**HandsOn 4: Difference between JPA, Hibernate and Spring Data JPA**

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

* It is just a specification (interface).
* No implementation - only rules/standards for ORM.
* Hibernate is one of the most common implementations.

**2. Hibernate**

* It is an actual implementation of JPA.
* Provides additional features beyond JPA (e.g., caching, custom query support).
* Requires manual session and transaction management.

**3. Spring Data JPA**

* It is a Spring-based abstraction over JPA.
* Removes boilerplate code using `JpaRepository`.
* Automatically manages sessions and transactions.

**Comparison Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification | Implementation of JPA | Abstraction over JPA (like Hibernate) |
| **Transaction Handling** | Not defined | Manual | Automatic using @Transactional |
| **Boilerplate Code** | No | Yes | Reduced significantly |
| **Used With** | Hibernate, Eclipse | JPA | JPA + Spring Framework |

**Hibernate Approach (Manual ORM Handling)**

// Hibernate Approach

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;

}

**Explanation**

* It is manually managing the database connection (Session), transaction (Transaction), and error handling.
* session.save(employee) is used to persist the object.
* It’s pure Hibernate, and doing everything manually - which means more boilerplate code.
* It must handle:
  + Opening/closing sessions
  + Starting/committing/rolling back transactions
  + Exception handling

**Spring Data JPA Approach (Simplified ORM with Spring)**

// Spring Data JPA Approach

// Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

// Service

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Explanation**

* EmployeeRepository extends JpaRepository, which gives built-in CRUD operations - no need to implement save, delete, findById, etc.
* Spring Boot automatically generates the code behind this interface.
* In addEmployee(), it just call employeeRepository.save(employee) - Spring handles the session, transaction, and commit/rollback automatically.
* The @Transactional annotation ensures that if any exception occurs, the transaction is rolled back automatically.