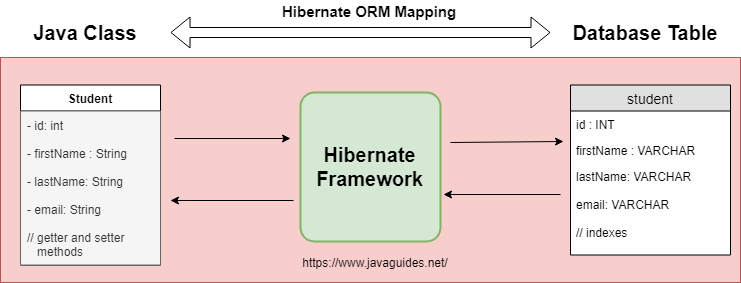
**JPA**

JPA stands for Java Persistence API. Think of it as a set of guidelines or a blueprint for how Java objects should be stored and retrieved from a relational database. JPA is a specification—it tells you *what* you can do, but not *how* to do it. It defines things like annotations (@Entity, @Id, etc.) and interfaces, but it doesn’t provide any actual code to make things happen.

* Key Points:
  + A **specification** (not an implementation).
  + Part of Java EE / Jakarta EE.
  + Defines how Java objects are mapped to relational database tables.
  + Requires a **provider** to implement it (e.g., Hibernate, EclipseLink).

The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database.



**Hibernate**

Hibernate is where the magic happens! It’s the most popular implementation of the JPA specification. Hibernate takes the rules set by JPA and provides the actual code that interacts with your database. It handles the heavy lifting: opening connections, managing transactions, converting Java objects to database rows, and vice versa.

* Key Points:
  + A **popular ORM (Object Relational Mapping) framework**.
  + Provides a concrete **implementation of the JPA specification**.
  + Can be used **with or without Spring**.
  + Offers advanced features beyond JPA like caching, custom SQL, etc.

Hibernate Implementation(Code):-

Session session = factory.openSession();

Transaction tx = null;

try {

tx = session.beginTransaction();

session.save(employee);

tx.commit();

} catch (HibernateException e) {

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

e.printStackTrace();

} finally {

session.close();

}

JPA uses the EntityManager interface provided by the specification. Uses EntityManager for persistence. Still requires manual wiring and boilerplate if used without Spring

**Spring Data JPA**

Spring Data JPA is like the “easy button” for working with JPA in Spring applications. It’s not an implementation of JPA, but rather a framework that sits on top of JPA providers (like Hibernate) and makes your life easier. With Spring Data JPA, you can write less code—just define interfaces, and Spring will generate the implementation for you at runtime.

* Key Points:
  + A **Spring module** that builds on top of JPA and its provider (e.g., Hibernate).
  + **Does not implement JPA** itself.
  + **Abstracts boilerplate**: no need to write queries or transaction code manually.
  + Integrates smoothly with Spring features like DI, AOP, and transaction management.

Spring Data JPA Implementation(Code):-

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

Spring Data JPA auto-generates repository implementation and manages transactions internally. Most concise and abstracted. Spring handles everything: repository, transaction, and session lifecycle.