# Hands-on 4: Difference Between JPA, Hibernate, and Spring Data JPA

This document explains the differences between **JPA**, **Hibernate**, and **Spring Data JPA**, three widely used technologies in Java for managing relational data in applications

## 🔹 Java Persistence API (JPA)

* JPA is a **Java specification (JSR 338)** for persisting, reading, and managing data between Java objects and relational databases.
* It provides a set of **interfaces and annotations**, but does **not include any implementation**.
* JPA defines the standard for ORM, and frameworks like Hibernate act as implementations of this specification.

## 🔹 Hibernate

* Hibernate is an **Object-Relational Mapping (ORM) tool** that implements the JPA specification.
* It provides additional features beyond the JPA spec like caching, native queries, etc.
* Requires manual handling of sessions and transactions.

## 🔹 Spring Data JPA

* Spring Data JPA is a **Spring module** that provides an **abstraction layer over JPA**.
* It does **not implement JPA**, but depends on providers like Hibernate underneath.
* It significantly reduces **boilerplate code** and integrates transaction management.
* Supports **method-based query generation**, pagination, and more.

## Key Differences

| Feature | JPA | Hibernate | Spring Data JPA |
| --- | --- | --- | --- |
| **Type** | Specification | ORM Framework / JPA Implementation | Abstraction Layer over JPA |
| **Scope** | Standard API for ORM | Full-fledged ORM solution | Simplifies JPA usage in Spring |
| **Dependency** | Requires an implementation | Can work standalone or as JPA provider | Requires JPA and an ORM (e.g., Hibernate) |
| **API** | EntityManager, JPQL | HQL, Session API, JPA APIs | Repository interfaces, Query methods |
| **Use Case** | Portable ORM code | Feature-rich ORM | Simplified data access in Spring |
| **Learning Curve** | Moderate | Steeper due to extra features | Easiest, but requires Spring knowledge |
| **Flexibility** | Limited to spec | Highly customizable | Abstracts complexity, less flexible |

## How They Work Together

* **JPA** provides the standard interface and annotations.
* **Hibernate** implements JPA and adds its own advanced features.
* **Spring Data JPA** builds on JPA (often using Hibernate as the provider) to simplify data access in Spring applications.
* In a typical Spring Boot application:
  + You use Spring Data JPA repositories for data access.
  + Spring Data JPA delegates to Hibernate for ORM functionality.
  + Hibernate implements the JPA specification to interact with the database.

## 🔸 Code Comparison

### 🔸 Hibernate Example

// HibernateExample.java  
public Integer addEmployee(Employee employee) {  
 Session session = factory.openSession(); // Hibernate session  
 Transaction tx = null;  
 Integer employeeID = null;  
  
 try {  
 tx = session.beginTransaction(); // Begin transaction  
 employeeID = (Integer) session.save(employee); // Save entity  
 tx.commit(); // Commit changes  
 } catch (HibernateException e) {  
 if (tx != null) tx.rollback(); // Rollback on error  
 e.printStackTrace();  
 } finally {  
 session.close(); // Close session  
 }  
  
 return employeeID;  
}

### 🔸 Spring Data JPA Example

#### 🔹 EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
  
}

#### 🔹 EmployeeService.java

@Autowired  
private EmployeeRepository employeeRepository;  
  
@Transactional  
public void addEmployee(Employee employee) {  
 employeeRepository.save(employee); // Auto-handled by Spring Data JPA  
}

### Behind the Scenes

* **JPA:** Defines the annotations and contract for persistence (e.g., @Entity, @Id).
* **Hibernate:** Implements the JPA contract and executes DB operations like INSERT, SELECT.
* **Spring Data JPA:**
  + Automatically generates the repository code (e.g., save, findAll).
  + Internally uses Hibernate as the JPA provider (unless configured otherwise).
  + Handles session & transaction lifecycle so you don’t have to.

## Conclusion

* Use **JPA** for portable, standard ORM code.
* Use **Hibernate** when you need advanced ORM features or are not using Spring.
* Use **Spring Data JPA** for simplified data access in Spring applications, leveraging Hibernate as the JPA provider.