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

**Java Persistence API (JPA)**

* A specification (part of JSR 338) for persisting, managing, and accessing data from Java objects.
* Defines interfaces and annotations, but **does not** provide an actual implementation.
* Needs an implementation like Hibernate or EclipseLink to work.

**Hibernate**

* An **Object-Relational Mapping (ORM) framework**.
* Implements the JPA specification (i.e., it **is** a JPA provider).
* Also offers extra features beyond JPA (like caching, interceptors, etc.).
* Requires manual management of sessions and transactions (as shown in your code).

**Example:**

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**

* An abstraction layer built on top of JPA.
* **Does not** implement JPA itself; instead, it relies on an implementation like Hibernate.
* Focuses on removing boilerplate by providing ready-made repository interfaces.
* Simplifies CRUD, query methods, and transaction management.

**Example:**

// EmployeeRepository.java

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

// EmployeeService.java

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Summary Table:**

|  | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation of JPA | Abstraction over JPA |
| Purpose | Define standard | Provide ORM & extra features | Reduce boilerplate, simplify CRUD |
| Code Effort | Needs implementation | Manual session & tx mgmt | Declarative repositories & auto tx mgmt |