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

**JAVA PERSISTENCE API(JPA)**

The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.Standard/Specification only — defines how to persist Java objects, but not how it's implemented.

* Defined by JSR 338
* Provides annotations like @Entity, @Id, @Table, etc.
* Requires an implementation like Hibernate, EclipseLink, etc.
* You write queries using JPQL (Java Persistence Query Language)

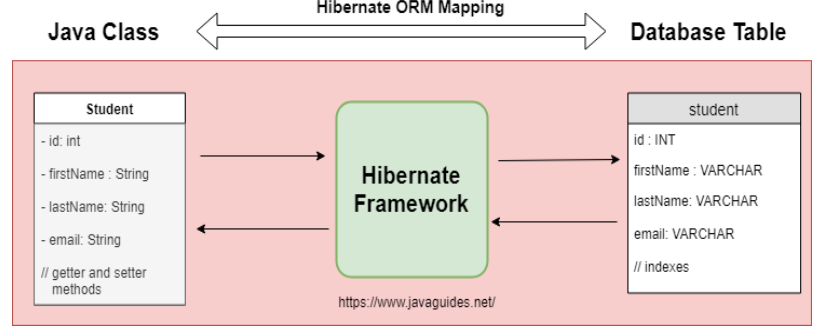
**HIBERNATE**

Hibernate is an object-relational mapping solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.

**Example**: Below diagram shows an *Object Relational Mapping* between the **Student** Java class and the **students**tablein the database.

* Implements all JPA specs + provides extra features
* Manages object-relational mapping (ORM)
* Provides both JPA-style API and native Hibernate API
* Requires more boilerplate if used directly

****

**Code Snippet:**

/\* 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();

      }

      returnemployeeID;

   }

**SPRING DATA JPA**

Spring Data is a part of the Spring Framework. The goal of Spring Data repository abstraction is to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Spring Data JPA is not a JPA provider. It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider (like Hibernate).

* Uses Spring’s dependency injection and transaction management
* No need to manually open sessions or manage transactions
* Reduces CRUD code to interfaces (e.g., JpaRepository)
* Uses @Repository, @Transactional, and auto-wired component

**Code Snippet:**

EmployeeRespository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowire

  privateEmployeeRepositoryemployeeRepository;

@Transactional

public void addEmployee(Employee employee) {

  employeeRepository.save(employee);

  }

**What Is the Difference Between Hibernate and Spring Data JPA?**

Hibernate is a JPA implementation, while Spring Data JPA is a JPA Data Access Abstraction.

Spring Data offers a solution to GenericDao custom implementations. It can also generate JPA queries on your behalf through method name conventions.

With Spring Data, you may use Hibernate, Eclipse Link, or any other JPA provider. A very interesting benefit is that you can control transaction boundaries declaratively using the @Transactional annotation.

Spring Data JPA is not an implementation or JPA provider, it's just an abstraction used to significantly reduce the amount of boilerplate code required to implement data access layers for various persistence stores.

Hibernate provides a reference implementation of the Java Persistence API that makes it a great choice as an ORM tool with the benefits of loose coupling.

**CONCLUSION**

* JPA is the blueprint
* Hibernate builds the house
* Spring Data JPA is the smart home automation system!