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

### **Step 1: Understand the Key Concepts**

Start by understanding the difference between:

* **JPA (Java Persistence API)**: This is a specification. It only defines how object-relational mapping should work in Java. It doesn’t include any actual implementation. Think of it like an interface — it needs a tool to implement it.
* **Hibernate**: This is a popular tool that implements the JPA specification. You use Hibernate when you want to manually control database operations using Session, Transaction, etc.
* **Spring Data JPA**: This is part of the Spring framework. It does not implement JPA by itself but builds on top of JPA implementations like Hibernate. It simplifies database access and removes the need to write repetitive boilerplate code, like opening sessions or managing transactions.

### **Step 2: Write a Short Conceptual Explanation**

You’re expected to explain (in a few sentences) how these three fit together:

* JPA is just a guideline or specification.
* Hibernate is one of the tools that follows that guideline.
* Spring Data JPA makes it easier to use Hibernate (or any JPA tool) in a Spring project by letting you use simple interfaces and annotations.

### **Step 3: Implement the Hibernate-Style Code**

Create a method using traditional Hibernate. This means:

* You open a session.
* Begin a transaction.
* Save the object.
* Commit the transaction.
* Handle exceptions and close the session manually.

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;

}

### **Step 4: Implement the Same Logic Using Spring Data JPA**

Now, implement the same addEmployee functionality using Spring Data JPA, which will look much simpler.First, create a repository interface:  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

Then, inject and use it in a service class:

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}  
**Step 5: Test It Using Spring Boot**

Use a CommandLineRunner or REST controller to call the addEmployee method and verify that it's working.  
@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

Employee emp = new Employee();

emp.setName("Jane");

emp.setDepartment("Finance");

employeeService.addEmployee(emp);

}

}

In the Hibernate-based approach, I had to manually open and close the session, start a transaction, handle exceptions, and manage everything explicitly. In contrast, Spring Data JPA allows me to simply call save() on a repository interface, and it handles all the underlying complexity like transactions and sessions automatically.