Hands on 4

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (JSR 338) | ORM framework and JPA implementation | Abstraction layer on top of JPA and its implementations (like Hibernate) |
| Provides Implementation | No | Yes | No (relies on JPA provider like Hibernate) |
| Code Writing Effort | Requires boilerplate code | Less than JDBC but still has boilerplate | Very minimal code required |
| Transaction Management | Not provided | Provided | Provided and auto-managed |
| Reduces Boilerplate | No | Partially | Yes |
| Query Writing | JPQL / Criteria API | HQL / JPQL | Method naming conventions generate queries automatically |
| Ease of Use | Moderate | Easier than JDBC | Very easy and beginner-friendly |
| Commonly Used In | General Java persistence standards | Standalone Java + Spring apps | Mostly Spring Boot applications |
| Repository Support | No | Needs custom DAOs | Built-in repository support with JpaRepository or CrudRepository |
| Example for Save Operation | Requires writing full code for session and transaction | Use session.save() with transaction | Just call employeeRepository.save(employee) |

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

**Code Comparison Example**

In Hibernate, to add an employee, we have to manually open a session, start a transaction, save the object, handle exceptions, and close the session. The code is longer and needs more boilerplate.

In Spring Data JPA, we just create a repository interface that extends JpaRepository, and we can directly call the save() method. The transaction management, session handling, and error handling are done automatically by Spring. This makes the code much shorter and easier to maintain.