# Difference Between JPA, Hibernate, and Spring Data JPA

### 1. Java Persistence API (JPA)

JPA (Java Persistence API) is a **specification** (JSR 338) for object-relational mapping (ORM) in Java. It allows mapping Java objects to database tables in a standardized way.

### Key Features:

Defines interfaces and annotations for ORM.

No implementation; needs a provider like Hibernate.

Helps manage data persistence, transactions, and queries.

Standardized approach to database access in Java EE and Spring.

### Common Annotations:

@Entity

@Id

@GeneratedValue

@Table

@Column

@OneToMany, @ManyToOne, etc.

### 2. Hibernate

Hibernate is a **JPA provider** and an ORM tool that offers the actual implementation for JPA interfaces. It is widely used and includes additional features beyond the JPA standard.

### Key Features:

Implements JPA.

Offers extra features like caching, lazy loading, custom SQL support.

Provides both JPA and its own native APIs.

### Code 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;

}

### 3. Spring Data JPA

Spring Data JPA is a part of the Spring ecosystem that **simplifies JPA usage** by providing a higher level of abstraction over JPA provider implementations like Hibernate.

### Key Features:

No implementation of JPA itself.

Reduces boilerplate code using repository interfaces.

Automatically handles common CRUD operations.

Integrates seamlessly with Spring Boot.

### Code Example Using Spring Data JPA:

**EmployeeRepository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}

**EmployeeService.java**

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

## Comparison

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation | Abstraction over JPA & Hibernate |
| Implementation | No | Yes | No |
| Boilerplate Code | Medium | High | Very Low |
| Ease of Use | Medium | Medium | High |
| Transactions | Manual setup | Manual setup | Automatic (with @Transactional) |
| CRUD Operations | Manual Implementation | Manual Implementation | Auto-generated via Repository |

## Summary

**JPA** provides the what to persist data but not the how.

**Hibernate** is the how—an actual tool that implements JPA.

**Spring Data JPA** makes it easier and cleaner to use JPA and Hibernate by removing boilerplate code.