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

* **Hibernate (Manual Implementation)**

/\* Method to CREATE an employee in the database \*/

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 (Simplified Approach)**

*EmployeeRepository.java –*

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

*EmployeeService.java –*

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Differences Between Hibernate and Spring Data JPA (Based on Code Snippets) –**

| **Feature/Aspect** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- |
| **Session Management** | Manually created using Session session = factory.openSession() | Not required — handled automatically by Spring |
| **Transaction Handling** | Manual with Transaction tx = session.beginTransaction() and tx.commit() | Handled via @Transactional annotation on service method |
| **Error Handling** | Explicit try-catch block with rollback on exception | Not required — Spring handles rollback automatically on exception |
| **Closing Resources** | Must call session.close() manually in finally block | No manual resource management needed |
| **Method to Save Data** | session.save(employee) | employeeRepository.save(employee) |
| **Code Length** | Long, with boilerplate | Short and concise |
| **Layer Separation** | Logic is written in the method itself | Follows Spring layered architecture (Repository + Service) |
| **Focus** | More low-level, gives full control over transaction/session | High-level abstraction, focuses on business logic |
| **Reusability & Testing** | Harder to test due to tight coupling with Hibernate APIs | Easier to test due to clean separation and use of Spring annotations |

**Why Use Spring Data JPA?**

* Reduces complexity and code size
* Eliminates manual transaction/session management
* Makes code more readable and maintainable
* Follows modern development best practices