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

**🔹 Java Persistence API (JPA)**

* JPA is a **specification** (JSR 338) for object-relational mapping (ORM).
* It defines the **standard interface** for persisting, retrieving, and managing Java objects in a relational database.
* JPA **does not provide an implementation**—instead, vendors like Hibernate implement the JPA standard.
* It defines concepts like Entity, EntityManager, and annotations such as @Entity, @Id, etc.

**🔹 Hibernate**

* Hibernate is a **popular ORM framework** and a **JPA implementation**.
* It allows developers to map Java classes to database tables.
* Developers need to handle **session management** and **transactions** manually (unless integrated with Spring).
* It includes both JPA-compliant features and additional non-standard features (like caching and performance optimizations).

**Example using Hibernate:**

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;

}

**🔹 Spring Data JPA**

* Spring Data JPA is **not a JPA implementation**.
* Spring Data JPA is a part of the Spring Framework that builds on top of JPA.
* It provides a **higher abstraction** on top of JPA, reducing boilerplate code.
* It **auto-generates repository methods**, manages transactions, and integrates tightly with Spring Boot.

**Example using Spring Data JPA:**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}