**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**

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (Interface) | ORM Tool, JPA Implementation | Framework on top of JPA implementations |
| **Standard** | Yes (JSR 338) | No (Implements JPA specification) | No (Uses JPA provider like Hibernate) |
| **Implementation Provided?** | No (Just a set of interfaces) | Yes (Provides implementation) | No (Relies on Hibernate or others) |
| **Boilerplate Code** | Requires more code and configuration | Less boilerplate compared to JPA | Very minimal due to use of repository pattern |
| **Configuration Effort** | Requires XML or Java config with EntityManager setup | Manual setup with SessionFactory | Minimal setup using Spring Boot starter |
| **CRUD Operations** | Must be coded manually | Slightly easier with session.save() etc. | Pre-defined via JpaRepository interface |
| **Query Language** | JPQL | HQL (Hibernate Query Language), JPQL | JPQL, Derived Queries, Native Queries, and @Query annotations |
| **Transaction Handling** | Manual or container-managed | Provided via API but manually controlled | Simplified using @Transactional and Spring-managed transactions |
| **Caching Support** | Not part of JPA standard | Yes (First level and optional second level via EHCache) | Relies on JPA provider's caching (e.g., Hibernate) |
| **Lazy Loading** | Supported via @OneToMany(fetch=FetchType.LAZY) etc. | Fully supported with fine control | Inherited from JPA provider |
| **Auditing** | Must be implemented manually | Manual or plugin-based | Built-in support via Spring Data JPA annotations (@CreatedDate, etc.) |
| **Paging & Sorting** | Not supported directly | Requires custom logic | Built-in methods via PagingAndSortingRepository or Pageable |
| **Custom Queries** | Manually defined using JPQL | Uses HQL/Criteria API | @Query and method-name-based derived queries |
| **Entity Lifecycle Events** | Supports lifecycle annotations (@PrePersist, etc.) | Fully supports them | Inherited via JPA annotations |
| **Best Use Case** | When using a JPA provider independently or across different frameworks | When more control or advanced ORM features are needed | When building Spring Boot applications with rapid development and clean structure |