# Hands-on 4:

# Difference between JPA, Hibernate, and Spring Data JPA

### Java Persistence API (JPA)

• JPA (Java Persistence API) is a standard specification (JSR 338) for ORM in Java.

• It defines how Java objects are persisted into relational databases.

• JPA itself does not contain any implementation.

• To use JPA, you need an implementation like Hibernate or EclipseLink.

### JPA (Java Persistence API)

### Code Snippet (Pure JPA)

### 

import javax.persistence.\*;  
  
@Entity  
public class User {  
 @Id  
 private int id;  
 private String name;  
 // getters and setters  
}  
  
// Usage  
EntityManager em = Persistence.createEntityManagerFactory("myPU").createEntityManager();  
em.getTransaction().begin();  
User user = new User();  
user.setId(1);  
user.setName("John");  
em.persist(user);  
em.getTransaction().commit();  
em.close();

### Hibernate

• Hibernate is a widely-used implementation of the JPA specification.

• It adds powerful features like HQL, caching, and lazy loading.

• Hibernate can operate independently or serve as a JPA provider.

Code Snippet (Using Hibernate)

import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
public class HibernateExample {  
 public static void main(String[] args) {  
 SessionFactory factory = new Configuration().configure().buildSessionFactory();  
 Session session = factory.openSession();  
 session.beginTransaction();  
 User user = new User();  
 user.setId(2);  
 user.setName("Jane");  
 session.save(user);  
 session.getTransaction().commit();  
 session.close();  
 factory.close();  
 }  
}

### Spring Data JPA

• Spring Data JPA is an abstraction layer on top of JPA, developed by the Spring team.

• It reduces boilerplate code using repository interfaces like JpaRepository.

• It seamlessly integrates with Spring Boot and manages queries, transactions, and more.

Code Snippet (Spring Data JPA)

import org.springframework.data.jpa.repository.JpaRepository;  
import javax.persistence.\*;  
  
@Entity  
public class User {  
 @Id  
 private int id;  
 private String name;  
 // getters and setters  
}  
  
public interface UserRepository extends JpaRepository<User, Integer> {  
}  
  
// In Service  
@Autowired  
private UserRepository userRepository;  
  
public void saveUser() {  
 User user = new User();  
 user.setId(3);  
 user.setName("Mike");  
 userRepository.save(user);  
}

### Feature Comparison

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation | Abstraction over JPA & Hibernate |
| Implementation | No | Yes | No |
| Boilerplate Code | Medium | High | Very Low |
| Ease of Use | Medium | Medium | High |
| Transactions | Manual setup | Manual setup | Automatic (with @Transactional) |
| CRUD Operations | Manual Implementation | Manual Implementation | Auto-generated via Repository |

### Summary

• Use JPA for vendor-independent ORM and standards compliance.

• Choose Hibernate if you require additional ORM features like caching or HQL.

• Opt for Spring Data JPA for ease of development in Spring Boot projects with minimal code.