Super set id : 4991632

EX_4_Difference Between JPA , Hibernate and Spring JPA

1. JPA (Java Persistence API)

JPA is a *standard specification* in Java that defines how you can map Java objects (like a Student or Employee class) to database tables, and how to interact with the database using Java objects.

It only provides the *rules* and *interfaces* but does **not** contain any *actual code* that does the database operations.

It is mainly used because it is a standard, many tools (called implementations) follow its rules, so your code can be portable across these tools.

Example Code (Defining a simple entity using JPA):

java

```
import javax.persistence.Entity;

import javax.persistence.Id;


@Entity  // This marks the class as a JPA entity (mapped to a table)

public class Student {

  @Id  // This marks 'id' as the primary key

  private int id;

  private String name;

}
```

- This *only* defines structure and rules, **not** how to save or fetch data.

2. Hibernate

Hibernate is one of the most popular **implementations** of JPA. It is a tool (or *framework*) that follows the JPA specification and provides the *actual working code* to save, update, delete, or query objects in your database.

Besides implementing JPA, Hibernate offers additional features like caching, lazy loading, and more options for querying.

It works *directly* with Hibernate's APIs (like Session and Transaction) to manage data in your database.

Example Code (using Hibernate directly):

java

```
Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();


Student student = new Student(4991632, "eshwar");

session.save(student);  // Save student in the database


tx.commit();

session.close();
```

- You have to **manually manage sessions and transactions**, which involves writing more code.

3. Spring Data JPA

Spring Data JPA is a project by Spring Framework that makes it *much easier* to use JPA (and Hibernate) by reducing the amount of code you need to write.

Instead of manually handling sessions or transactions, Spring Data JPA provides the **Repository** pattern so you can just focus on your business logic.

here we define an interface (e.g., StudentRepository) that extends one of Spring Data's interfaces and Spring automatically provides implementations for basic CRUD (Create, Read, Update, Delete) operations.

Example Code (Spring Data JPA usage):

java

```
import org.springframework.data.jpa.repository.JpaRepository;


public interface StudentRepository extends JpaRepository<Student, Integer> {

}
```

And in your service or controller:

java

```
@Autowired

private StudentRepository repository;


public void addStudent() {

  Student student = new Student(4991632, "eshwar");

  repository.save(student);  // No need to manage sessions or transactions

}
```

- Spring Data JPA **uses Hibernate internally** by default, but hides all complexity from you.