***Hands on 4 : Difference between JPA, Hibernate and Spring Data JPA***

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

* JPA is a **Java specification** (JSR 338) that defines how to manage relational data in Java applications.
* It provides **interfaces and annotations** for ORM (Object Relational Mapping), but does not include any actual implementation.
* To use JPA in a project, a **JPA provider** is needed. Hibernate is one of the most commonly used implementations of JPA.
* JPA focuses on how to map Java objects to database tables and how to manage them using entity managers.

**Example using JPA annotations:**

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue

private int id;

@Column(name = "name")

private String name;

// Getters and setters

}

**2. Hibernate**

* Hibernate is a **popular ORM framework** that implements JPA.
* You can use Hibernate either **with or without JPA**.
* Hibernate provides extra features beyond JPA, such as caching, lazy loading, and custom query language (HQL).
* When using Hibernate directly, the developer is responsible for manually opening sessions, beginning transactions, saving or querying data, and closing the session.

**Example using Hibernate (no Spring):**

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;

}

* Here, all the database interaction is managed manually through Hibernate’s API.

**3. Spring Data JPA**

* Spring Data JPA is **part of the Spring ecosystem** and acts as a wrapper around JPA.
* It **does not implement JPA itself**. Instead, it provides **abstraction and automation** to reduce the amount of boilerplate code required.
* Spring Data JPA automatically creates repository implementations at runtime and handles basic CRUD operations out of the box.
* It also manages transactions and session handling behind the scenes, making the code cleaner and easier to maintain.

**Example using Spring Data JPA:**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

* Here, the developer does not need to write code for opening sessions or managing transactions explicitly. Spring handles all of that automatically.