**Week: 3**

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

Submitted By:

Disha Jain

Superset Id: 6388877

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

**Overview:**

* JPA is a **Java specification** (JSR 338) for accessing, persisting, and managing data between Java objects and a relational database.
* It provides **standard annotations and interfaces** to define entity relationships and perform database operations.
* JPA **does not contain any implementation** — it needs a provider like **Hibernate** or **EclipseLink**.

**Key Features:**

* **Entity Mapping** using annotations like @Entity, @Id, @OneToMany, etc.
* **JPQL (Java Persistence Query Language)** for queries.
* **Lifecycle callbacks** (e.g., @PrePersist, @PostLoad).
* Defines **EntityManager** for persistence operations.

**Example:**

@Entity

public class Employee {

@Id

private int id;

private String name;

}

You'd still need an implementation like Hibernate to make this work.

**2. Hibernate**

**Overview:**

* Hibernate is a **popular ORM (Object Relational Mapping) framework** that implements the JPA specification.
* It also provides **extra features** that go beyond JPA such as:
  + **Caching mechanisms**
  + **Custom query language** (HQL)
  + **Dirty checking**
  + **Interceptor APIs**
  + Native SQL support

**Key Features:**

* Implements **both JPA and its own APIs**.
* Handles **automatic table creation**, **lazy loading**, and **batch fetching**.
* Uses **SessionFactory** and **Session** instead of JPA's EntityManager.
* Offers **first-level and second-level caching** for performance.

**Code Example:**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Employee employee = new Employee(1, "John");

session.save(employee);

tx.commit();

session.close();

Here, you manually handle session, transaction, exception handling, etc.

**3. Spring Data JPA**

**Overview:**

* Spring Data JPA is part of the **Spring Data project**, which provides a **high-level abstraction** over JPA (and Hibernate).
* It helps you **reduce boilerplate code** by generating repository implementations automatically.
* Works perfectly with **Spring Boot**, allowing you to use annotations like @Repository, @Transactional, @Autowired, etc.

**Key Features:**

* Uses **JpaRepository**, **CrudRepository** interfaces.
* No need to write implementations — Spring Data **auto-generates query logic**.
* Supports **derived query methods** (findByNameAndDepartment).
* Integrates with **Spring’s dependency injection** and **transaction management**.
* Supports **pagination**, **sorting**, and **custom queries** using @Query.

**Code Example:**

**Repository:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

List<Employee> findByName(String name);

}

**Service Layer:**

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

No session management, no transaction boilerplate — Spring handles everything!

**Key Differences Table**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (Interface only) | Framework (Implementation of JPA + extra features) | Abstraction layer over JPA & Hibernate |
| Purpose | Define ORM standard | Provide ORM functionality | Simplify persistence using repositories |
| Implementation required | Yes (like Hibernate) | No (self-contained) | Yes (internally uses Hibernate or other JPA provider) |
| Boilerplate code | Medium | High | Very Low |
| Session/EntityManager handling | Manual | Manual | Automatic (by Spring container) |
| Query language | JPQL | HQL + JPQL | Derived methods, JPQL, native SQL with @Query |
| Repository pattern | Not available | Not built-in | Built-in via JpaRepository, CrudRepository |
| Integration with Spring | Needs configuration | Needs configuration | Seamless integration |
| Transaction management | Manual/Programmatic | Manual/Programmatic | Declarative (@Transactional) |
| Ideal for | Defining portable ORM code | Full control over ORM operations | Rapid development and cleaner code |