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

Introduction to JPA, Hibernate, and Spring Data JPA:

**JPA (Java Persistence API):**

* Specification only (JSR 338)
* No implementation provided
* Defines how Java objects persist to relational DB
* Common implementations: Hibernate, Eclipse Link.

**Hibernate:**

* ORM tool and **implementation of JPA**
* Requires manual handling of sessions, transactions

**Spring Data JPA:**

* Abstraction over JPA and Hibernate
* Eliminates boilerplate code (e.g., session handling
* Provides ready-to-use repository interfaces.

Comparison Table:

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation of JPA | Abstraction over JPA & Hibernate |
| Boilerplate Code | Medium | High | Very Low |
| Transaction Handling | Manual | Manual | Automatic with @Transactional |
| Query Methods | JPQL | HQL | Method Name Derived Queries |
| Dependency | javax.persistence | org.hibernate | org.springframework.data.jpa |

Code Comparison:

Hibernate Example:

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 Example:

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

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}