* **Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**

| Feature | JPA | Hibernate | Spring Data JPA |
| --- | --- | --- | --- |
| Type | Specification | Implementation + ORM | Abstraction layer over JPA |
| Defines ORM behavior | ✅ Yes | ✅ Implements JPA + own features | ❌ Uses JPA/Hibernate underneath |
| Provides boilerplate reduction | ❌ No | ❌ No | ✅ Yes |
| Query language | JPQL | HQL + JPQL | Derived queries + JPQL + Native SQL |
| Managed by Spring Boot | ❌ No | ❌ No | ✅ Yes |

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

* What it is: A specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.
* Key Point: JPA itself is just an interface — it defines how ORM should work, but doesn't implement anything.
* Who provides implementation? Tools like Hibernate, EclipseLink, OpenJPA.

**Ex:**

@Entity

public class Book {

@Id

private Long id;

private String title;

}

**2. Hibernate**

* **What it is:** A **concrete implementation** of the JPA specification and a **full ORM tool**.
* **Key Point:** Hibernate provides:
  + Entity management
  + Caching
  + Lazy loading
  + HQL (Hibernate Query Language)
* You can use Hibernate either:
  + With **JPA annotations**
  + Or using **native Hibernate APIs**

**Ex:**

Session session = sessionFactory.openSession();

Book book = session.get(Book.class, 1L);

**3. Spring Data JPA**

* **What it is:** A **higher-level abstraction** built on top of **JPA** and **Hibernate** (or any JPA provider).
* **Key Point:** Spring Data JPA:
  + Removes boilerplate code (like EntityManager, queries)
  + Auto-generates repository implementations at runtime
  + Integrates easily with Spring Boot
* **Ex:**

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByTitle(String title);

}