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

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

**Type**: Specification  
**Maintained by**: Jakarta EE (formerly Java EE)  
**Purpose**: Provides a standard for Object Relational Mapping (ORM)

**Description:**

* JPA defines a set of annotations and interfaces to map Java objects to relational database tables.
* It is only an API, not an implementation.
* Requires a JPA provider (e.g., Hibernate, EclipseLink) for actual functionality.

**Key Features:**

* Annotations: @Entity, @Table, @Id, @OneToMany, etc.
* Entity lifecycle management
* JPQL (Java Persistence Query Language)
* Transaction management

**Example Usage:**

**java**

@Entity

public class Employee {

@Id

private Long id;

private String name;

}

**2. Hibernate**

**Type**: Implementation of JPA  
**Maintained by**: Red Hat  
**Purpose**: ORM framework that implements the JPA specification (and goes beyond it)

**Description:**

* Hibernate is a popular, mature ORM framework.
* It provides all JPA functionalities and also offers extended features not covered by the JPA specification.
* Can be used with or without JPA.

**Additional Features (beyond JPA):**

* Native SQL queries
* Hibernate Criteria API
* First-level and second-level caching
* Advanced fetch strategies
* Batch processing

**Example Usage (Native Hibernate):**

**java**

Session session = sessionFactory.openSession();

Employee emp = session.get(Employee.class, 1L);

**3. Spring Data JPA**

**Type**: Abstraction Layer on top of JPA and Hibernate  
**Maintained by**: Spring Framework (Pivotal/VMware)  
**Purpose**: Simplifies database access using Spring and JPA

**Description:**

* Provides an abstraction layer to reduce boilerplate code for data access.
* Uses interfaces and method naming conventions to auto-generate queries.
* Supports custom queries using @Query annotation.
* Integrates seamlessly with Spring Boot and Spring Core features.

**Key Features:**

* Repository support: CrudRepository, JpaRepository, PagingAndSortingRepository
* Query derivation from method names
* Pagination and sorting support
* Custom JPQL/native SQL with @Query

**Example Usage:**

**java**

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByName(String name);

@Query("SELECT e FROM Employee e WHERE e.department.id = :id")

List<Employee> findByDepartmentId(@Param("id") Long id);

}

**Summary Table**

| **Concept** | **Type** | **Maintained by** | **Role/Purpose** |
| --- | --- | --- | --- |
| JPA | Specification | Jakarta EE | Defines standard API for ORM |
| Hibernate | Implementation | Red Hat | JPA provider + additional ORM features |
| Spring Data JPA | Abstraction Layer | Spring Framework | Simplifies JPA usage in Spring-based applications |

**Comparison Highlights**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Framework | Spring abstraction layer |
| Can be used directly? | No | Yes | Yes (within Spring apps) |
| Requires JPA Provider? | Yes | No (it is one) | Yes (typically Hibernate) |
| Advanced ORM Features | Limited | Extensive | Delegated to Hibernate |
| Integration with Spring | Manual | Manual | Full Spring integration |
| Query generation from method name | No | No | Yes |
| Custom Queries via Annotations | Limited | Yes | Yes |
| Boilerplate Code | High | Medium | Low |