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

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (Interface/API) | Implementation (ORM tool) | Abstraction over JPA (helper framework by Spring) |
| **Purpose** | Defines how to manage relational data in Java apps | Maps Java objects to DB (ORM), implements JPA | Reduces boilerplate, provides repository abstraction |
| **Implements** | Not applicable (just API) | Yes (implements JPA spec) | Uses JPA providers like Hibernate |
| **Boilerplate Code** | Requires more code manually | Requires code for session handling, transactions | Removes most of the boilerplate (via JpaRepository, etc.) |
| **Provided By** | Oracle (JSR 338) | Red Hat | Spring Framework |
| **Example Use** | EntityManager, @PersistenceContext | SessionFactory, Session, Transaction | JpaRepository, @Transactional, @Repository |
|  | **#REAL WORKING EXAMPLE** |  |  |

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

* A specification only. No implementation.
* Defines how ORM (Object Relational Mapping) should be done.
* Needs a provider (e.g., Hibernate).

**2. Hibernate**

* A concrete implementation of JPA.
* You handle sessions, transactions manually.

**Example using Hibernate (without Spring Data JPA):**

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

Country country = new Country();

country.setCode("IN");

country.setName("India");

session.save(country);

tx.commit();

session.close()

**3. Spring Data JPA**

* A wrapper over JPA with auto-generated methods.
* Removes boilerplate code using repositories.

**Example using Spring Data JPA:**

CountryRepository.java

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContainingIgnoreCase(String name);

}

CountryService.java

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public Country addCountry(Country country) {

return countryRepository.save(country);

}

public Optional<Country> getCountryByCode(String code) {

return countryRepository.findById(code);

}

public List<Country> searchCountriesByName(String name) {

return countryRepository.findByNameContainingIgnoreCase(name);

}

public void deleteCountry(String code) {

countryRepository.deleteById(code);

}

public Country updateCountry(String code, Country newCountry) {

Country existing = countryRepository.findById(code).orElseThrow();

existing.setName(newCountry.getName());

return countryRepository.save(existing);

}

}

**Code Comparison**

**Using Hibernate (Manual)**

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

session.save(employee);

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

} finally {

session.close();

}

You have to manage:

* Sessions
* Transactions
* Exceptions
* Session closing

**Using Spring Data JPA**

**Repository Interface**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**Service Class**

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

Spring handles:

* Session & transaction management
* Query generation
* Object persistence