**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA**   
  
 Java Persistence API (JPA)

* JSR 338 Specification for persisting, reading and managing data from Java objects
* Does not contain concrete implementation of the specification
* Hibernate is one of the implementation of JPA

Hibernate

* ORM Tool that implements JPA

Spring Data JPA

* Does not have JPA implementation, but reduces boiler plate code
* This is another level of abstraction over JPA implementation provider like Hibernate
* Manages transactions

**Refer code snippets below on how the code compares between Hibernate and Spring Data JPA  
 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;

}

**Spring Data JPA** EmployeeRespository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowire

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

​​​​​​​   
  
 **Reference Links:**   
<https://dzone.com/articles/what-is-the-difference-between-hibernate-and-sprin-1>   
<https://www.javaworld.com/article/3379043/what-is-jpa-introduction-to-the-java-persistence-api.html>

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

**Definition:** Official specification (JSR 338) for persisting, reading, and managing data between Java objects and a relational database.

**Key Points:**

1. It's just a **specification**, not an implementation.
2. Provides annotations like **@Entity, @Id, @Table,** etc.
3. Needs an implementation like **Hibernate**, **EclipseLink**, etc.

### **Hibernate**

**Definition:** A **popular ORM (Object Relational Mapping) tool** and the most widely used implementation of JPA.

**Key Features:**

1. Provides **actual code** to implement JPA functionalities.
2. Offers additional features like caching, lazy loading, etc.
3. Can be used with or without JPA.

**Hibernate 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**

**Definition:** A **Spring-based abstraction layer** built on top of JPA (e.g., Hibernate).

**Key Features:**

1. **No need to write boilerplate DAO code** (like openSession(), beginTransaction(), etc.).
2. Uses **repositories** (JpaRepository, CrudRepository, etc.).
3. Automatically handles transactions, queries, etc.
4. Simplifies CRUD and complex query creation using method names or @Query.

**Spring Data JPA Code 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);

}