**Hands-on 4: Difference Between JPA, Hibernate, and Spring Data JPA**

This report summarizes the differences between JPA (Java Persistence API), Hibernate, and Spring Data JPA and includes a comparative overview of how they are used for data persistence in Java-based applications. The goal is to understand their relationships and differences through explanation and code examples.

# Java Persistence API (JPA)

JPA is a specification provided by Oracle under JSR 338. It defines a standard for object-relational mapping (ORM) between Java objects and relational databases. However, JPA does not provide any concrete implementation on its own. Instead, it relies on implementations like Hibernate to perform its operations.

# Hibernate

Hibernate is a popular ORM tool that implements the JPA specification. It provides the actual logic and mechanisms for performing database operations. While Hibernate reduces a lot of JDBC boilerplate, it still requires manual session and transaction handling if used standalone (without Spring).

Example: Adding an Employee 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 an abstraction over JPA and its implementations such as Hibernate. It is part of the Spring Framework and is designed to reduce boilerplate code in data access layers. It simplifies CRUD operations by auto-generating queries and managing transactions behind the scenes.

Example: Adding an Employee using Spring Data JPA

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
}

@Service  
public class EmployeeService {  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}

# Summary Comparison Table

|  |  |  |  |
| --- | --- | --- | --- |
| ***Feature*** | **JPA** | **Hibernate** | **Spring Data JPA** |
| ***Type*** | Specification | Implementation | Abstraction |
| ***Has implementation?*** | No | Yes | No (uses Hibernate) |
| ***Handles transactions?*** | No | Yes (Manual) | Yes (auto) |
| ***Boilerplate required?*** | No | Yes | No |
| ***Ease of use*** | Medium | Harder | Easiest |

# Conclusion

JPA serves as a guideline for ORM frameworks. Hibernate provides a concrete implementation with more flexibility, but can be verbose. Spring Data JPA offers the most developer-friendly experience by eliminating boilerplate and handling common operations automatically. In this hands-on, we explored the difference in implementation between Hibernate and Spring Data JPA through simple add operation examples.