**Hands on 4**

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

**JPA:**

* JPA stands for Java Persistence API.
* JPA is a specification defined by JSR 338 for object-relational mapping in Java.
* It provides guidelines and APIs to map Java objects to relational database tables.
* JPA itself is not an implementation; it requires a provider like Hibernate or EclipseLink.
* Developers write entity classes using annotations like @Entity, @Id, @Column, etc.
* JPA ensures a standard way of managing persistence logic in any Java application.

**Example:**

import jakarta.persistence.EntityManager;

import jakarta.persistence.EntityManagerFactory;

import jakarta.persistence.Persistence;

public class EmployeeService {

public void addEmployee(Employee employee) {

EntityManagerFactory emf = Persistence.createEntityManagerFactory("employeePU");

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

em.persist(employee);

em.getTransaction().commit();

em.close();

emf.close();

}

}

**Hibernate**

* Hibernate is a popular ORM framework and a concrete implementation of the JPA specification.
* It handles the entire database interaction layer, including caching, lazy loading, and dirty checking.
* It allows direct use of Hibernate APIs or JPA APIs.
* Hibernate requires more boilerplate code compared to Spring Data JPA.
* Manual handling of Session, Transaction, and Query objects is typically required.

**Example:**

/\* Method to CREATE an employee in the database \*/

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Spring Data JPA**

* Spring Data JPA is not a JPA implementation, but a high-level abstraction built on top of JPA and Hibernate.
* It significantly reduces boilerplate code by providing interfaces like JpaRepository, CrudRepository.
* No need to write SQL queries for simple operations (insert, update, delete, find).
* Transaction management is simplified with the use of @Transactional.
* It automatically implements repository interfaces based on method names.

**Example:**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowire

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}