Step 1: Understanding the Core Concepts

In this hands-on, I have explored the key differences between JPA, Hibernate, and Spring Data JPA. All three are part of the persistence layer in Java-based applications and help in managing data between Java objects and relational databases.

Step 2: What is JPA?

JPA (Java Persistence API) is a Java specification (JSR 338) that defines how Java objects can be mapped to relational database tables. It provides a standard set of APIs for object-relational mapping but does not include any concrete implementation.

Instead, we use a provider like Hibernate that implements the JPA specification.

Step 3: What is Hibernate?

Hibernate is a popular ORM (Object-Relational Mapping) framework that implements the JPA specification. It handles the actual mapping of Java classes to database tables and allows us to perform database operations without writing complex SQL queries.

Hibernate supports advanced features like lazy loading, caching, and automatic schema generation.

Step 4: What is Spring Data JPA?

Spring Data JPA is not a JPA implementation. It is a Spring module that provides an additional layer of abstraction over JPA. It simplifies the development of data access layers by eliminating boilerplate code.

Using Spring Data JPA, we can perform CRUD operations just by extending interfaces like JpaRepository, without writing actual SQL or Hibernate code. It also manages transactions and provides support for query creation by method names.

Step 5: Code Comparison — Hibernate vs Spring Data JPA

To understand the difference better, I have included sample code below that shows how adding an employee record differs when using Hibernate and Spring Data JPA.

~ Hibernate Code Example:

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 Code Example:

EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowired

private EmployeeRepository employeeRepository;

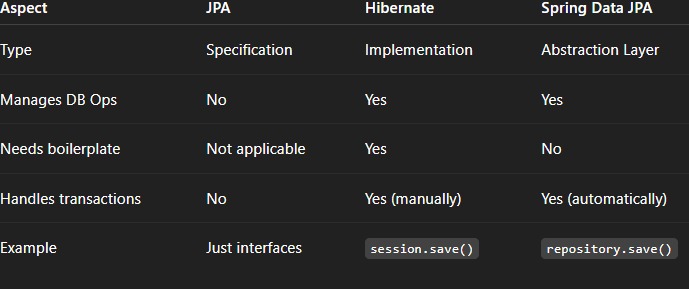
@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

Step 6: Summary Table — Key Differences



Step 7: Optional Output (if required)

Since this task is conceptual, there is no fixed output required. However, if the save() method is called in Spring Data JPA, we might expect the following log or message:

