## ****Difference Between JPA, Hibernate, and Spring Data JPA****

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

JPA is a **standard specification** in Java for managing relational data in object-oriented applications.

It provides a **set of interfaces and annotations**, such as @Entity, @Id, EntityManager, etc., to define how objects should be stored in a database.

Being just a specification, JPA **does not contain actual implementation code**.

It relies on **ORM tools like Hibernate, EclipseLink, or OpenJPA** to execute the logic defined by its API.

### ****2) Hibernate****

Hibernate is an **open-source ORM framework** that implements JPA.

It allows Java developers to **map Java objects to database tables** and provides complete ORM capabilities.

Hibernate includes **JPA features** and also adds extra functionalities like **first-level and second-level caching, batch processing, and custom query options**.

When used standalone, Hibernate requires **explicit management of sessions and transactions**, unless combined with a framework like Spring.

### ****3) Spring Data JPA****

Spring Data JPA is a **part of the Spring Data project**, which aims to simplify database access using JPA.

It **sits on top of JPA and Hibernate**, providing a layer of abstraction that removes the need to write repetitive DAO code.

Developers can use **predefined repository interfaces** (like JpaRepository) to perform CRUD operations without implementing anything manually.

It handles **transaction management, session handling, and exception translation** automatically when integrated with Spring Boot.

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

### ****Using Hibernate (Manual Approach)****

java

CopyEdit

public Integer saveStudent(Student student) {

Session session = sessionFactory.openSession();

Transaction transaction = null;

Integer studentId = null;

try {

transaction = session.beginTransaction();

studentId = (Integer) session.save(student);

transaction.commit();

} catch (Exception ex) {

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

ex.printStackTrace();

} finally {

session.close();

}

return studentId;

}

In this example, the developer must handle the session, transaction, and exception manually.

### ****Using Spring Data JPA (Declarative Approach)****

#### ****StudentRepository.java****

java

CopyEdit

public interface StudentRepository extends JpaRepository<Student, Integer> {

}

#### ****StudentService.java****

java

CopyEdit

@Servicepublic class StudentService {

@Autowired

private StudentRepository studentRepository;

@Transactional

public void saveStudent(Student student) {

studentRepository.save(student);

}

}

Here, Spring handles all the low-level operations (session, transaction, exception) for you using JpaRepository.