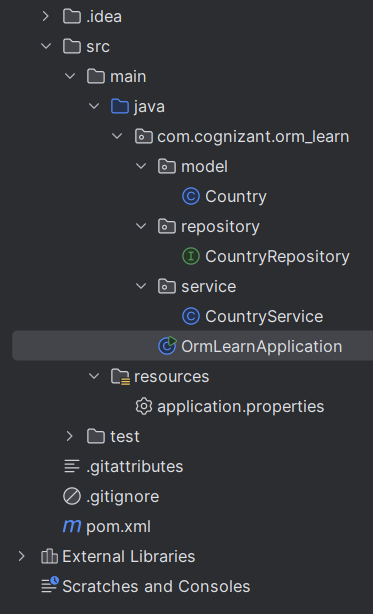
**SPRING DATA JPA**

**Hands on - 1**

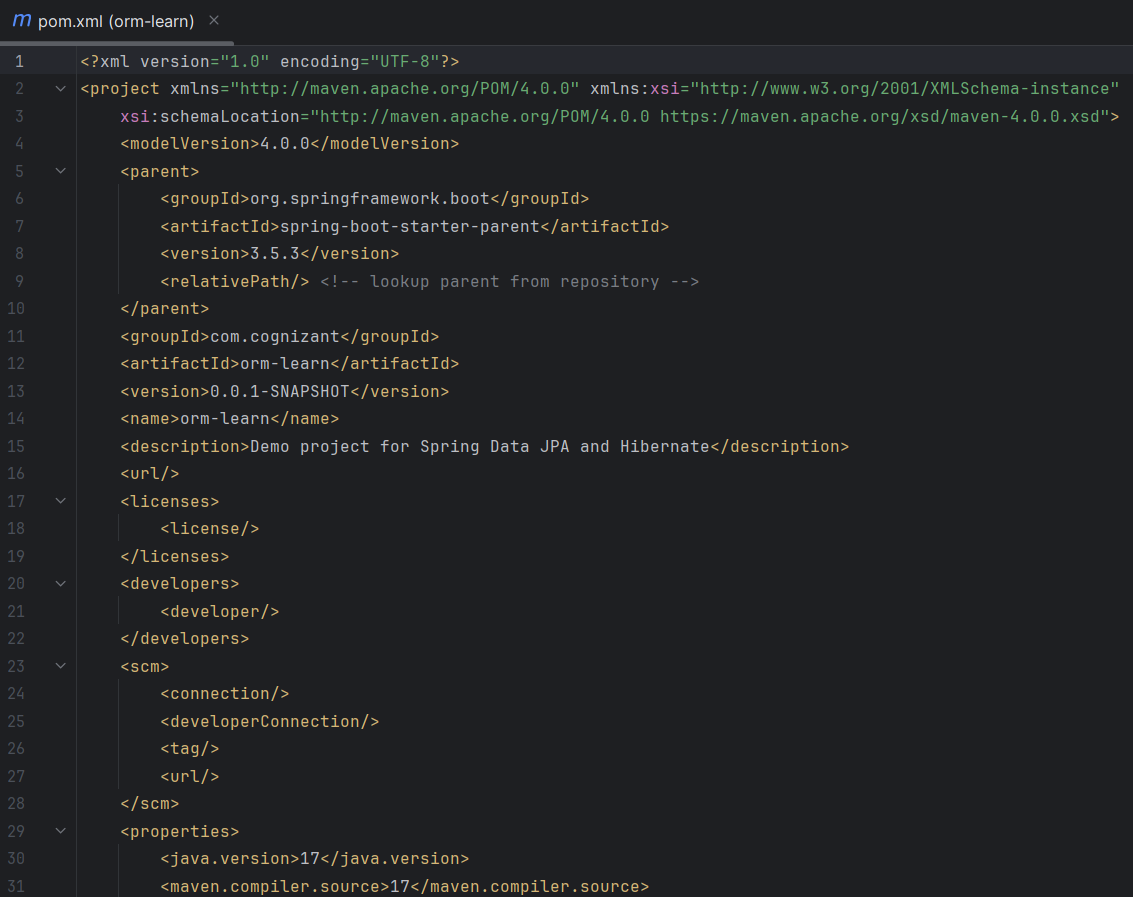
***Spring Data JPA - Quick Example***

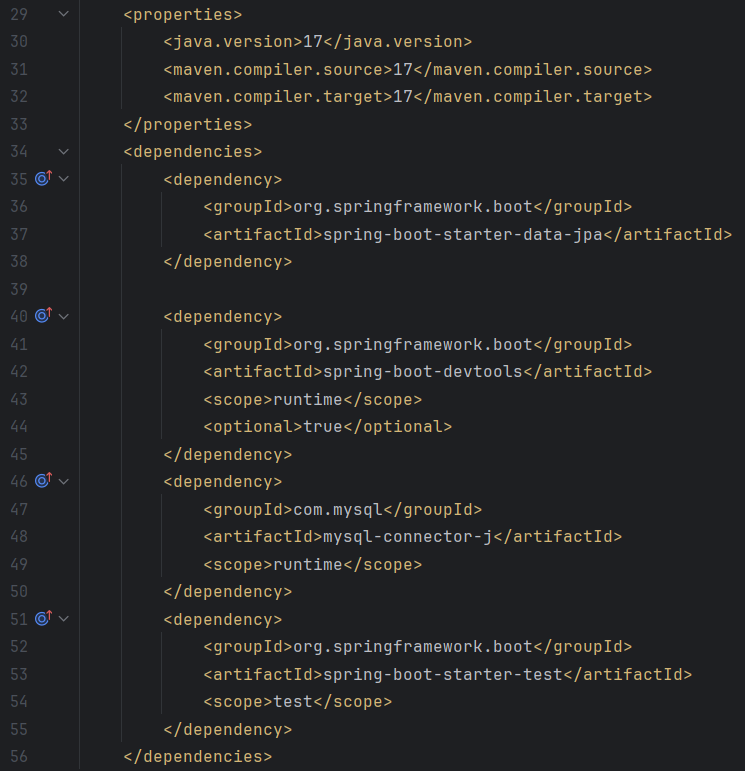
**Project Structure**

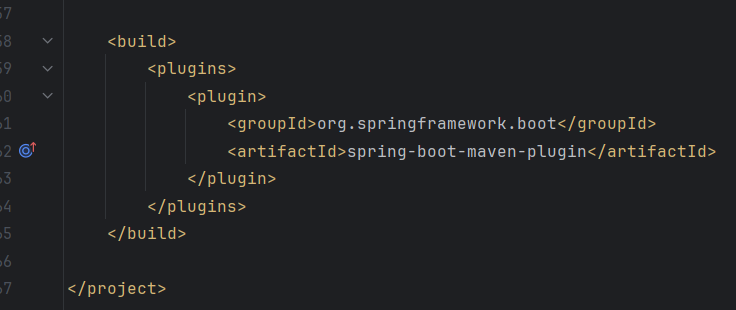
****

**CodeSnippets**

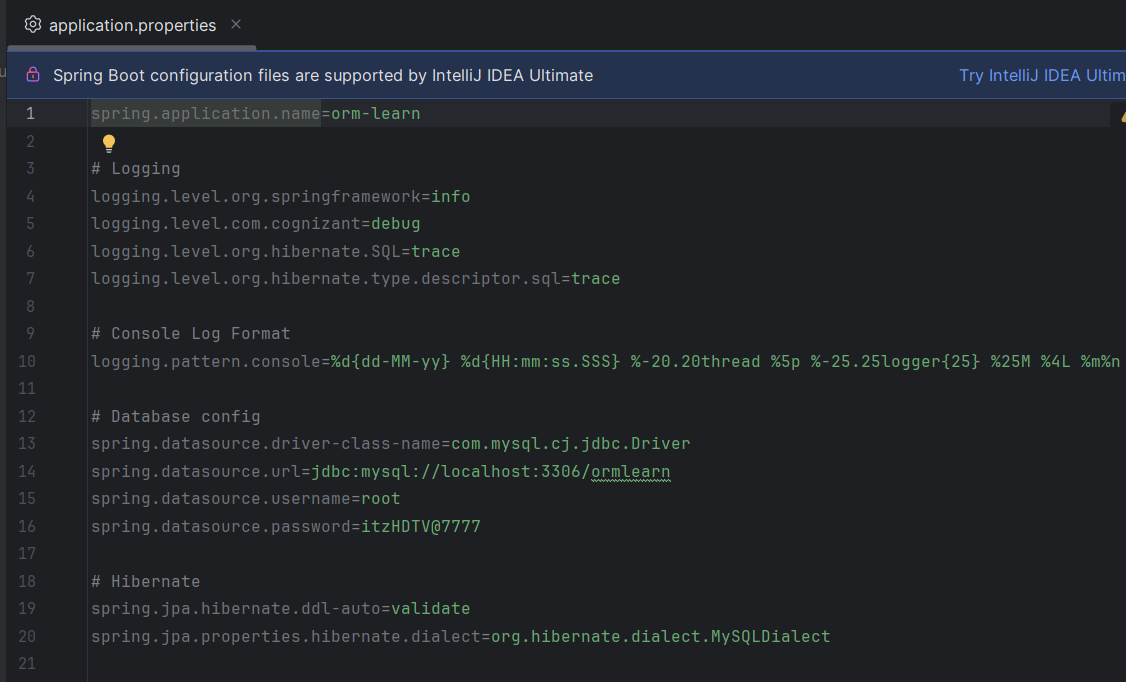
1. ***pom.xml***



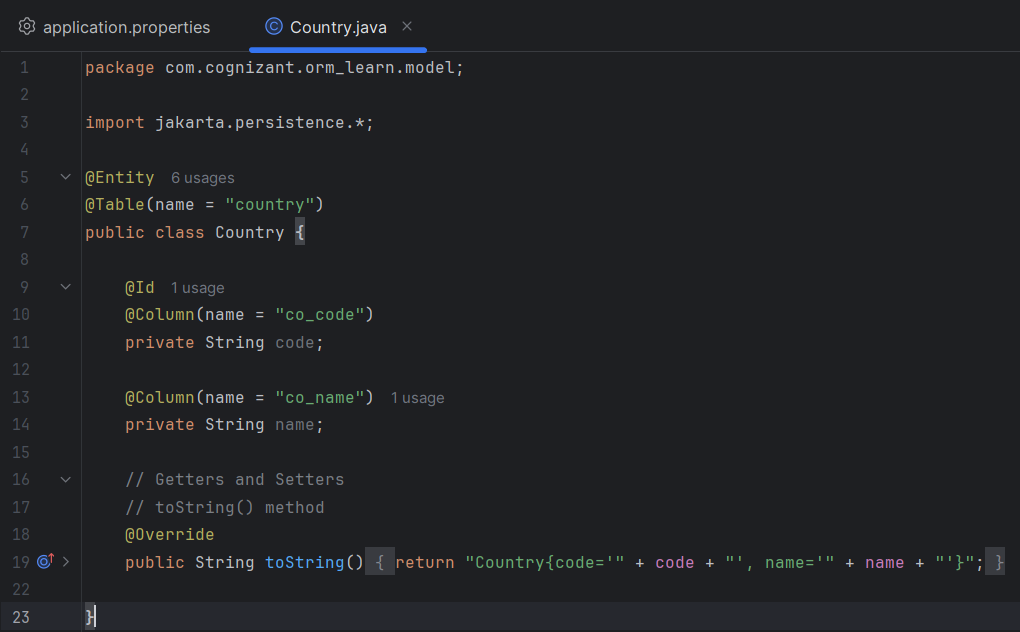




1. ***application.properties***



1. ***Country.java***



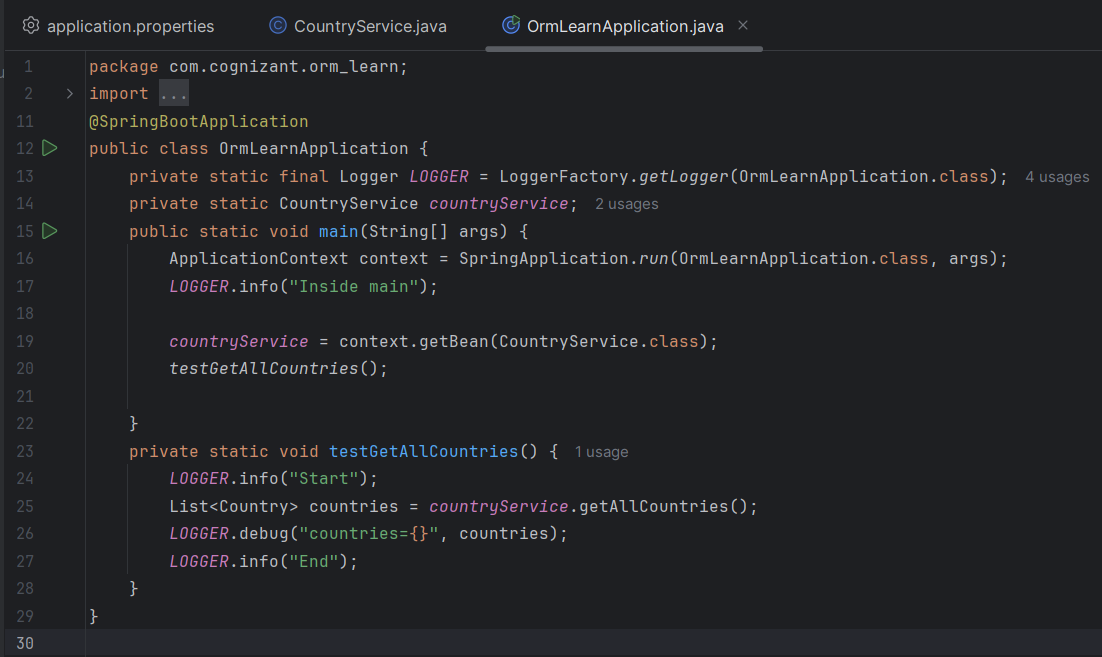
1. ***CountryRepository.java***



1. ***CountryService.java***



1. ***OrmLearnApplication.java***



**HandsOn- 4**

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

Following are the key differences between JPA, Hibernate and Spring Data JPA on the basis of following Parameters

## **1. TYPE:**

**🔹 *JPA* – *Specification***

* **JPA (Java Persistence API)** is **not a tool** or library.
* It’s a **standard (interface/specification)** defined by Oracle (JSR 338) that outlines how Java objects should be stored in a relational database.
* It doesn’t provide implementation — just **rules** (like interfaces in Java).

**🔹 *Hibernate* – *Implementation***

* Hibernate is a **real Java library/tool** that implements the rules of JPA.
* It handles SQL generation, database connection, caching, lazy loading, etc.
* You use **SessionFactory, Session, Transaction** objects.

**🔹 *Spring Data JPA* – *Abstraction Framework***

* Not a JPA implementation.
* It is a **Spring Framework module** built **on top of JPA providers like Hibernate**.
* Gives you **ready-made repository interfaces** to remove 90% of boilerplate.

## **PURPOSE:**

**🔹 *JPA***

* Provides a **common way to define database mappings**, queries, transactions.
* Makes it possible to switch between different ORMs (like Hibernate, EclipseLink).

**🔹 *Hibernate***

* Implements JPA + adds **extra features** (like caching, batch processing).
* Provides a complete ORM tool for advanced use.

**🔹 *Spring Data JPA***

* Designed to **simplify your code**.
* You can just define an interface like JpaRepository<Employee, Integer> — no need to write full DAO layers.
  1. **IMPLEMENTATION**

**🔹 *JPA*: No**

It’s just a **specification**, like interface Shape.

**🔹 *Hibernate*: Yes**

It provides the **actual code** that fulfills JPA rules.

**🔹 *Spring Data JPA*: No**

Instead, it works on top of Hibernate (or any JPA provider), making your life easier.

* 1. **BOILERPLATE**

**🔹 *JPA***

You have to write:

* EntityManager
* EntityTransaction
* Persist calls

**Example:**

*EntityManager em = emf.createEntityManager();*

*em.getTransaction().begin();*

*em.persist(employee);*

*em.getTransaction().commit();*

**🔹 *Hibernate***

You must handle:

* SessionFactory
* Sessions
* Manual transactions

**Example:**

*Session session = factory.openSession();*

*Transaction tx = session.beginTransaction();*

*session.save(employee);*

*tx.commit();*

**🔹 *Spring Data JPA***

You just:

* Extend JpaRepository
* Call .save(), .findAll() etc.

No need to manage sessions or transactions manually.

* 1. **PROVIDED BY**

**🔹 *JPA***

Defined by **Oracle** as part of **Java EE / Jakarta EE**.

**🔹 *Hibernate***

Created and maintained by **Red Hat / Hibernate.org**.

**🔹 *Spring Data JPA***

Created and maintained by **Spring Team / VMware**.

* 1. **TRANSACTIONS**

**🔹 *JPA***

You manage transactions with EntityTransaction manually, unless you're using it inside Spring.

**🔹 *Hibernate***

You use Transaction object — must begin, commit, rollback yourself.

**🔹 *Spring Data JPA***

Just annotate with @Transactional and **Spring handles everything** for you.

* 1. **EASE OF USE**

**🔹 *JPA***

Beginner-friendly but **requires understanding of how it works** and managing some parts yourself.

**🔹 *Hibernate***

Powerful but **verbose**. Good for **low-level control**, but takes effort to set up.

**🔹 *Spring Data JPA***

Most **developer-friendly**.

* You only write interfaces and annotations.
* Spring handles the heavy lifting (saving, deleting, finding records, etc.).