**Hands-on 4 – Difference between JPA, Hibernate, and Spring Data JPA**

**Introduction**

This document explains the differences between Java Persistence API (JPA), Hibernate, and Spring Data JPA. These are commonly used technologies in Java-based enterprise applications for database access and Object-Relational Mapping (ORM).

**1. Java Persistence API (JPA)**

* JPA is a specification (JSR 338) that defines a standard for mapping Java objects to relational database tables.
* It provides a set of annotations and interfaces to manage persistence but does not contain any concrete implementation.
* To use JPA, an implementation is required, such as Hibernate or EclipseLink.
* JPA helps decouple the application code from specific ORM tools, making it easier to switch between implementations if needed.

**2. Hibernate**

* Hibernate is an ORM framework that provides an implementation of the JPA specification.
* It allows developers to work with Java objects while abstracting away the complexities of interacting with relational databases.
* Hibernate also offers additional features such as caching, lazy loading, and HQL (Hibernate Query Language).
* It is one of the most widely used JPA providers in the Java ecosystem.

**3. Spring Data JPA**

* Spring Data JPA is not an implementation of JPA. Instead, it is a Spring framework module that simplifies data access using JPA.
* It abstracts the boilerplate code required to perform common operations like saving, deleting, and querying data.
* Spring Data JPA works on top of a JPA provider (typically Hibernate) and integrates seamlessly with Spring’s transaction management.
* It enables developers to define repository interfaces and automatically provides implementations at runtime.

**4. Code Comparison**

**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;

}

**Using Spring Data JPA**

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

java

CopyEdit

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**5. Summary of Differences**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| Type | Specification | Implementation | Abstraction layer over JPA |
| Requires Implementation | Yes | No | Yes (uses JPA provider like Hibernate) |
| Boilerplate Code | More | More | Minimal |
| Transaction Management | Not included | Manual | Automatic (via Spring) |
| Custom Queries | JPQL | HQL | JPQL, method names, or annotations |

**6. Conclusion**

JPA defines the standard for ORM in Java, while Hibernate is a concrete implementation that follows the JPA specification. Spring Data JPA builds on top of JPA and Hibernate to provide a more simplified and developer-friendly approach to data persistence. Each layer adds abstraction and productivity benefits, depending on the complexity and requirements of the application.

**7. References**

1. [What is the difference between Hibernate and Spring Data JPA – DZone](https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1)
2. [Introduction to JPA – JavaWorld](https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html)