# Difference between JPA, Hibernate and Spring Data JPA

## 1. Java Persistence API (JPA)

JPA is a specification (an interface) for managing relational data in Java applications.

### Key Features:

* It is not a framework or implementation.
* Defines how to persist, retrieve, and manage data from a relational database.
* It is defined by the Java Community Process (JCP) under JSR 338.
* Provides annotations like @Entity, @Table, @Id, @OneToMany, etc.
* It provides a set of standardized APIs for object-relational mapping (ORM)
* Works with any JPA-compliant implementation (like Hibernate, EclipseLink)

### Example (JPA Annotations):

@Entity  
@Table(name = "users")  
public class User {  
 @Id  
 @GeneratedValue  
 private Long id;  
  
 private String name;  
}

## 2. Hibernate

Hibernate is a popular implementation of the JPA specification.

### Key Features:

* It is a framework.
* It can be used with or without JPA.
* It implements JPA. Adds additional features on top of JPA (e.g., better caching, filters, more fetching strategies)
* Provides hibernate.cfg.xml or properties-based configuration.
* Most widely used ORM tool in Java ecosystem.

### Example (Hibernate):

/\* Method to CREATE an employee in the database \*/  
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 part of the Spring Data project. It is a Spring-based abstraction over JPA, which simplifies repository implementation.

### Key Features:

* Built on top of JPA and uses Hibernate (or another JPA provider) internally.
* Automatically generates query methods using method naming conventions.
* Reduces boilerplate code.
* Provides built-in CRUD operations and supports custom finder methods using method naming conventions (e.g., findByName, deleteByCode).
* Integration with Spring Boot for auto-configuration and database initialization.

### Example (Spring Data JPA):

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

## Relationship Summary

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | JPA | Hibernate | Spring Data JPA |
| Type | Interface | Framework (implementation) | Abstraction (on top of JPA) |
| Provides Annotations | Yes | Yes, inherits from JPA | Yes, uses JPA annotations |
| Requires JPA | - | Yes | Yes |
| Boilerplate Code | More | More | Low |
| Can be used alone | No | Yes | No (depends on JPA provider) |

## Conclusion

* Use JPA if you want to write portable code that works with any JPA provider.
* Use Hibernate if you need advanced features not available in JPA.
* Use Spring Data JPA for rapid development with Spring Boot to minimize boilerplate.