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

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

**What It Is:**

* A **Java specification** (just an interface) for accessing, persisting, and managing data between Java objects and relational databases.
* Provided by jakarta.persistence (formerly javax.persistence).
* Requires an implementation like Hibernate.

**Responsibilities:**

* Define annotations like @Entity, @Id, @OneToMany, etc.
* Does not do the actual ORM; just specifies **what** needs to be done.

**Example Code:**

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Country {

@Id

private String code;

private String name;

// getters and setters

}

This only defines the structure. You **need an implementation** (like Hibernate) to run it.

**2. Hibernate**

**What It Is:**

* A **concrete implementation of JPA**.
* Also includes **non-JPA features** like caching, interceptors, and custom SQL.
* Interacts with the database and translates Java objects into SQL queries.

**Responsibilities:**

* Implements the behavior defined by JPA.
* Adds advanced ORM capabilities.

**Example Usage:**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

public class HibernateExample {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml") // hibernate config file

.addAnnotatedClass(Country.class)

.buildSessionFactory();

Session session = factory.getCurrentSession();

session.beginTransaction();

Country country = session.get(Country.class, "IN");

System.out.println(country.getName());

session.getTransaction().commit();

}

}

This is **manual Hibernate**. It handles everything from sessions to transactions.

**3. Spring Data JPA**

**What It Is:**

* A **Spring-based abstraction** over JPA.
* Automatically implements **repository interfaces** and **CRUD** logic.
* Works with **JPA + Hibernate** under the hood but reduces boilerplate code.

**Responsibilities:**

* Automatically provides repository implementations.
* Makes data access **declarative and easy**.

**Example Usage:**

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

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String namePart); // Custom query method

}

@Service

public class CountryService {

@Autowired

private CountryRepository repository;

public List<Country> getAll() {

return repository.findAll();

}

public Country getByCode(String code) {

return repository.findById(code).orElse(null);

}

}

No need to write query/transaction code manually. Spring handles it all.

**Summary Table:**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (API only) | Implementation (JPA + extras) | Spring abstraction over JPA |
| Provided by | Jakarta EE | Hibernate.org | Spring Framework |
| Code Needed | Requires lots of boilerplate | Medium (manual sessions etc.) | Minimal (declarative repositories) |
| Example Usage | @Entity, @Id | SessionFactory, session.get() | JpaRepository, findBy...() |
| Best For | Portability | Custom control | Productivity & simplicity |

**When to Use What?**

* **Use JPA**: If you want **vendor independence** and just need basic ORM.
* **Use Hibernate directly**: If you need **advanced features** like caching, custom fetch profiles, interceptors.
* **Use Spring Data JPA**: In **Spring Boot apps**, for **fast development**, unless you need very fine-grained control over queries.