WEEK 3

Mandatory Hands-On

**EX:2 Difference between JPA, Hibernate and Spring Data JPA**

| **Feature** | **JPA (Java Persistence API)** | | **Hibernate** | | **Spring Data JPA** |
| --- | --- | --- | --- | --- | --- |
| **What is it?** | A Specification (API) | | An Implementation (ORM Framework). | | An Abstraction Layer. |
| **Primary Role** | Defines a standard for Object-Relational Mapping (ORM) in Java. | | Provides a full-featured ORM solution implementing JPA and adding its own features. | | Simplifies the data access layer, reducing boilerplate code for repositories. |
| **Relationship** | The standard blueprint. | | A concrete implementation of the JPA blueprint; a "JPA Provider". | | Sits on top of a JPA Provider (like Hibernate); not a JPA implementation itself. |
| **Core Component** | EntityManager, @Entity, @Id, Query, JPQL | | Implements EntityManager; also has native API like Session, SessionFactory, HQL. | | JpaRepository<T, ID>, CrudRepository<T, ID>; provides repository interfaces out of the box. |
| **Boilerplate Code** | Moderate – you manually implement DAO/repository classes and manage EntityManager. | | Similar – manual DAO implementations needed unless using other abstraction layers. | | Very low – only define interfaces; Spring auto-generates implementations. |
| **Provider Independence** | | High – designed to switch between providers with minimal changes. | | Low if using native APIs; high if strictly using JPA interfaces. | High – abstracts over JPA and supports provider swapping via configuration. |
| **Example Usage** | | entityManager.persist(user); entityManager.find(User.class, id); | | session.save(user); (native API) or entityManager.persist(user); (JPA API) | userRepository.save(user); userRepository.findById(id); |