Hands-on 4: Understanding JPA, Hibernate, and Spring Data JPA

# Introduction

In Java, persistence refers to the ability of an application to save data so that it can be reused even after the application restarts. Instead of writing lots of SQL code, developers can use frameworks like JPA, Hibernate, and Spring Data JPA to simplify this process.

# What is JPA?

Java Persistence API (JPA) is a specification (a set of rules and interfaces) that tells us how Java objects should be stored and retrieved from databases. It’s part of the Java EE standard and is defined under JSR 338. JPA itself does not provide an actual implementation. You need a tool like Hibernate to make it work.

Think of JPA like a blueprint. Imagine JPA is a recipe book (specification), but to cook the food, you still need a chef (Hibernate or EclipseLink, etc.).

# What is Hibernate?

Hibernate is a popular Object-Relational Mapping (ORM) tool that follows JPA rules. It allows you to map Java classes to database tables without writing SQL manually. Hibernate is more than just JPA. It includes extra features like caching, lazy loading, and advanced mappings.

Hibernate Example 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;  
}

# What is Spring Data JPA?

Spring Data JPA is a wrapper on top of JPA (and Hibernate) that removes repetitive code. It allows you to focus on your business logic and not worry about managing sessions or transactions manually. It provides predefined repository interfaces like JpaRepository, which already include basic operations like save(), delete(), findById().

Spring Data JPA Example:

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

# Summary of Differences

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | JPA | Hibernate | Spring Data JPA |  |  |
| Type | Specification | Implementation of JPA | Abstraction over JPA |  |  |
| SQL Management | Manual | Partial | Mostly automatic |  |  |
| Requires Boilerplate | Yes | Yes | No |  |  |
| Extra Features | No | Yes | No (relies on Hibernate) |  |  |
| Transactions | Not managed | Developer-managed | Managed by Spring |  |  |
| Ease of Use | Moderate | Easy | Very Easy |  |  |

# 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