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

## **Java Persistence API (JPA)**

• JPA stands for Java Persistence API and is a Java specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.

• It is not a framework or implementation—it’s just a set of interfaces and rules.

• JPA enables developers to work with Java objects without writing SQL queries directly.

• Provides annotations like @Entity, @Id, @GeneratedValue, @OneToMany, etc.

• It supports JPQL (Java Persistence Query Language) for querying the database using entity objects.

• Common JPA implementations include Hibernate, EclipseLink, OpenJPA, etc..

## **Hibernate**

• Hibernate is a popular ORM (Object Relational Mapping) framework and the most widely used implementation of JPA.

• It maps Java classes to database tables using annotations or XML configuration.

• Handles connection management, SQL generation, caching, lazy loading, and more.

• Developers can use Hibernate APIs directly or via JPA.

• Hibernate adds extra features like:  
 1. Criteria API (for dynamic queries)  
 2. First-level and second-level caching  
 3. Hibernate Query Language (HQL)

**Example using Hibernate :**  
public Integer addEmployee(Employee employee) {  
 Session session = factory.openSession();  
 Transaction tx = null;  
 Integer employeeID = null;  
 try {  
 tx = session.beginTransaction();  
 employeeID = (Integer) session.save(employee);  
 tx.commit();  
 } catch (HibernateException e) {  
 if (tx != null) tx.rollback();  
 e.printStackTrace();  
 }

finally {  
 session.close();  
 }  
 return employeeID;  
}

**Spring Data JPA**

• Spring Data JPA is a part of the Spring Data project, and it provides an abstraction layer over JPA.

• It does not implement JPA itself but simplifies the development of data access layers.

• Reduces boilerplate code by providing ready-to-use repository interfaces.

• Allows creation of queries just by defining method names.

• Integrates seamlessly with Spring Boot and supports automatic transaction management.

• Commonly used interfaces: JpaRepository, CrudRepository, PagingAndSortingRepository.

**Example using Spring Data JPA :**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}  
  
@Service  
public class EmployeeService {  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}

## **Differences**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification | Implementation | Abstraction over JPA + Repository APIs |
| **Provides implementation?** | No | Yes | No |
| **Boilerplate code** | Medium | High | Low |
| **Query support** | JPQL | HQL, Criteria API | Derived queries, JPQL |
| **Transaction Management** | Manual/Programmatic | Manual or with Spring integration | Auto-managed by Spring |

## **Conclusion**

• JPA defines how to work with persistent data in Java.

• Hibernate is a powerful implementation of JPA with additional features.

• Spring Data JPA further simplifies working with JPA by abstracting repository creation and handling CRUD operations with minimal code.