save entity

tx.commit(); // Commit transaction

} catch (HibernateException e) {

if (tx != null) tx.rollback(); // Rollback on error

e.printStackTrace();

} finally {

session.close(); // Close session

}

return employeeID;

}

**✅ Spring Data JPA Example**

EmployeeRepository.java

java

Copy code

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

java

Copy code

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee); // No session, no transaction management needed

}

**🔍 Key Differences in Code**

| **Aspect** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- |
| **Session Handling** | **Manual via Session** | **Handled internally** |
| **Transactions** | **Manual via Transaction** | **Handled via @Transactional** |
| **Repository Layer** | **Must write manually** | **Auto-generated by Spring** |
| **Code Length** | **Longer with error handling** | **Short and clean** |

**🔹 Conclusion**

* JPA provides a standard API for persistence.
* Hibernate is a powerful JPA implementation.
* Spring Data JPA builds on top of JPA and Hibernate to reduce repetitive code.
* Using Spring Data JPA simplifies development by allowing developers to focus on business logic while Spring handles the rest.

**Hands on 5**

**Implement services for managing Country**

**Configuration:**

**application.properties**

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

spring.datasource.username=root

spring.datasource.password=your\_password

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.show-sql=true

**Country.java**

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters, Setters, Constructors, toString()

}

**CountryRepository.java**

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String namePart); }

**CountryService.java**

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public Country findCountryByCode(String code) { return countryRepository.findById(code).orElse(null); }

public Country addCountry(Country country) { return countryRepository.save(country); }

public Country updateCountry(Country country) { return countryRepository.save(country); }

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

public List<Country> findCountriesByPartialName(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

}

**MainApplication.java**

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private CountryService countryService;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

System.out.println("Inside main");

}

@Override

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

System.out.println("Start");

List<Country> countries = countryService.findCountriesByPartialName("in");

System.out.println("countries=" + countries);

System.out.println("End");

}

}

**SQL Data Setup:**

INSERT INTO country (co\_code, co\_name) VALUES

("AF", "Afghanistan"), ("AL", "Albania"), ("DZ", "Algeria"), ("AS", "American Samoa"),

("AD", "Andorra"), ("AO", "Angola"), ("AI", "Anguilla"), ("AQ", "Antarctica"),

("AG", "Antigua and Barbuda"), ("AR", "Argentina"), ("AM", "Armenia"), ("AW", "Aruba"),

("AU", "Australia"), ("AT", "Austria"), ("AZ", "Azerbaijan"), ("BS", "Bahamas"),

("BH", "Bahrain"), ("BD", "Bangladesh"), ("BB", "Barbados"), ("BY", "Belarus"),

("BE", "Belgium"), ("BZ", "Belize"), ("BJ", "Benin"), ("BM", "Bermuda"),

("BT", "Bhutan"), ("BO", "Bolivia, Plurinational State of"), ("BQ", "Bonaire, Sint Eustatius and Saba"),

("BA", "Bosnia and Herzegovina"), ("BW", "Botswana"), ("BV", "Bouvet Island"),

("BR", "Brazil"), ("IO", "British Indian Ocean Territory"), ("BN", "Brunei Darussalam"),

("BG", "Bulgaria"), ("BF", "Burkina Faso"), ("BI", "Burundi"), ("KH", "Cambodia"),

("CM", "Cameroon"), ("CA", "Canada"), ("CV", "Cape Verde"), ("KY", "Cayman Islands"),

("CF", "Central African Republic"), ("TD", "Chad"), ("CL", "Chile"), ("CN", "China"),

("CX", "Christmas Island"), ("CC", "Cocos (Keeling) Islands"), ("CO", "Colombia"),

("KM", "Comoros"), ("CG", "Congo"), ("CD", "Congo, the Democratic Republic of the"),

("CK", "Cook Islands"), ("CR", "Costa Rica"), ("HR", "Croatia"), ("CU", "Cuba"),

("CW", "Curaçao"), ("CY", "Cyprus"), ("CZ", "Czech Republic"), ("CI", "Côte d'Ivoire"),

("DK", "Denmark"), ("DJ", "Djibouti"), ("DM", "Dominica"), ("DO", "Dominican Republic"),

("EC", "Ecuador"), ("EG", "Egypt"), ("SV", "El Salvador"), ("GQ", "Equatorial Guinea"),

("ER", "Eritrea"), ("EE", "Estonia"), ("ET", "Ethiopia"), ("FK", "Falkland Islands (Malvinas)"),

("FO", "Faroe Islands"), ("FJ", "Fiji"), ("FI", "Finland"), ("FR", "France"),

("GF", "French Guiana"), ("PF", "French Polynesia"), ("TF", "French Southern Territories"),

("GA", "Gabon"), ("GM", "Gambia"), ("GE", "Georgia"), ("DE", "Germany"),

("GH", "Ghana"), ("GI", "Gibraltar"), ("GR", "Greece"), ("GL", "Greenland"),

("GD", "Grenada"), ("GP", "Guadeloupe"), ("GU", "Guam"), ("GT", "Guatemala"),

("GG", "Guernsey"), ("GN", "Guinea"), ("GW", "Guinea-Bissau"), ("GY", "Guyana"),

("HT", "Haiti"), ("HM", "Heard Island and McDonald Islands"), ("VA", "Holy See (Vatican City State)"),

("HN", "Honduras"), ("HK", "Hong Kong"), ("HU", "Hungary"), ("IS", "Iceland"),

("IN", "India"), ("ID", "Indonesia"), ("IR", "Iran, Islamic Republic of"),

("IQ", "Iraq"), ("IE", "Ireland"), ("IM", "Isle of Man"), ("IL", "Israel"),

("IT", "Italy"), ("JM", "Jamaica"), ("JP", "Japan"), ("JE", "Jersey"),

("JO", "Jordan"), ("KZ", "Kazakhstan"), ("KE", "Kenya"), ("KI", "Kiribati"),

("KP", "Democratic People's Republic of Korea"), ("KR", "Republic of Korea"),

("KW", "Kuwait"), ("KG", "Kyrgyzstan"), ("LA", "Lao People's Democratic Republic"),

("LV", "Latvia"), ("LB", "Lebanon"), ("LS", "Lesotho"), ("LR", "Liberia"),

("LY", "Libya"), ("LI", "Liechtenstein"), ("LT", "Lithuania"), ("LU", "Luxembourg"),

("MO", "Macao"), ("MK", "Macedonia, the Former Yugoslav Republic of"),

("MG", "Madagascar"), ("MW", "Malawi"), ("MY", "Malaysia"), ("MV", "Maldives"),

("ML", "Mali"), ("MT", "Malta"), ("MH", "Marshall Islands"), ("MQ", "Martinique"),

("MR", "Mauritania"), ("MU", "Mauritius"), ("YT", "Mayotte"), ("MX", "Mexico"),

("FM", "Micronesia, Federated States of"), ("MD", "Moldova, Republic of"),

("MC", "Monaco"), ("MN", "Mongolia"), ("ME", "Montenegro"), ("MS", "Montserrat"),

("MA", "Morocco"), ("MZ", "Mozambique"), ("MM", "Myanmar"), ("NA", "Namibia"),

("NR", "Nauru"), ("NP", "Nepal"), ("NL", "Netherlands"), ("NC", "New Caledonia"),

("NZ", "New Zealand"), ("NI", "Nicaragua"), ("NE", "Niger"), ("NG", "Nigeria"),

("NU", "Niue"), ("NF", "Norfolk Island"), ("MP", "Northern Mariana Islands"),

("NO", "Norway"), ("OM", "Oman"), ("PK", "Pakistan"), ("PW", "Palau"),

("PS", "Palestine, State of"), ("PA", "Panama"), ("PG", "Papua New Guinea"),

("PY", "Paraguay"), ("PE", "Peru"), ("PH", "Philippines"), ("PN", "Pitcairn"),

("PL", "Poland"), ("PT", "Portugal"), ("PR", "Puerto Rico"), ("QA", "Qatar"),

("RO", "Romania"), ("RU", "Russian Federation"), ("RW", "Rwanda"), ("RE", "Réunion"),

("BL", "Saint Barthélemy"), ("SH", "Saint Helena, Ascension and Tristan da Cunha"),

("KN", "Saint Kitts and Nevis"), ("LC", "Saint Lucia"), ("MF", "Saint Martin (French part)"),

("PM", "Saint Pierre and Miquelon"), ("VC", "Saint Vincent and the Grenadines"),

("WS", "Samoa"), ("SM", "San Marino"), ("ST", "Sao Tome and Principe"),

("SA", "Saudi Arabia"), ("SN", "Senegal"), ("RS", "Serbia"), ("SC", "Seychelles"),

("SL", "Sierra Leone"), ("SG", "Singapore"), ("SX", "Sint Maarten (Dutch part)"),

("SK", "Slovakia"), ("SI", "Slovenia"), ("SB", "Solomon Islands"), ("SO", "Somalia"),

("ZA", "South Africa"), ("GS", "South Georgia and the South Sandwich Islands"),

("SS", "South Sudan"), ("ES", "Spain"), ("LK", "Sri Lanka"), ("SD", "Sudan"),

("SR", "Suriname"), ("SJ", "Svalbard and Jan Mayen"), ("SZ", "Swaziland"),

("SE", "Sweden"), ("CH", "Switzerland"), ("SY", "Syrian Arab Republic"),

("TW", "Taiwan, Province of China"), ("TJ", "Tajikistan"),

("TZ", "Tanzania, United Republic of"), ("TH", "Thailand"),

("TL", "Timor-Leste"), ("TG", "Togo"), ("TK", "Tokelau"), ("TO", "Tonga"),

("TT", "Trinidad and Tobago"), ("TN", "Tunisia"), ("TR", "Turkey"),

("TM", "Turkmenistan"), ("TC", "Turks and Caicos Islands"), ("TV", "Tuvalu"),

("UG", "Uganda"), ("UA", "Ukraine"), ("AE", "United Arab Emirates"),

("GB", "United Kingdom"), ("US", "United States"),

("UM", "United States Minor Outlying Islands"), ("UY", "Uruguay"),

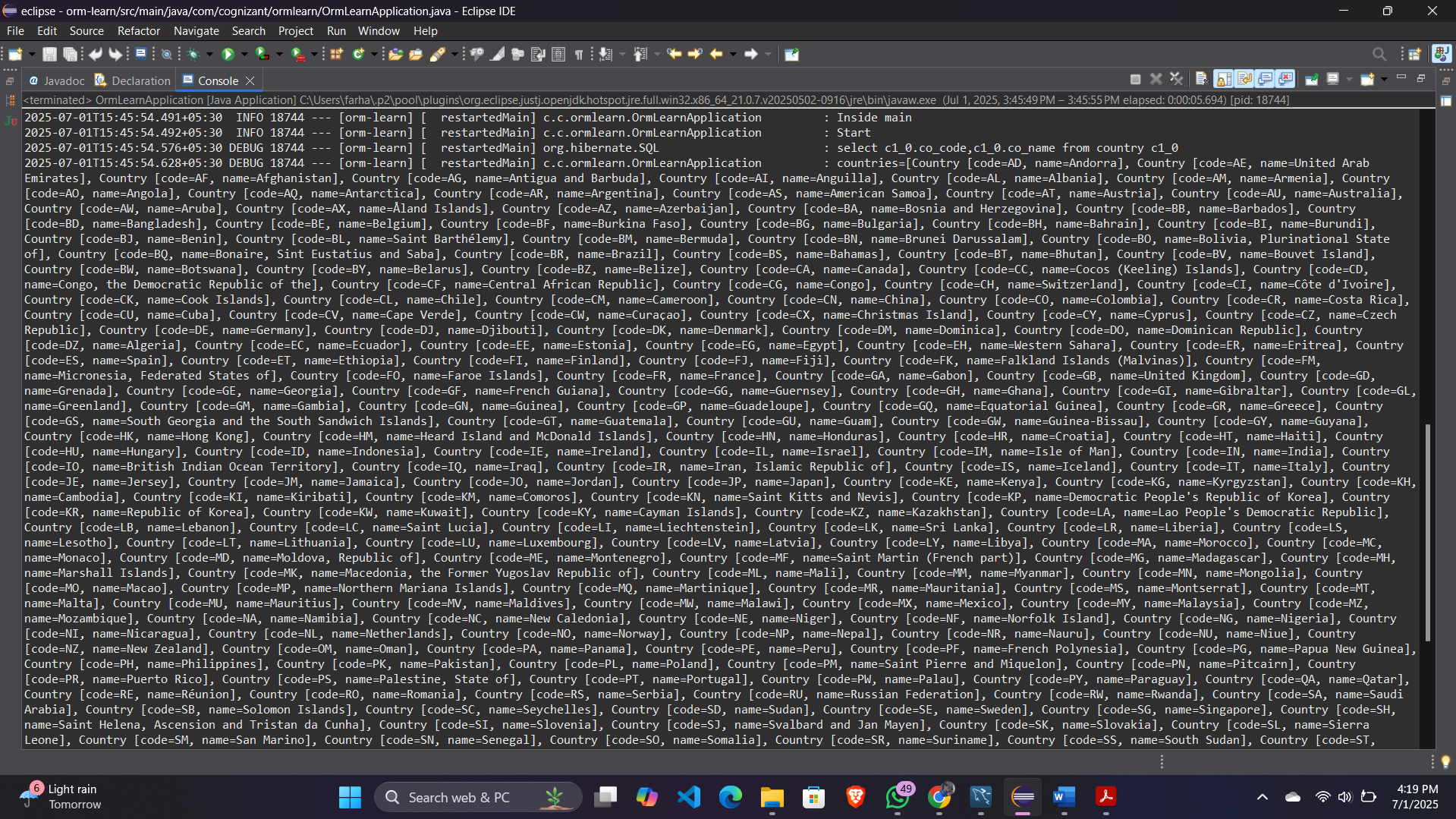
("UZ", "Uzbekistan"), ("VU", "Vanuatu"), ("VE", "Venezuela, Bolivarian Republic of"),

("VN", "Viet Nam"), ("VG", "Virgin Islands, British"), ("VI", "Virgin Islands, U.S."),

("WF", "Wallis and Futuna"), ("EH", "Western Sahara"), ("YE", "Yemen"),

("ZM", "Zambia"), ("ZW", "Zimbabwe"), ("AX", "Åland Islands");

**Output:**





**Hands-On 6:**

**Package:com.cognizant.ormlearn**

**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);

***LOGGER***.info("Inside main");

*countryService* = context.getBean(CountryService.class);

*testGetAllCountries*();

*testFindCountryByCode*();

}

private static void testGetAllCountries() {

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

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

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

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

}

private static void testFindCountryByCode() {

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

try {

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

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

} catch (CountryNotFoundException e) {

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

}

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

}

}

**Package: com.cognizant.ormlearn.model**

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

*@Entity*

*@Table*(name = "country")

public class Country {

*@Id*

*@Column*(name = "co\_code")

private String code;

*@Column*(name = "co\_name")

private String name;

// Getters and setters

*@Override*

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**Package: com.cognizant.ormlearn.repository**

**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> {

Country findByCode(String code);

}

**Package:com.cognizant.ormlearn.service**

**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;

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

*@Transactional*

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

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

if (!result.isPresent()) {

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

}

return result.get();

}

}

**Package: com.cognizant.ormlearn.service.exception**

**CountryNotFoundException.java**

package com.cognizant.ormlearn.service.exception;

public class CountryNotFoundException extends Exception {

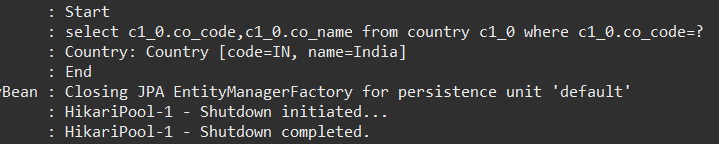
public CountryNotFoundException(String message) {

super(message);

}

}

**Output:**

****

**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)**

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

**CountryService.Repository**

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(readOnly = true)

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

@Transactional(readOnly = true)

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

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

if (!result.isPresent()) {

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

}

return result.get();

}

@Transactional

public void addCountry(Country country) {

countryRepository.save(country);

}

}

**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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

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

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class OrmLearnApplication {

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

private static CountryService countryService;

public static void main(String[] args) throws CountryNotFoundException {

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

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

testFindCountryByCode();

testAddCountry();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

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

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

LOGGER.info("End");

}

private static void testFindCountryByCode() 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 country = new Country();

country.setCode("XY");

country.setName("Xyland");

countryService.addCountry(country);

Country result = countryService.findCountryByCode("XY");

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

LOGGER.info("End");

}

}

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

import jakarta.persistence.Column;

*@Entity*

*@Table*(name = "country")

public class Country {

*@Id*

*@Column*(name = "co\_code")

private String code;

*@Column*(name = "co\_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*

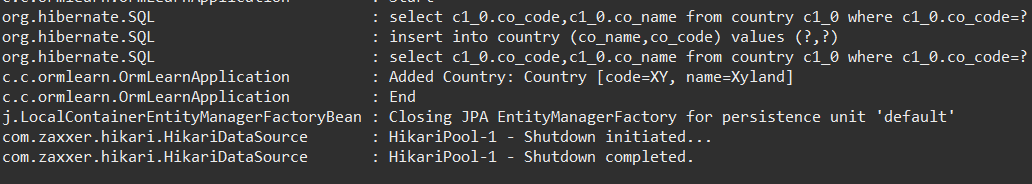
public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**Output:**

****

**Hands on 8**

**Update a country based on code**

**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) throws CountryNotFoundException {

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

*countryService* = context.getBean(CountryService.class);

***LOGGER***.info("Inside main");

// Uncomment to test specific hands-on

// testAddCountry();

*testUpdateCountry*();

}

private static void testUpdateCountry() throws CountryNotFoundException {

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

// Update name of an existing country (e.g., "IN" to "Bharat")

*countryService*.updateCountry("IN", "Bharat");

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

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

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

}

private static void testAddCountry() throws CountryNotFoundException {

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

Country newCountry = new Country();

newCountry.setCode("ZZ");

newCountry.setName("Testland");

*countryService*.addCountry(newCountry);

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

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

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

}

}

**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 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 newName) throws CountryNotFoundException {

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

if (!result.isPresent()) {

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

}

Country country = result.get();

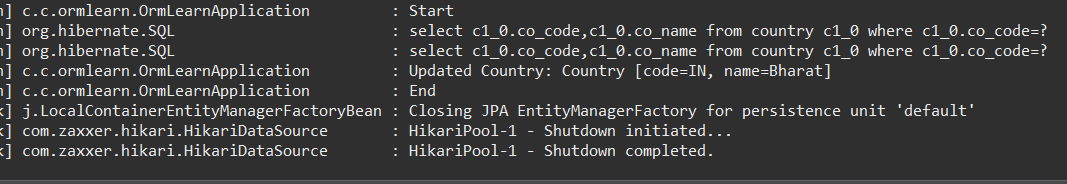
country.setName(newName);

countryRepository.save(country);

}

}

**Output:**

****

**Hands on 9**

**Delete a country based on code**

**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 CountryService *countryService*;

private static final Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.class);

public static void main(String[] args) {

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

***LOGGER***.info("Inside main");

*countryService* = context.getBean(CountryService.class);

// test methods

*testDeleteCountry*();

}

private static void testDeleteCountry() {

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

String codeToDelete = "ZZ"; // use the same code added during addCountry

*countryService*.deleteCountry(codeToDelete);

try {

Country country = *countryService*.findCountryByCode(codeToDelete);

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

} catch (CountryNotFoundException e) {

***LOGGER***.info("Country deleted successfully: " + codeToDelete);

}

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

}

}

**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 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 {

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

if (!result.isPresent()) {

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

}

Country country = result.get();

country.setName(name);

countryRepository.save(country);

}

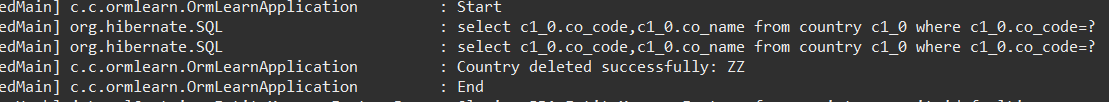
*@Transactional*

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

}

**Output:**