**Hands on 4**

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

**Solution:**

**Java Persistence API (JPA)**

JPA (Java Persistence API) is a **specification (JSR 338)** that defines a standard for persisting, reading, and managing data from Java objects in a relational database. It provides **standard annotations** (such as @Entity, @Id) and APIs (such as EntityManager) to handle persistence but **does not provide a concrete implementation**. An implementation provider like Hibernate or EclipseLink is required to perform the actual ORM operations defined by JPA.

**Hibernate**

Hibernate is an **Object-Relational Mapping (ORM) framework** that **implements the JPA specification**. It is responsible for **mapping Java objects to database tables**, managing SQL generation, connection pooling, caching, and entity state management. Hibernate offers both **JPA APIs (EntityManager)** and its own **native APIs (Session, Transaction)**, allowing flexible and fine-grained control over database operations. Unlike JPA, Hibernate is a complete implementation that can be directly used for persistence operations in Java applications.

**Spring Data JPA**

Spring Data JPA is a **Spring module that provides an abstraction layer over JPA**, making data access simpler and reducing boilerplate code. It requires a **JPA implementation provider like Hibernate** to work under the hood. By using repository interfaces such as JpaRepository and CrudRepository, Spring Data JPA enables **automatic CRUD operations**, supports **automatic transaction management using @Transactional**, and provides query generation through method naming conventions and @Query annotations, allowing rapid and cleaner data access in Spring-based applications.

CODE COMPARISONS:

**Using Hibernate**

We handle **session management, transactions, and exception handling manually** while persisting an employee entity. In this approach, we open a session, begin a transaction, save the employee object, commit the transaction upon success, and handle rollbacks explicitly in case of exceptions.

**Using Spring Data JPA**

Spring Data JPA simplifies the persistence operation by **eliminating manual session and transaction management**.

In this approach, we define a repository interface extending JpaRepository, which provides built-in methods like save(). We inject this repository into the service class, and within a @Transactional method, we persist the employee entity using employeeRepository.save(employee), without managing transactions or sessions explicitly.

Through this comparison, we understand that JPA provides the **guidelines and standards** for persistence, Hibernate **implements these standards**, and Spring Data JPA **simplifies the use of these standards**, reducing the effort required to perform CRUD operations while maintaining clean and maintainable code in Spring-based applications.