Week-3

Spring Data JPA

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

**Java Persistence API (JPA)**

* A specification (JSR 338) that defines how Java objects should be stored and retrieved from relational databases.
* It acts as a **standard API** for object-relational mapping (ORM) in Java EE and SE applications.
* **Does not provide implementation**, only defines interfaces like EntityManager, EntityTransaction, and concepts like @Entity, @Id, JPQL, etc.
* Requires an implementation such as **Hibernate**, **EclipseLink**, etc.

**Hibernate**

* A **concrete ORM framework** that implements the JPA specification.
* It provides the actual **code that performs database operations**, caching, lazy loading, dirty checking, etc.
* Can be used with or without JPA. When used directly, it involves **manual session handling** and explicit transaction management.
* Supports both **JPA standard** features and **Hibernate-specific enhancements**.

**Spring Data JPA**

* A part of the **Spring Data** project. It sits on top of JPA and JPA providers like Hibernate.
* It is **not a JPA provider**; instead, it simplifies and abstracts data access.
* Reduces the amount of boilerplate code by using **repository interfaces** and **method name conventions** (e.g., findByName(), deleteById() etc.).
* Integrates seamlessly with Spring features like **Dependency Injection**, @Transactional, and AOP.

CODE COMPARISON

public Integer saveEmployee(Employee emp) {

Session session = sessionFactory.openSession();

Transaction tx = null;

Integer id = null;

try {

tx = session.beginTransaction();

id = (Integer) session.save(emp);

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return id;

}

Spring Data JPA (Repository Interface)

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

public void saveEmployee(Employee emp) {

employeeRepository.save(emp);

}

}

CODE ANALYSIS

**Hibernate**

* Requires **manual session creation**, **explicit transactions**, and **exception handling**.
* Offers **low-level control** over persistence, but results in **boilerplate-heavy** code

**Spring Data JPA**

* Minimal code needed for common tasks.
* By extending JpaRepository, you automatically inherit methods like:
  + save(), findAll(), findById(), deleteById() etc.
* Enhances **developer productivity** by focusing on business logic rather than plumbing code

**Conclusion**

* **JPA** is like the **rulebook** – it sets the guidelines for ORM.
* **Hibernate** is the **player** – it follows the rules and adds some of its own.
* **Spring Data JPA** is the **referee and manager** – it ensures smooth play, removes overhead, and enhances productivity.