Superset ID: 6373431

COGNIZANT DIGITAL NURTURE 4.0 JAVA FSE

WEEK-3

Spring Data JPA

HandsOn 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

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); }

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

* **JPA (Java Persistence API):** Reiterate that it's a specification (like a blueprint) for object-relational mapping in Java. It defines *how* you should persist Java objects to a relational database, but it doesn't provide the actual implementation.
  + JPA itself does not contain any concrete implementation. It's like a blueprint or a contract that defines *how* an ORM tool should behave.
  + **Core Concepts:** Defines concepts like entities, entity managers, persistence units, queries (JPQL), and lifecycle callbacks.
* **Hibernate:** Confirm it's a popular concrete implementation of the JPA specification. It's an ORM (Object-Relational Mapping) tool that handles the actual mapping of Java objects to database tables and executes SQL.
  + Hibernate provides the actual code and logic to perform the persistence operations defined by JPA.
  + When used directly (without higher-level abstractions like Spring Data JPA), it often requires more explicit management of sessions and transactions.
* **Spring Data JPA:** Emphasize that it's not a JPA implementation itself. Instead, it's a higher-level abstraction built on top of JPA providers (like Hibernate). Its primary goal is to significantly reduce the boilerplate code required for data access by providing powerful conventions and automatically generating repository implementations. It also manages transactions.
  + Through conventions and method name parsing, Spring Data JPA automatically generates repository implementations, meaning you write very little code for common CRUD operations.
  + Seamlessly integrates with Spring's powerful features like dependency injection, AOP (Aspect-Oriented Programming), and declarative transaction management (@Transactional).

**CODE ANALYSIS**

Hibernate (Direct API Usage)

The Hibernate example demonstrates a more verbose and "hands-on" approach. It requires manual handling of the Session (the primary interface for database operations in Hibernate) and explicit transaction management (beginning, committing, and rolling back transactions). This gives fine-grained control but introduces significant boilerplate code, especially for common CRUD operations.

Spring Data JPA

The Spring Data JPA example showcases its power in reducing boilerplate.

By simply extending JpaRepository, the EmployeeRepository interface automatically inherits a rich set of CRUD methods (save(), findById(), findAll(), etc.) without any implementation code.

Spring Data JPA internally uses the configured JPA provider (e.g., Hibernate) to perform this operation.

In conclusion, JPA sets the rules, Hibernate plays by those rules (and adds its own capabilities), and Spring Data JPA makes playing by those rules much, much easier by taking care of repetitive tasks and seamlessly integrating with the broader Spring framework. For modern Spring-based applications, Spring Data JPA is the recommended approach for data access due to its efficiency and developer productivity gains.