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

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (Interface) | Implementation of JPA + ORM tool | Abstraction Layer on top of JPA & Hibernate |
| **Implementation** | JPA itself is just an API, not an implementation | Provides actual implementation of JPA | Uses JPA under the hood, mostly with Hibernate |
| **Boilerplate Code** | Requires more code for operations | Slightly reduces boilerplate | Greatly reduces boilerplate code |
| **Usage** | Defines rules for entity management and persistence | Entity mappings, queries, transaction handling | Provides ready-to-use repository interfaces like JpaRepository |
| **Provided by** | Java EE / Jakarta EE | Red Hat / Hibernate.org | Spring Framework |
| **Main Focus** | Defines the **standard** for persistence in Java | Provides full ORM functionality | Simplifies data access for developers |

**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**

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}