# WEEK 3 -Spring Data JPA with Hibernate-Hands On \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Difference between JPA, Hibernate, and Spring Data JPA

# *Hibernate, JPA, and Spring Data JPA are all frameworks used for interacting with relational databases in Java applications.*

# *****Hibernate******is an Object-Relational Mapping (ORM) framework that provides a high-level API for interacting with relational databases. It allows you to map Java objects to database tables and perform database operations using a high-level API, instead of writing low-level SQL code.*

*****JPA******(Java Persistence API) is a specification that defines a set of interfaces and annotations for working with relational databases in Java applications. JPA provides a common API for ORM frameworks like Hibernate, EclipseLink, and OpenJPA, making it easier to switch between different ORM frameworks without changing your code.*

*****Spring Data JPA******is a part of the Spring Framework that provides a higher-level, easier-to-use API for working with JPA. It reduces the amount of boilerplate code required to interact with a database using JPA, and provides a repository abstraction that provides a number of methods out of the box for performing CRUD operations on JPA entities.*

*In summary, Hibernate is an ORM framework that provides a high-level API for interacting with databases.*

*JPA is a specification that defines a common API for ORM frameworks like Hibernate.*

*Spring Data JPA is a part of the Spring Framework that provides a higher-level, easier-to-use API for working with JPA.*

**Code Comparison:**

**Hibernate 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 Example**  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
}  
  
@Autowired  
private EmployeeRepository employeeRepository;  
  
@Transactional  
public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
}

### ****Comparison:****

**Type:**

**JPA**: *It is an interface for ORM.*

**Hibernate**: *It is a framework that implements JPA.*

**Spring Data JPA**: *It is an abstraction layer over JPA and its providers like Hibernate.*

**Implementation:**

**JPA**: *It does not provide any implementation, it only defines the rules.*

**Hibernate**:*It provides the actual implementation.*

**Spring Data JPA**: *It does not provide implementation,it relies on Hibernate or other JPA providers.*

**Boilerplate Code:**

**JPA**: *Requires moderate amount of code for setup and usage.*

**Hibernate***: Requires high amount of manual code (Example: session handling).*

**Spring Data JPA**: *Requires very little code.*

**Transaction Handling:**

**JPA**: *Handled manually or via JTA .*

**Hibernate**: *Managed manually using Transaction objects.*

**Spring Data JPA**: *Automatically managed using Spring’s @Transactional annotation.*

**Main API Used:**

**JPA**: *Uses EntityManager, Entity, and related annotations.*

**Hibernate**: *Uses Session, Transaction, and native Hibernate APIs.*

**Spring Data JPA**: *Uses JpaRepository, CrudRepository, and method-based queries.*