**EXERCISE:**

**Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**

1. **Java Persistence API:**

JPA is a standard specification in Java for object-relational mapping (ORM), which allows developers to map Java objects/entities to relational database tables. Common Annotations used is this are @Entity, @Id, @GeneratedValue, @Column, @OneToMany and @ManyToOne. It’s characteristics are:

* JPA is part of Jakarta EE (formerly Java EE).
* It only defines interfaces and annotations; it does not provide implementation.
* Requires a JPA provider like Hibernate, EclipseLink, or OpenJPA to function.
* Allows the use of EntityManager to perform operations like persist, merge, remove, and query.
* **USE:** Developers who want full control and portability across JPA providers without being tied to Spring.

**For example:**

EntityManager em = entityManagerFactory.createEntityManager();

em.getTransaction().begin();

em.persist(new User("Astha"));

em.getTransaction().commit();

1. **Hibernate:**

Hibernate is a powerful ORM framework and the most widely used JPA implementation. It provides all of JPA's capabilities plus many additional features. It Offers features like First- and second-level caching, Lazy/eager loading, Dirty checking, Batch processing, Custom type mappings and so on. Its Key Characteristics are:

* Hibernate is both a standalone ORM and a JPA provider.
* Can be configured using XML or annotations.
* Provides its own APIs like Session, SessionFactory, Criteria, and HQL

**Uses:**

* Useful for projects needing advanced ORM features beyond JPA’s scope.
* Can be used with or without Spring.

**Example for native Hibernate:**

Session session = sessionFactory.openSession();

session.beginTransaction();

session.save(new User("Astha"));

session.getTransaction().commit();

1. **Spring Data JPA:**

Spring Data JPA is a Spring-based abstraction built on top of JPA. It simplifies database access significantly by eliminating boilerplate code and reducing the need to write DAOs (Data Access Objects) manually. Its Key Characteristics are:

* It uses interfaces like JpaRepository, CrudRepository, PagingAndSortingRepository.
* Automatically implements standard CRUD operations.
* Supports query methods by name and custom queries via @Query.
* Easily integrates with Spring Boot, making configuration minimal and application startup fast.

**Use Case:**

* Ideal for rapid development with minimal boilerplate.
* Preferred choice in modern Spring Boot applications.

**Example:**

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}