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

1. **Java Persistence API (JPA)**

* **Definition:** JPA is the **Java specification** (JSR 338) for **mapping Java objects** to relational database tables.
* **Key point:** JPA **does not provide an implementation** itself — it defines *how* to persist, read, and manage data, but not *how to do it technically*.
* **Example:** To actually *use* JPA, you need a concrete provider such as **Hibernate**.

1. **Hibernate**

* **Definition:** Hibernate is an **ORM (Object-Relational Mapping) framework** that **implements the JPA specification**.
* **Key point:** Hibernate handles:
  + Database connections
  + Sessions and transactions
  + SQL query generation
  + Object-to-table mapping
* **How it works:** Developers need to manage **Session**, **Transaction**, open/close connections, and handle exceptions manually.

1. **Spring Data JPA**

* **Definition:** Spring Data JPA is a **Spring framework module** that provides **an abstraction layer over JPA providers** (like Hibernate).
* **Key point:** It:
  + Removes boilerplate code
  + Provides ready-made CRUD operations
  + Uses JpaRepository interfaces
  + Manages transactions automatically using @Transactional
  + Works with Hibernate or any other JPA provider under the hood.

**Code Comparison**

**Example 1: Hibernate Code**

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;

}

**Key points:**

* Developer writes **session management** and **transaction handling** code manually.
* More lines of code, more error handling.

**Example 2: Spring Data JPA Code**

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**Key points:**

* Uses **JpaRepository** to get CRUD operations automatically.
* **No manual session/transaction handling.**
* **@Transactional** manages transactions automatically.
* Less boilerplate, easier to maintain.

**Conclusion**

* JPA is a specification whereas Hibernate is an implementation of JPA and Spring Data JPA is a spring abstraction on top of JPA.
* There are no Boilerplates in JPA while Hibernate has more code and Spring Data JPA has minimal code.
* JPA needs provider for usage while Hibernate provides ORM & session and Spring Data JPA user hibernate.

**\*There is no runtime output which can be produced for this Hands-On.**