**DIFFERENCE BETWEEN JPA , HIBERNATE AND SPRING DATA JPA**

**JPA(Java Persistence API):**

It defines a standard approach for mapping Java objects (entities) to relational database tables and managing their persistence.

JPA is not an implementation – it only defines interfaces, annotations, and rules for object-relational mapping (ORM).  
It requires a implementation such as Hibernate, EclipseLink, or OpenJPA to perform actual database operations.

**Main Goals of JPA:**

* To standardize ORM across Java applications.
* To decouple business logic from persistence logic.
* To reduce boilerplate JDBC code.
* To make applications database-independent and easier to maintain.

Example: Saving an Entity with JPA

**Student.java**

import jakarta.persistence.\*;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

// Constructors, Getters, and Setters

}

**Main.java**

import jakarta.persistence.\*;

public class Main {

public static void main(String[] args) {

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

EntityManager em = emf.createEntityManager();

em.getTransaction().begin();

Student student = new Student();

student.setName("Raj");

em.persist(student);

em.getTransaction().commit();

em.close();

emf.close();

}

}

**persistence.xml**

<persistence xmlns="http://jakarta.ee/xml/ns/persistence" version="3.0">

<persistence-unit name="myPU">

<class>Student</class>

<properties>

<property name="jakarta.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/testdb"/>

<property name="jakarta.persistence.jdbc.user" value="root"/>

<property name="jakarta.persistence.jdbc.password" value="password"/>

<property name="jakarta.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>

<property name="jakarta.persistence.schema-generation.database.action" value="create"/>

</properties>

</persistence-unit>

</persistence>

HIBERNATE:

Hibernate is a popular open-source ORM (Object-Relational Mapping) tool for Java. It provides a complete implementation of the JPA specification and also offers additional powerful features beyond JPA, such as:

* Lazy loading
* First- and second-level caching
* HQL (Hibernate Query Language)
* Criteria queries
* Batch processing

Hibernate can be used in two ways:

1. As a JPA provider (EntityManager)
2. As a standalone ORM tool (using SessionFactory and Session)

**Main Goals of Hibernate:**

* To automate object-to-table mapping.
* To manage database operations through objects instead of SQL.
* To offer a powerful and flexible ORM solution with extra features not defined in JPA.

**Example: Saving a Student Using Hibernate**

**Student.java**

import jakarta.persistence.\*;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

**Main.java**

import org.hibernate.\*;

import org.hibernate.cfg.Configuration;

public class Main {

public static void main(String[] args) {

SessionFactory factory = new Configuration()

.configure("hibernate.cfg.xml")

.addAnnotatedClass(Student.class)

.buildSessionFactory();

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Student student = new Student();

student.setName("Raj");

session.save(student);

tx.commit();

} catch (Exception e) {

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

e.printStackTrace();

} finally {

session.close();

factory.close();

}

}

}

**hibernate.cfg.xml**

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/testdb</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">password</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

<property name="hibernate.hbm2ddl.auto">create</property>

<property name="show\_sql">true</property>

</session-factory>

</hibernate-configuration>

**Pros of Hibernate:**

* Implements all JPA features.
* Powerful additional features (caching, lazy loading, HQL).
* Can work with or without JPA.

SPRING DATA JPA:

Spring Data JPA is not a JPA implementation.Instead, it uses a JPA provider (e.g., Hibernate) internally and offers automatic query generation, transaction management, and simplified repository access.

**Goals of Spring Data JPA:**

* To eliminate boilerplate code for data access.
* To let developers focus on business logic, not database setup.
* To provide automatic implementation of repository interfaces.
* To enable easy integration with Spring Boot and other Spring modules.

**Example: Saving a Student Using Spring Data JPA**

**Student.java**

import jakarta.persistence.\*;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

**StudentRepository.java**

import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

**StudentService.java**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

@Service

public class StudentService {

@Autowired

private StudentRepository studentRepository;

@Transactional

public void saveStudent() {

Student student = new Student();

student.setName("Raj");

studentRepository.save(student);

}

}

**In application.properties (Spring Boot config):**

spring.datasource.url=jdbc:mysql://localhost:3306/testdb

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=create

spring.jpa.show-sql=true

**Code-Level Difference Between Hibernate and Spring Data JPA:**

Both Hibernate and Spring Data JPA are used to perform database operations in Java applications, but they differ significantly in terms of abstraction level, boilerplate code, and developer responsibility. Below is a clear comparison using the same use case: saving a Student entity to the database.

**Hibernate Code Explanation**

Hibernate provides a low-level API where the developer must manually manage:

* Session opening and closing
* Transaction boundaries
* Exception handling
* Save operations

**Code Using Hibernate:**

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

Student student = new Student();

student.setName("Raj");

session.save(student);

tx.commit();

} catch (Exception e) {

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

e.printStackTrace();

} finally {

session.close();

}

**Explanation:**

* The developer is responsible for creating and closing the session.
* Transactions must be explicitly started and committed.
* Save operations use session.save().
* This approach offers fine-grained control but involves more boilerplate code and is less maintainable.

**Spring Data JPA Code Explanation**

Spring Data JPA provides a higher level of abstraction by:

* Automatically managing transactions
* Eliminating the need to write boilerplate DAO code
* Allowing save operations through simple repository interfaces

**Code Using Spring Data JPA:**

public interface StudentRepository extends JpaRepository<Student, Long> {}

@Service

public class StudentService {

@Autowired

private StudentRepository studentRepository;

@Transactional

public void saveStudent() {

Student student = new Student();

student.setName("Raj");

studentRepository.save(student);

}

}

**Explanation:**

* The developer does not need to manage sessions or transactions.
* The save() method is provided by the inherited JpaRepository.
* Transactions are handled automatically using the @Transactional annotation.
* This approach significantly **reduces code**, improves **readability**, and speeds up development.

**Differences Between JPA, Hibernate, and Spring Data JPA**

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (defines rules, no implementation) | Implementation of JPA and standalone ORM framework | Abstraction layer built on top of JPA and Hibernate |
| **Implementation** | Needs a provider like Hibernate, EclipseLink | Fully functional ORM, implements JPA | Uses Hibernate (or other provider) internally, doesn’t implement JPA itself |
| **Boilerplate Code** | Moderate — requires managing EntityManager and transactions | High — requires managing sessions, transactions, and exceptions manually | Very Low — minimal code, handled via Spring's repository and annotations |
| **Transaction Management** | Manual — using EntityManager.getTransaction() | Manual — using Transaction tx = session.beginTransaction() | Automatic — handled using @Transactional annotation |
| **Querying** | JPQL (Java Persistence Query Language) | HQL (Hibernate Query Language), JPQL, Criteria API | Method name-based queries + JPQL + native SQL if needed |
| **Ease of Use / Productivity** | Medium — standard but requires more effort | Low to Medium — powerful but verbose | High — quick development, Spring Boot auto-configuration support |