**Ex 1 : Spring Data JPA – Quick Example**

**Product.java**

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.Id;

@Entity

public class Product {

@Id

@GeneratedValue

private Long id;

private String name;

private String category;

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

public String getCategory() { return category; }

public void setCategory(String category) { this.category = category; }

}

ProductRepository.java

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {

List<Product> findByCategory(String category);

}

**ProductService.java**

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ProductService {

@Autowired

private ProductRepository repo;

public void saveProduct() {

Product product = new Product();

product.setName("Laptop");

product.setCategory("Electronics");

repo.save(product);

}

public List<Product> getElectronics() {

return repo.findByCategory("Electronics");

}

}

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

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification (interface) | Implementation (framework) | Abstraction layer on top of JPA |
| **Created By** | Java EE / Jakarta EE | Red Hat | Spring Team (Pivotal) |
| **Purpose** | Standard API for ORM in Java | ORM framework implementing JPA | Simplifies JPA by removing boilerplate |
| **Implements** | Only the interfaces and annotations | Full JPA + extra features like caching, lazy loading | Uses JPA (usually Hibernate) + Spring features |
| **Examples** | @Entity, @Id, EntityManager | Session, Transaction, Criteria, HQL | CrudRepository, JpaRepository, @Repository |
| **Config File** | persistence.xml | hibernate.cfg.xml | application.properties or application.yml in Spring Boot |
| **Boilerplate Code** | High – manual transaction handling, persistence context | Medium – simplified but still manual | Low – auto-wired repositories, no need for boilerplate |
| **Query Language** | JPQL (Java Persistence Query Language) | JPQL + HQL (Hibernate Query Language) | JPQL + Query derivation from method names |
| **Transactions** | Manual via EntityManager | Manual via Session and Transaction | Handled automatically with @Transactional |
| **Lazy/Eager Fetching** | Supported | Advanced support with better control | Inherited from JPA/Hibernate |
| **Caching** | Basic support | Built-in support (1st and 2nd level cache) | Delegated to Hibernate (depends on provider) |
| **Pagination/Sorting** | Manual | Manual | Built-in with Pageable, Sort interfaces |
| **Repository Layer** | Must be manually implemented | Optional helper classes | Interface-based (JpaRepository, CrudRepository, etc.) |
| **Learning Curve** | Steep – needs deep understanding of JPA | Moderate – still low-level API | Easy – ideal for beginners and rapid development |
| **Best Use Case** | If you want pure Java EE application or need full control | When you want advanced ORM features with flexibility | If you're using Spring Boot or want fast, clean development |
| **Customization** | Full flexibility | Full flexibility + extra features | High-level abstraction (but can customize with custom repos) |
| **Vendor Independence** | Yes – provider-neutral (choose Hibernate, EclipseLink, etc.) | No – Hibernate specific | Uses JPA, usually with Hibernate under the hood |
| **Unit Testing** | Needs mocking EntityManager | SessionFactory setup required | Easy with @DataJpaTest |
| **Performance Tuning** | Needs manual optimization | Better optimization features | Relies on underlying JPA provider (often Hibernate) |
| **Community Support** | Good | Excellent – widely used ORM framework | Excellent – widely used in Spring ecosystem |
| **Popular With** | Java EE projects, enterprise systems | Hibernate-focused apps, legacy systems | Spring Boot and microservice-based applications |
| **Additional Features** | None (just the API) | Caching, Dirty Checking, Native SQL Support | Auto Query Methods, Pagination, Specification, Query by Example |