Jpa with Hibernate:-

JPA Application:-
----

any java application, that uses JPA api to perform persistnce operation (CRUD) operation with
the DB s/w is called as JPA application.

JPA architecture:-

Entity class or persistence class:-

--it is a class using which we map our table.

--if we are using the annotaion, then we need not map this class with the table inside the xml mapping file.

--an Entity class or persistence class is a java class that is developed corresponding to a table of DB.

--this class has many instance variables should be there as same as columns in the corresponding table

--we should take Entity class as a POJO class.

--we need to provide mapping information with the table in this class only using annotaitons.

Note:- when we gives this persistance /Entity class obj to the ORM s/w, then ORM s/w will decide the destination DB s/w based on the configuration done in a xml file which is called as hibernate-configuration file.

Configuration file:-

--it is an xml file its name is "persistence.xml".

--this file must be created under src/META-INF folder in normal java application, where as in maven or gradle based application this file should be inside the src/main/resources/META-INF folder

--this file content will be used by ORM s/w (ORM engine) to locate the destination DB s/w.

--in this file generally 3 types of details we specify:-

1.DB connection details

2.ORM specific details (some instruction to the ORM s/w like dialect info,show_sql ,etc)

3. annotation based entity/persistence class name.(optional from latest hibernate version)

Note:- generally we take this file 1 per DB basis.

--we should always create this configuration file by taking support of example applications inside
the project folder of hibernate download zip file or by taking the reffernce from the Google.
ex:-

persistence.xml:-

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
           version="2.0">


   <persistence-unit name="studentUnit" >

         <class>com.ratan.Student</class>
```

```
<properties>

        <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"
/>
       <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/ratandb" />
         <property name="javax.persistence.jdbc.user" value="root" />
         <property name="javax.persistence.jdbc.password" value="root" />

  /*
        <property name="hibernate.connection.driver_class"
value="com.mysql.cj.jdbc.Driver"/>
         <property name="hibernate.connection.username" value="root"/>
         <property name="hibernate.connection.password"  value="root"/>
         <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/ratandb"/>
    */



      </properties>
   </persistence-unit>
</persistence>
```

the root tag is :-

<persistence> with some xml-namespace

--the child tag of <persistence> tag is <persistence-unit>

--this <persistence-unit> has 2 child tags:-

**1. <class> tag ,:-using which we specify the Entity class name(fully qualified name) that used
annotations to map a table (optional from letest version of hibernate)**

2.<properties> tag :- using this tag,we specify some configuration details to the ORM s/w

Persistence-unit:- it is a logical name of the configuation of our DB and some other details.

How to get the Hibernate s/w:
=========================

1. download the hiberate s/w (zip file) and add the required jar file in the classpath of our project

2.maven approach:

hibernate-core jar file

**persistence.xml : take this file from sample application or from hibernate docs..
and modify it accordingly.**

ORM engine :-

--it is a specialized s/w written in java that performs translation of jpa calls into the sql call by using mapping annotation and configuration file details and send the mapped sql to the DB s/w using JDBC.

--ORM engine is provided by any ORM s/w.

steps to devlop the JPA application:-
--

1.create a maven project(change the java version) and add the hibernate-core dependency to the pom.xml.

2.add jdbc driver related dependency to the pom.xml

3.create a folder called "META-INF" inside src/main/resources folder, and create the "persistence.xml" file inside this folder by taking reference from Hibernate docs or from google.

example:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
             version="2.0">

   <persistence-unit name="studentUnit" >


<properties>

          <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"
/>
         <property name="javax.persistence.jdbc.url"
value="jdbc:mysql://localhost:3306/ratandb" />
           <property name="javax.persistence.jdbc.user" value="root" />
           <property name="javax.persistence.jdbc.password" value="root" />


       </properties>

   </persistence-unit>
</persistence>
```

step 4:- create as many Entity/Perssitence classes as there r tables in the DB, apply the at least 2 annotations to these classes

@Entity :- on the top of the class
@Id :- on the top of PK mapped variable

--if we apply above 2 annotations then our java bean class will become Entity or Persistence class.

--inside these classes , we need to take variable corresponding to the columns of the tables.

step 5:- create a client application and activate ORM engine by using JPA api related following classes and interface and perform the DB operations.

1.Persistence class

2.EntityManagerFactory

3.EntityManager

--if we use Hibernate core api then we need to use

Configuration class

SessionFactory(I)

Session(I)

example :

Student.java: // Entity class

```
package com.masai;

import javax.persistence.Entity;
import javax.persistence.Id;
```

```
@Entity
public class Student {

	@Id
	private int roll;

	private int marks;
	private String name;

	public Student() {
		// TODO Auto-generated constructor stub
	}

	public Student(int roll, String name, int marks) {
		super();
		this.roll = roll;
		this.name = name;
		this.marks = marks;
	}

	public int getRoll() {
		return roll;
	}

	public void setRoll(int roll) {
		this.roll = roll;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public int getMarks() {
		return marks;
	}

	public void setMarks(int marks) {
		this.marks = marks;
	}

	@Override
	public String toString() {
		return "Student [roll=" + roll + ", name=" + name + ", marks=" + marks + "]";
	}

}
```

Demo.java:

```
package com.masai;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class Demo {

	public static void main(String[] args) {

		EntityManagerFactory emf=
Persistence.createEntityManagerFactory("studentUnit");


		EntityManager em= emf.createEntityManager();


		Student student= em.find(Student.class, 20);

		if(student != null)
			System.out.println(student);
		else
			System.out.println("Student does not exist");

		em.close();


	}

}
```

--to get the Object from the DB we need to call :- find(--) method of EM object

this find(--) method takes 2 parameter

1.the Classname of the Object which we want,

2.the ID value for which we want the object.

Note:- when we call createEntityManagerFactory(-) method by suppliying persistence-unit name on the Persistence class,we will get the EntityManagerFactory object.

--this method loads the "persistence.xml" file into the memory

--EntityManagerFactory obj should be only one per application per DB.

this EMF obj contains :-

connection pool (readly available some jdbc connection obj)

some meta information

--this EMF is a heavy weight object.

--by using this EMF class only we create the EntityManager object.
--EMF is a heavy weight object, it should be one per application

EntityManager em= emf.createEntityManager();

Note:- inside every DAO method(for every use case) we need to get the EntityManager obj
--after performing the DB opeation for that use-case we should close the EM obj.

EM should be one per use-case (one per DAO method)

JPA application ----------------->EntityManager(I) --------------------->ORM engine ------>JDBC------------>DB s/w

Inserting a Record:
===============

--in order to perform any DML (insert update delete) the method calls should be in a transactional area.

em.getTransaction(); method return "javax.persistence.EntityTransaction(I) " object.

this EntityTransaction obj is a singleton object, i.e per EntityManager obj, only one Transaction object is created.

--to store the object we need to call persist(-) method on the EM object.

example:

Demo.java: (Insert operation)

```
package com.masai;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

public class Demo {

	public static void main(String[] args) {

		EntityManagerFactory emf=
Persistence.createEntityManagerFactory("studentUnit");


		EntityManager em= emf.createEntityManager();
```

```
                    Student student= new Student(30, "Ratan", 500);


//                  EntityTransaction et= em.getTransaction();
//
//                  et.begin();
//
//                  em.persist(student);
//
//                  et.commit();


                    em.getTransaction().begin();

                    em.persist(student);

                    em.getTransaction().commit();




                    System.out.println("done...");




                    em.close();


          }

}
```

Main.java:- Delete:-

```
public class Main {

          public static void main(String[] args) {
```

```
                    EntityManagerFactory
emf=Persistence.createEntityManagerFactory("studentUnit");

                    EntityManager em= emf.createEntityManager();


                    Scanner sc=new Scanner(System.in);

                    System.out.println("Enter roll to delete ");
                    int roll=sc.nextInt();

                    Student student= em.find(Student.class, roll);

                    if(student != null){

                              em.getTransaction().begin();

                              em.remove(student);

                              em.getTransaction().commit();


                              System.out.println("Student removed....");

                    }else
                              System.out.println("Student not found...");

                    em.close();


                    System.out.println("done");


          }

}
```

Main.java :- Update the marks:-
==========================

```
public class Main {

	public static void main(String[] args) {

		EntityManagerFactory
emf=Persistence.createEntityManagerFactory("studentUnit");

		EntityManager em= emf.createEntityManager();


		Scanner sc=new Scanner(System.in);

		System.out.println("Enter roll to give grace marks ");
		int roll=sc.nextInt();

		Student student=em.find(Student.class, roll); //if it returns the obj then the
obj will be in p.state


		if(student == null){
			System.out.println("Student does not exist..");
		}
		else
		{

			System.out.println("Enter the grace marks");
			int marks=sc.nextInt();

			em.getTransaction().begin();

			student.setMarks(student.getMarks()+marks);

			em.getTransaction().commit();

			System.out.println("Marks is graced...");

		}
		em.close();

		System.out.println("done");


	}
}
```

--in the above application we didn't call any update method, we just change the state of the persistence/entity obj
inside the transactional area, at the end of the tx , ORM engine will generate the update sql.

--this is known as the ORM s/w maintaining synchronization bt entity obj and the db table records.

--we have a method called merge() inside the EntityManager obj to update a record also.

Life-cycle of persistence/entity object:-
----

an entity obj has the 3 life-cycle state:-

1.new state/transient state

2.persistence state/managed state

3.detached state

1.new state/transient state:-

--if we create a object of persistence class and this class is not attached with the EM obj, then
this stage is known as new state/transient state

Student s=new Student(10,"ram",780);

2.persistence state:-

--if a persistence class obj or Entity obj is associated with EM obj, then this obj will be in persistence state.

ex:-

when we call a persist(-) method by supplying Student entity obj then at time student obj will be in persistence state

or

when we call find() method and this method returns the Student obj, then that obj will also be in persistence state.

Note:- when an entity class obj is in persisitence state ,it is in-sync with the DB table ,i.e any change made on that obj inside the tx area will reflect to the table automatically.

ex:-

```
                    Student s=new Student(150,"manoj",850); // here student obj is in transient
state .

                    em.getTransaction().begin();

                    em.persist(s); // here it is in the persistence state

                    s.setMarks(900);


                    em.getTransaction().commit();
```

detached state:-

--when we call close() method   or call clear() method on the EM obj, then all the associated entity obj will be in detached state.

--in this stage the entity objs will not be in-sync with the table.

Note:- we have a merge() method in EM obj, when we call this method by supplying any detached object then that detached object will bring back in the persistence state.

ex:-

Main.java:-

```
public class Main {

	public static void main(String[] args) {

		EntityManagerFactory
emf=Persistence.createEntityManagerFactory("studentUnit");

		EntityManager em= emf.createEntityManager();

		Student s= em.find(Student.class, 20); //persistence state

		em.clear(); //detached state

		em.getTransaction().begin();

		s.setMarks(500);

		//em.persist(s);// it will throw duplicate ID related exception
		em.merge(s); //persistence state

		em.getTransaction().commit();

		em.close();

		System.out.println("done");
	}
}
```

em.persist()

em.find()------------>persistence state-----------em.close(), em.clear()--------->detached state---->em.merge()--->reflect in the table.

--after merge() method, we can not do modification on that object(it will not be reflected.).