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

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

What it is: A specification (i.e., a set of interfaces and rules) provided by Java for ORM (Object Relational Mapping).

Purpose: Allows Java developers to map Java objects to relational database tables.

Contains: Interfaces like EntityManager, Query, EntityTransaction, etc.It doesn't do anything on its own – it needs an implementation like Hibernate.

Example:

@Entity

public class Student {

@Id

private Long id;

private String name;

}

**2. Hibernate**

What it is: A popular implementation of JPA (also has its own extended features beyond JPA).

Role: Provides the actual functionality behind JPA interfaces like EntityManager.

Extra Features: Lazy loading, caching, HQL (Hibernate Query Language), etc.

You can use Hibernate with or without JPA.

Example (Hibernate-specific):

Session session = sessionFactory.openSession();

session.save(student);

**3. Spring Data JPA**

What it is: A Spring Framework module built on top of JPA.

Goal: Make it easier to use JPA by reducing boilerplate code.

Auto-implements common database operations (like save, findAll, delete) via JpaRepository interface.

Uses JPA (usually with Hibernate) under the hood.

Example:

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation (JPA + extras) | Spring module (abstraction over JPA) |
| Standalone usage | No | Yes | No (needs Spring) |
| Boilerplate code | Moderate | Moderate | Very low |
| Vendor-specific features | No | Yes (like HQL, caching) | No (delegates to JPA provider) |
| Abstraction Level | High-level API | Mid-level ORM framework | High-level + Spring integration |