**Hands-on 4 – JPA vs Hibernate vs Spring Data JPA**

**1. Difference Between JPA, Hibernate, and Spring Data JPA**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (JSR 338) | ORM Framework (JPA Implementation) | Abstraction Layer over JPA |
| Boilerplate Code | High | Moderate | Very Low |
| Transaction Management | Manual | Manual | Automatic |
| Ease of Use | Low | Medium | High |
| Requires Configuration | Yes | Yes | Minimal (Spring Boot Handles Most) |

**2. Hibernate Code Example (Executed by Me)**

**EmployeeDAO.java**

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;

}

**Output:**

Hibernate: insert into employee (name, department) values (?, ?)

[INFO] Employee inserted successfully with ID: 1

**3. Spring Data JPA Code Example (Executed by Me)**

**Employee.java**

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

// Getters and Setters

}

**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);

} }

**OrmLearnApplication.java**

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

EmployeeService service = context.getBean(EmployeeService.class);

Employee emp = new Employee();

emp.setName("Parvathareddy");

emp.setDepartment("Engineering");

service.addEmployee(emp);

LOGGER.info("Employee inserted using Spring Data JPA");

}

}

**Output :**

Hibernate: insert into employee (department, name) values (?, ?)

2025-07-06 21:15:52.456 INFO Employee inserted using Spring Data JPA

**4. Conclusion :**

From this hands-on activity, I personally implemented and executed both Hibernate and Spring Data JPA approaches.

* Hibernate required me to manually manage sessions and transactions.
* Spring Data JPA made the same task far easier by using annotations and auto-generated methods.
* Both methods achieved the same output in the database, but Spring Data JPA saved time and reduced error chances.