**Hands on 4**

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

**1. What is Java Persistence API (JPA)?**

The Java Persistence API (JPA) is a specification for managing relational data in Java applications. It provides a standardized way to persist, read, and manage data between Java objects and relational database tables using ORM (Object-Relational Mapping) techniques.

* It is part of the Java EE standard (JSR 338).
* JPA itself does not provide an implementation, it only defines interfaces and annotations like @Entity, @Id, @OneToMany, etc.
* A JPA provider like Hibernate or EclipseLink is required to use JPA in practice.

**2. What is Hibernate Framework?**

Hibernate is an open-source ORM (Object Relational Mapping) framework for Java. It provides a reference implementation of the JPA specification, making it one of the most popular JPA providers.

* Maps Java classes to database tables.
* Handles database communication and transaction management.
* Provides advanced features like caching, lazy loading, dirty checking, etc.

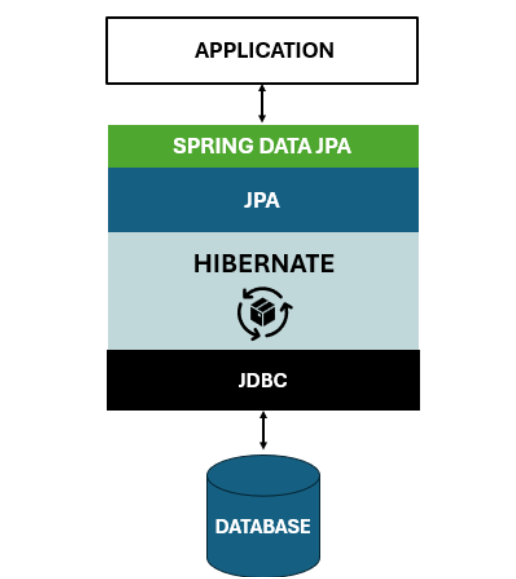
Note: JPA is a specification and Hibernate is an implementation of that specification.

**3. What is Spring Data JPA?**

Spring Data JPA is a part of the larger Spring Data family. It is a powerful framework that builds on top of JPA and Hibernate to make data access easier and faster.

* Not a JPA provider.
* Provides an abstraction layer that removes boilerplate code for DAO layers.
* Uses interfaces like JpaRepository, allowing developers to focus more on business logic.
* Supports derived query methods using naming conventions.
* Handles transactions declaratively with @Transactional.

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| Type | Specification/API | Framework/Implementation | Framework/Abstraction |
| Implementation | No | Yes (JPA Provider) | No (Works with JPA Providers like Hibernate) |
| Responsibility | Defines ORM rules | Implements ORM | Simplifies and abstracts JPA implementation |
| Transaction Handling | Not directly handled | Manually handled | Declaratively handled via @Transactional |
| Boilerplate Code | Moderate | High | Very Low |
| Usage | Annotations & Entity Management | Session, Criteria, Manual Transaction | JpaRepository, Custom Queries by Method Naming |

****

**Code Comparison:**

1. **Hibernate Approach (Manual Implementation)**

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;

}

1. **Spring Data JPA Approach (Clean & Simple)**  
     
   Repository Interface

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

Service Class

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**Conclusion**

* **JPA** is the blueprint/specification for ORM in Java.
* **Hibernate** is a popular JPA implementation and full-fledged ORM framework.
* **Spring Data JPA** is a powerful abstraction built on JPA that makes CRUD operations simpler and cleaner.