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

**Java Persistence API (JPA)**

* JPA is a **JSR 338 specification** for persisting, reading, and managing data from Java objects.
* It defines interfaces and rules but **does not provide implementation**.
* **Hibernate** is one of the most commonly used implementations of JPA.

**Hibernate**

* Hibernate is an **Object-Relational Mapping (ORM) tool** that provides a concrete implementation of JPA.
* It handles the mapping of Java classes to database tables.
* Developers have to **manually handle sessions, transactions, and boilerplate code** unless frameworks are added.

**Spring Data JPA**

* Spring Data JPA is a **Spring-based abstraction layer over JPA**.
* It **does not implement JPA** itself but relies on a provider like **Hibernate**.
* It **reduces boilerplate code** by auto-generating common methods like save(), findById(), findAll() etc.
* Provides integration with **Spring’s @Transactional** support and **automated repository management**.

**Code Comparison: Hibernate vs Spring Data JPA**

**🔹 Hibernate**

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;

}

In Hibernate, you manually open/close sessions, manage transactions, and handle exceptions.

**🔹 Spring Data JPA**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}