Hibernate

What is Framework?

*A framework is a body or platform of pre-written code which can be used by a programmer for the development of an application.
*A framework will not do anything by itself and it has to be controlled by a programmer.
*A java framework is a collection of classes, interfaces and some tools.
Ex:--Hibernate ,Spring , Ibatis, Tapestry, Struts,etc…

Types of Framework:-

There are 2 Different types of Framework

1>Invasive Framework

2>Non-Invasive Framework

1>Invasive Framework:-

A framework which will force the programmer to extends its classes and implement its interfaces is called as invasive framework.

Ex:-EJB framework, Struts,etc…

2>Non Invasive Framework:-

A framework which does not force the programmer to extends its classes and implements its interface is called as Non Invasive Framework.

Ex:-Hibernate, Spring ,TapeStry, etc…

Dis-Advantages of JDBC

1>Duplicate code or Boiler Plate Code

To insert a record using JDBC we need to execute the following step

1>Load and register the Driver

2>Establish a connection with the database server

3>Create a statement or a platform

4>Execute the SQL queries or statements

5>Close all the costly resources

-Similarly to update ,delete and fetch we need to follow the same steps which results in **Boiler plate code or Duplicate code.**

2>JDBC Does Not Support Automatic Table Creation

3>JDBC Does Not Provide Any Strategy For The Generation Of Primary Key .

4>Fetching The Data From The Multiple Tables By Using JDBC Requires Complex Join Queries .

5>JDBC does not support cache mechanism because of which the traffic between Java Application and the Database server increases and efficiency decreases.

Cache:-

It is a temporary layer of Storage which is used to store the data to serve the future request.

What is the advantage of Cache ?

It decreases the traffic between Application and the Database server.

To Solve all these problems Framework is the Solution

POJO(Plain Old Java Object)

Defination:-

Pojo is a class which will not have any special restrictions other than those which are imposed by java.

Rules are like :-

1>Rules for Keywords

2>Rules for identifiers.

3>Rules For Constructor

Specifications of POJO

*A POJO class must be public.

*All the fields of POJO class must be private.

* It should have a public no-arg constructor.

*It should have public getters and setters for all the fields.

*It should not extend any pre-specified class.
*It should not implement any pre-specified interfaces

Easy Loading:-

-If you use pojo easy loading will be there since it extends object class ,hence only object class methods has to be loaded.

Advantages of POJO:-

*Easy to understand

*Simple Design

*Easy maintenance.

*Readability can be improved.

*We can develop lightweight Applications.

*The application which uses pojo implementation is known as lightweight application.

Light-Weight Framework:-

A framework which makes use of POJO implementation is called as Light-Weight Framework.

ORM(Object Relational Mapping)

Definition:-

The process of converting an object into relational model(Table) is called as Object relational mapping .

*We can achieve Object Relational Mapping with the help of ORM tool.

*An ORM tool will help us to simplify the interaction between Java Application and the DataBase Server.

*An ORM tool will provide us a platform where we can deal with the Objects instead of Writing the SQL statements.

Ex:-Hibernate,ibatis,TapeStry,etc…

*ORM tool uses mapping file to convert class into table, fields into Columns.

Entity Class(POJO Class)

The class which represents table in the database we called Entity class or POJO class.

Rules to create an Entity Class

*An entity class must be public.
*The fields of entity class must be private.
*An entity class must have a no-arg constructor.
*An entity class must have public getters and setters for all the fields.
*An entity class must have a field which represents the primary key .

Specifications ORM

*Each entity class represents a table in the database Server.
*Every field of an entity class represents a column in the table.
*Every object of an entity class represents a record in the table.

1>What is Framework and Explain its types.

2>Why we need Hibernate for DB connectivity when we have JDBC.

3>Explain POJO

4>What is ORM?

Hibernate

*Hibernate is an open-Source, non-invasive, light-weight ORM tool which is used to achieve Object relational mapping.
*Hibernate provides a platform where we can directly deal with objects instead of writing the SQL queries.
*By using Hibernate we can simplify the interaction between the Java application and the database server.

*Hibernate also implements JPA , so we can also use JPA specifications with the help of Hibernate.
*.cfg file is used to give the information about the Driver Class, URL ,username, password.

Hibernate Architecture:-

Advantages of Hibernate:-

*It is an open-Source Framework.
*It is a light weight framework because of its POJO implementations.
*Hibernate supports the automatic table creation.
*Hibernate will also provides the strategy for the generation of Primary keys .
*Hibernate supports two levels of cache mechanism(ie 1st level cache and 2nd level cache because of which the traffic between java application and the database server will decrease and the efficiency will increase.
*Hibernate supports dialect.

Dialect:

*It is a component of ORM tool.
*It Helps to generate SQL Queries
*The orm tool has to communicate with different versions OF MySQL .
*It is like version(Ex:-Flavour of Langauge)
*Dialect is a class.

*To solve the version problem we have different versions of dialect.

Ex:-MySQL 5

MySQL 55

MySQL 8

*Hibernate Supports dialect which helps in Object Relational Mapping.

***Hibernate Supports HQL statements using which we can write database independent queries.**

HQL(Hibernate Query Language)

*HQL is database independent.
*It is a query language which is similar to SQL using which we can write database independent queries .
*In HQL we use entity class name instead of using the table name in the query.

SQL	HQL
select * from Employee;	select e from Employee e
select * from Employee where id=?;	select e from Employee e where e.id=:id(named parameter)
select * from Employee where salary=?;	select e from Employee e where e.salary=?1(numbered parameter)
select name from Employee;	Select e.name from Employee e

Coding :-

Steps to create a Hibernate Project

Step 1:- Create a Simple Maven Project

Step 2:- Add the following dependencies in pom.xml

1>Hibernate Core-Relocation(5.6.15 Final)

2>MySQL Connector Java(5.1.6)

*The root tag for pom.xml--------> <project>

Steps to add Hibernate Core-Relocation Dependency

*Open the browser and Search for Maven Repository.

*Open MavenRepository.com and search for Hibernate core-relocation.

*Click on Hibernate core-relocation and open

*Select 5.6.15 final version and open it.
*Copy the dependency and paste it in pom.xml inside
<dependencies> </dependencies>
*Follow the Steps from 1 to 5 to add mysql connector java dependency.

Note:-

We need to give info to the hibernate regarding database with the help of configuration file.

Hibernate Configuration File

*It is a file which is used to specify the resources required for the hibernate to connect with the database server.
*This file must be created in src/main/resources and it must be saved with an extension of .cfg.xml

*The root tag of this file is
<hibernate-configuration>

*The child element of <hibernate-configuration> is
<session-factory>

*The child element of <hibernate-configuration> is
<session-factory>

*<property> is a child element of <session-factory>
Which is used to configure the properties like driver class ,username,password,url,hbm2ddl.auto,dialect,show_sql,format_sql,etc…

```
<hibernate-configuration>
    <session-factory>
<!—Database  Properties-->
<property name =”connection.driver_class”>com.mysql.jdbc.Driver  </property>
<property name=”connection.url”> jdbc:mysql://localhost:3306/hibdb?createDatabaseIfNotExist=true</property>
<property name=”connection.username”>root</property>
<property name=”connection.password”>admin</property>
<!--Hibernate Properties-->
<property name=”hbm2ddl.auto”>update</property>
(1st time it will create later it will update)
<property name=”show_sql”>true</property>
(To display SQL query on Console window)
<property name=”dialect”>org.hibernate.dialect.MySQL55Dialect</property>
<property name=”format_sql”>true</property>
    </session-factory>
</hibernate-configuration>
```

Common Values for hbm2ddl.auto:

1.none:

- **Description**: No action will be performed.
- **Use Case**: When you don't want Hibernate to alter the database schema in any way.

2.validate:

- **Description**: Hibernate will validate the database schema. It ensures that the schema matches the entities' mappings but does not make any changes to the schema.
- **Use Case**: Useful in production to ensure that the schema is as expected without making any modifications.

3.update:

- **Description**: Hibernate will update the database schema. It will try to alter the existing schema to match the entities' mappings, adding missing tables, columns, etc., but it will not remove existing database objects.
- **Use Case**: Suitable for development environments where you want to keep the schema in sync with your entity mappings without losing data.

4.create:

- **Description**: Hibernate will create the database schema, dropping any existing tables and data first.
- **Use Case**: Useful for initial development and testing where you want a fresh schema on each run.

5.create-drop:

- **Description**: Hibernate will create the database schema at startup and drop it at shutdown.
- **Use Case**: Ideal for testing scenarios where you want a clean schema at the start and end of each test session.

6.none (alternative):

- **Description**: In some configurations, setting this to none or leaving it unset means no action will be taken, similar to validate.
- **Use Case**: When schema management is handled entirely outside of Hibernate.

org.hibernate.cfg.Configuration

*It is a class belongs to Hibernate which represents the hibernate configuration.
*This class will provide the methods to load the hibernate configuration file and also to build SessionFactory.
*Following are the important methods of Configuration Class

1>configure()

This method will consider hibernate.cfg.xml as the default configuration resource and returns Configuration Object.
*It is used to load the hibernate configuration file to the Hibernate.
*The return-type of this method is Configuration.

2>configure(String file)

This method will consider the argument as the configuration resource and returns the configuration object.
*The return-type of this method is Configuration.

org.hibernate.SessionFactory

*It is an interface belongs to hibernate Framework.

*It is used to check the configuration settings.

*SessionFactory provides a pool of sessions.

*When your application needs to interact with the database ,it typically requests a Session from the SessionFactory.

buildSessionFactory()

*It is a no-static factory /helper method which is used to create and return the implementation class object of Session Factory.

Hence the return type of this method is SessionFactory interface.

To test the Configuration write the below Code

```
public class TestCfg
{
      main()
            {
Configuration conf=new Configuration();
conf.configure();//or//conf.configure("hib.cfg.xml");//For loading
SessionFactory fac=conf.buildSessionFactory();//For validation
S.O.P(fac);//
            }
}
```

Hibernate Mapping File

The process of mapping an entity class with a database table is called Hibernate Mapping.

*If you want to map the class with table in the database we need to do Hibernate mapping.
*Hibernate mapping can be done in 2 different ways
1>By using Hibernate mapping file
2>JPA Annotation

1>Hibernate Mapping File

* A file which is used to map an entity class with a table in the database server is called as Hibernate Mapping file.

*A hibernate mapping file should be saved with an extension of .hbm.xml

* It must be created in src/main/resources folder.

*The root tag of hibernate mapping file is <hibernate-mapping>

*The <class> tag is the child tag of <hibernate-mapping> which is used to map the entity class with the table in the database.

*The <id> tag is the child tag of <class> tag which is used to specify the details of primary key(It is used to represents primary key column in the table)

* <generator> tag is the child tag of <id> which is used to provide the generation strategy for primary key .

***Following are the important Generator Classes(Strategies)**

assigned sequence native uuid sequencehilo
increment hilo identity guid

*The <property> tag is the child tag of <class> which is used to specify the details of column.

Following are the important attributes of <property> tag

not-null ,unique ,length. etc

employee.hbm.xml

```
<hibernate-mapping>
        <class  name=”org.jsp.Employee”  table=”employee”>
                                        //name is the attribute
                                        //table attribute is optional
                                        //If the table is present it will map else it will create .
                                        //<id> tag represent the primary key
            <id name =”id”  column=”id”>
                <generator  class=”identity”/>
            </id>
            <property name=”name” column=”name” not-null=”true”/>
            <property name=”desg”   column=”desg”   not-null=”true”/>
        </class>
</hibernate-mapping>
```

Steps to map Employee class with Employee Table

Step 1:-

Create Employee class

Employee

id, name, desg, salary, phone, password

//Setters and getters//

Step 2:-

Create employee.hbm.xml in src/main/resources

Use generator class as identity.

Step 3:-

Write the following code in hibernate.cfg.xml file inside <session-factory> element of hibernate.cfg.xml

<mapping resource="employee.hbm.xml"/>

Step 4:-

Execute TestCfg class

```
+class TestCfg
{
        main()
            {
              Configuration conf=new Configuration();
                              conf.configure();
              SessionFactory sef=conf.buildSessionFactory();
                              SYSO(sef);
            }
}
```

WE have created Configuration file ,mapping file and we have checked the configuration also.

Now we need to save the record into the table

For that save( ) is there in the Session interface.

```
Configuration conf=new Configuration();
conf.configure();
SessionFactory sef=conf.buildSessionFactory();
Session ses=sef.openSesion();
ses.save();
```

org.hibernate.Session:-

*It is an interface present in Hibernate Framework.

*Session in Hibernate like a conversation/interaction between your application and the database.

*Session Interface provides some methods using which we can save,update ,fetch and delete the records from the database server.

*Following are the important methods present in Session Interface

1>save(Object)
2>update(Object)
3>saveOrUpdate(Object)
4>get(Class<T>,Serializable)
5>load(Class<T>,Serializable)
6>delete(Object)

Syntax:-

```
Session ses=sef.openSession();
```

openSession():-

* It is a method present in SessionFactory interface
*openSession() creates and returns the implementation class object of Session interface.

*Hence the return type of openSession() is Session interface.

save(Object):-

*It is a method present in Session Interface
*This method is used to save a record in the database server.
*This method will save the record either by assigning the generated identifier or by using the assigned identifier(by default) and then returns the generated identifier.
*The return type of this method is Serializable.
*save() returns the identifier (primary key) of the saved entity as a Serializable object

org.hibernate.Transaction

*It is an interface present in Hibernate Framework.

*It is used to represent the hibernate Transaction.

Transaction means:-Sequence of operations which are treated as Single unit of work. Either all the operations has to be executed successfully or if any operation fails then rollback the complete operation from the beginning.

Ex:-Balance Enquiry or Money Withdrawal at ATM

Syntax:-

Transaction tran=ses.beginTransaction();

beginTransaction():-

*It is a method present in Session interface.

*beginTransaction() is a non-static factory/helper method which is used to create and return the implementation class object of Transaction interface.

*Hence the return type of this method is Transaction.

Note :-

Another Syntax for Transaction

Transaction tran=ses.getTransaction();

*Transaction interface provides some methods for Transaction in Hibernate

1>begin():-

2>commit()

*Execution:-

Code to insert a record into the employee table by using Session interface.

Use identity and after that assigned that’s fine while inserting

But first assigned later identity will not work(hibernate will not modify that)

To Fetch The Record

Eager and Lazy Concept:-

get(Class<T>,Serializable id):-

*It is a method present in Session interface.

*This method is used to fetch a record from the table by using an identifier

*This method will search for the specified identifier in a table which is mapped with the specified entity class.

*If the table have the specified primary key ,this method will create an object of 1st argument type,it will initialize that object and returns the reference ,else it will return null.

*It will throw TypeMisMatchExeception, if the second argument type is not same as the type of primary key in the specified entity class.

*It will throw UnKnownEntityException ,if the 1St argument is not an entity ,it will throw IllegleArgumentException if the second argument Is null.

*The return type of this method is same as the type of 1st argument.

(ie If we pass Employee.class as an argument return type would be Employee).

load(Class<T>,Serializable id):-

*It is an method present in Session interface

*This method is used to fetch a record from the table by using an identifier(primary key)

*This method will assume that identifier is present and returns the proxy object by assigning the passed identifier.

*This method will initialize the object only when we try to use the object .

*It will throw ObjectNotFoundException when we try to use an object which is not there in the database.

Note:-Below 3 points are same as get() points

*It will throw TypeMisMatchException if the 2nd argument type is not same as the type of Primary key in specified entity class.

*It will throw UnKnownEntityExeption if the 1st argument is not an entity.It will throw IllegleArgumentException if the 2nd argument is null.

*The return type of this method is same as the type of 1st argument.

Note:-One extra thing about load() is it throws ObjectNotFoundException but

get() returns null .

Usage of Proxy:-

Imp Note:If you want to delete a record from the database normally first you need to fetch by using get() but if you use get() --→it hits the database server and returns the real object

But to delete the record we can use proxy object also using load()-→load() will not hit the database server ,it gives proxy object which contains only identifier which is enough to delete the record.

The Difference between get() and load()

load()	get()
1>load() will return a proxy object by assigning the passed identifier	1>It will return an initialized object if the identifier is present else it will return null
2>load() is lazy	2>get() is eager
3>load() can not be used to check the existence of a record.	3>get() can be used to check the existence of a record.
4>load() is faster than get()	4>get() is slower than load().

Example program to Fetch a record from the Employee Table by using get()

Note:Use if condition to avoid NullPointerException

Example program to fetch a record from Employee table by using load()

Handle ObjectNotFoundException

Usage of Proxies in Association Mapping :

---Ex:-One To Many Relationship

One Department can have many Employees

If I try to fetch Department info Using load(Department.class,1)

It will fetch only Department not all the employees

But if I use get(Department.class,1) ---------It will load Along with the department it will Load Employee also

Program to update a record in the Employee Table.

Use if condition in the program

```java
public class FetchUsingGet
{
public static void main(String[] args) {
Configuration conf=new Configuration();
conf.configure();
SessionFactory sef=conf.buildSessionFactory();
Session ses=sef.openSession();
Employee e=ses.get(Employee.class,1);
        System.out.println(e.getId());     //Only to show demo //Later use if
                                                condition
}
```

```
}  output:
Hibernate:
    select
        employee0_.id as id1_0_0_,
        employee0_.name as name2_0_0_,
        employee0_.password as password3_0_0_,
        employee0_.phone as phone4_0_0_
    from
        Empployee employee0_
    where
        employee0_.id=?
1
```

```java
public class FetchUsingLoad
{
public static void main(String[] args) {
Configuration conf=new Configuration();
conf.configure();
SessionFactory sef=conf.buildSessionFactory();
Session ses=sef.openSession();
Employee e=ses.load(Employee.class,1);
System.out.println(e.getId());     //Only to show demo
}

}
```

output:

1

It will not hit the database server so no query is going to generate

But in case of get() Query is going to generate

To Update A Record By Using Hibernate

There are 3 different ways in which we can update a record in Hibernate
1>Fetch and Update
2>update(Object)
3>saveOrUpdate(Object)

1>Fetch and Update

Here you fetch the complete Employee info using get() and update whichever field you want
-Whenever we use get()-→Object will be there in Persistent State.
Any modification on the persistent object will directly affect the record in the table
It is not mandatory that you need to update all the fields.

update(Object)

*It is a method present in Session interface.

*This method will update the record in the database server.

*This method will update the record if the identifier is present or else it will throw an exception**(OptimisticLockException- Root Cause StaleStateException).**

Note:-While using update() if you provide the complete values of a record that's ok if not that particular column will be null or 0 or 0.0

*While updating the record in Hibernate using update(),the object must be there in detached State[StaleState means-----Not Current or Outdated)

saveOrUpdate(object)

*This method is declared in Session interface.
*This method will either save or update a record in the database server.

*This method will save a record if the identifier is not present and it will update the record if the identifier is present.

Note:-If the generator class is **assigned** and if the identifier is **not present** then this method will save() that particular record

Note:-

*Both update() and saveOrUpdate() works in the same manner if ,the identifier is present

*To use saveOrUpdate()

- Generator class must be **assigned.**
- If it is "**identity**" it will not work so, change it to assigned.

Example program to update a record in the database server by using update(Object)

Example program to update a record in the database server by using saveOrUpdate(Object)

delete(Object):-

*This method is declared in Session Interface.

*This method is used to delete a record from the database server.

***First you fetch the object from the database server using get() later you delete**

*Here if you want to delete the record we need to pass the complete object.

*It will throw **IllegleArgumentException** if the argument is not an entity or if the argument is null.

delete(10) ----is not possible because that 10 is not representing any record in the table.You may get **IllegleArgumentException** with a Root Cause **MappingException**

Example Program to delete a record from the table

After Session s=sef.openSession();
s.delete(1);//You will get IllegleArgumentException

All CRUD operations are completed

*The limitations with respect to **get() and load()** are both methods uses primary key to find the record.

*If I ask you to fetch the record based on the name ,password, phone , email then it is not possible

Because none of them are primary key.

*If you want to fetch all the record from the Employee table there is **no inbuilt method**.

*Then the solution is HQL. It is similar to SQL but in **SQL We write table name** but in **HQL we write entity class name**.

*SQL is database dependent where as HQL is database independent.

org.hibernate.query.Query<T>

*It is an interface present in Hibernate Framework.

*This interface is used to execute the HQL queries.

*We can fetch the results from the database server with the help of the methods which are present in this interface which uses HQL statement or Queries .

*The implementation object of this interface can be created with the help of createQuery(String) which is present in Session interface.

Syntax:-

Query<T> q=ses.createQuery(String HQL);

NOTE:-Here type of Query it depends on the result

If the query is returning Employee object then it is Employee

If it is returning integer then it is integer

Following are the important method present in Query interface.

1>getResultList():-

*This method will return a List of results returned by a query object.

* We should call this method if the query is returning more than one result ,the return type of this method is java.util.List.

*The Size of the list will be same as the number of results returned by Query.

*If there is no result List size will be zero

2>getSingleResult():-

*We should call this method if the query returns exactly one result.

*This method will return a Single result returned by Query.

*It will throw NoResultException if the query does not have any result .

*It throws NonUniqueResultException if the query have more than one result.

*The return type of this method is same as the type of the Query.

Hibernate Query Parameters

*These are used to hold the dynamic values from the user during the execution time in an HQL Statement.

*We have 2 types of Parameter

1>Named Parameter(:name)

2>Numbered Parameter(?Integer)

*We must assign the values for the Hibernate parameters.

*To Set the value we have following methods in Query interface

1>setParameter(int position ,object):-

It is used to assign the value for a numbered parameter.

2>setParameter(String ,object):-

It is used to assign the value for named parameter

Code to fetch all the record from the Employee Table

Import the Query interface from org.hibernate.query.Query.

FindAllEmployees

* Session s=new Configuration.configure().buildSessionFactory().openSession();//This is method chaining.

*HQL is case sensitive.(Otherwise it throws exception)

Code to verify an Employee by email and Password

Create an Employee entity class having the following attributes (id,name,phone,email,designation,salary and password) and

Perform the following tasks

1>Save Employee

2>Update Employee

3>Find Employee by id

4>Delete Employee

5>Verify Employee by phone and password

6>Verify Employee by email and password

7>Verify Employee by id and password

8>Find Employees by name

9>Find Employees by Designation

10>Find Employees by salary

11>Find Employees between a salary range

Create the employee entity class having the following attributes (id,name,phone,email,designation,salary and password)

1>Save Employee

2>Update Employee

3>Find Employee by id

4>Delete Employee

5>Verify Employee by phone and password

6>Verify Employee by email and password

7>Verify Employee by id and password

8>Find Employees by name

9>Find Employees by designation

10>Find Employees by salary

11>Find Employees between a salary range.

JPA Annotations

Till now we have seen how to do mapping using mapping file (ie hbm.xml file).
(ie to map our entity class with table in the database)

But Hibernate mapping can be done in 2 different ways

1>By using Hibernate mapping file

2>Using JPA annotation

Previously we were using Hibernate mapping file but here we need to use Annotation.

Note:-

***If you map the entity class with table in the database using hibernate mapping file(hbm.xml file), in Hibernate Configuration File we need to use "mapping resource"**

***If you want to map Entity class with the table in the database server by using JPA Annotation, in Hibernate Configuration File we need to use "mapping class"**

*These are the annotations which are used to map an entity class with a table in the database server .These annotations are present in **javax.persistence** package.

*Following are the important annotations which are used to map an entity class with a table in the database server.

@Entity:-

-It is a class level annotation belongs to JPA and present in **javax.persistence** package.
-This annotation is used to **mark the class as an entity class.This Entity class is going to map as table in the database server because of ORM.**
-The class which is annotated with @Entity will represents a table in the database.

@Table

-It is a class level annotation belongs to JPA and present in **javax.persistence** package.
-Since we are using @Entity annotation ,the annotated class is going to represent a table. To provide the details of the table we use this.

@Id:-

-It is an annotation belongs to JPA and it is present in **javax.persistence** package.

-It is used to mark the field as a primary key .

@GeneratedValue:-

-It is an annotation belongs to JPA and present in javax.persistence package.

-This annotation is used to **mark the field as Generated value**. We can provide the strategy for the generation of primary key by using the enum javax.persistence.GenerationType which belongs to JPA.

-Following are the important GenerationTypes:-

1>GenerationType.IDENTITY

2> GenerationType.SEQUENCE

3> GenerationType.AUTO

4> GeneratiotType.TABLE

@Column:-

-It is an annotation belongs to JPA and present in **javax.persistence** package.

-We use this annotation **just above the field of entity class to provide the information about the column** which is mapped with the annotated filed.

Following are the important attributes of @Column

1>name

2>unique

3>nullable

4>length

5>precision.etc…

Steps to map User class with User table by JPA annotation.

1>Step:1

Create User Entity class

```
package org.jsp.hibernatedemo;
@Entity
@Table(name=”User”)
public class User
{
  @Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private int id;
 @Column (nullable=false)
  private String name;
```

```
@Column (nullable=false,unique=true)
  private long phone;
@Column (nullable=false,unique=true)
  private String email;
 @Column (nullable=false)
private String password;
//getters and Settters//
}
```

Step 2:-

Add the following code in hibernate-configuration file
Inside <session-factory> element

<mapping class="org.jsp.hibernatedemo.User"/>

Note:-Only @Entity and @Id is enough in the entity class to create table

Perform the following tasks by taking input from the User

1>Save User

2>Update User

3>Find User by id

4>Verify user by phone and password

5>Verify User by email and password

6> Verify User by id and password

7>Delete User by id

8>Find User by name

9>Find User by phone

10>Find user by email

11>Fetch phone number of all Users

12>Fetch all users

13>Fetch the email id of all users

Enum:-

It is used to represents the named constants

Instead of using arbitrary numbers or Strings to represent constants enum provides meaningful names.

Enum come with built in methods like values(),name() ,ordinal(),valuesOf()

```
public class Status
{
   public static final int ACTIVE=1;
   public static final int INACTIVE=2;                    ////If you write same thing using class

   public static final int PENDING=3;
}

public enum Status
{
    ACTIVE,INACTIVE,PENDING                               //By using enum
}
```

Enum is used to represents the group of named constants

*we can write abstract methods in enum

Enum can be declared inside the class or outside the class

In Java by using class if you want to represent named constants we may have to write like this

+ class Color

{

    public static final String RED=“red”;

    public static final String BLUE=“blue”;

}

Same thing if you want to write it using enum then

+ enum Color

{

  RED,BLUE;

}

By default enum is final
*enum extends java.lang.Enum class

Enum is the predefined list written by the Programmer
We create Software /Program ,so that End User may use it or Software developer may use it
Ex:---Scanner class,String class some other developer has written but we need to use it in the same manner in our application without any modification
Lets consider being a developer I have written one class

First Scenario

```
class X
{
      void doWork( int a )
      {
          //some code    Being a developer I wish that the user who will use this should pass the int value
      }
}
```

Now user will create the object of this class then he will use this method

```
   X x=new X();
x.doWork(10);//I am restricting the user to pass int value only
```

Second Scenario

```
class X
{
      void doWork( String a )
      {
          //some code    Being a developer I wish that the user who will use this should pass the String value
      }
}
```

Now the user who uses this method should pass the String value only
Like below

```
 X x=new X();
x.doWork("Guru");//I am restricting the user to pass String  value only
```

Lets consider the 3rd Scenario

Now I need to restrict my user to use the predefined list of values which I have defined in List
So what is the solution-----→ie enum

I can not make the method to take String because user may enter any String value (in Second Scenario)
Now define your own enum which is a group of Named Constants
From this list of named constants only user should pick then, we should use enum]

Now in our case JPA has defined some enum ie GenerationType to define the Strategy
Like below

```
enum  GenerationType
{
IDENTITY,
SEQUENCE,
AUTO,
TABLE;
}
```

Now the user who uses the below annotation they should use the same above enum values only
Like below

```
@GeneratedValue(strategy=GenerationType.IDENTITY)
```

Not any other value
So now we can conclude that enum is used to represent the predefiened list of named constants

AUTO :-The persistence provider (Ex:-hibernate) will pick the appropriate strategy (which could be IDENTITY,SEQUENCE or TABLE)

The persistence provider (e.g., Hibernate) selects the appropriate strategy based on the underlying database and any specified configuration. This is often a good default choice if you're not sure which strategy to use.

IDENTITY:-The database automatically generates the primary key value.

SEQUENCE:-Database sequence is used to generate primary key values.

Uses a database sequence object to generate unique identifiers. This strategy is particularly useful for databases that support sequences, such as Oracle and PostgreSQL.

TABLE:-A table is used to generate unique primary key values.

A separate table is used to generate primary key values. This table contains a column that keeps track of the next available ID for each entity type.

Only Problem with Hibernate is Tight-Coupling

We have used Configuration class

SessionFactory

Session

Transaction

Query interface

if my manager ask me to change the orm tool then?

with respect to Hibernate

Then the solution is **JPA**

-It is a specification does not have any implementation.

-Using JPA we can achieve loose coupling which can be implemented by orm tools

-We can use JPA with

- JPA with Hibernate
- JPA with Struts
- JPA with TopLink

-Using JPA we can achieve loose coupling between Java Application and the ORM tool

-JPA Provides specifications that can be implemented by ORM tools using which we can achieve object relational mapping.

-JPA provides a platform using which we can directly deal with the objects instead of writing SQL statements.

-By using JPA we an simplify the interaction between Java application and the database server.

-As JPA is just a specification it can not do anything by itself we need to use the implementations like Hibernate,iBatis, Tapestry,etc… along with JPA to communicate with the database server.

JPA version History:-

1.0----Was introduced in the year 2006

2.0--- in 2009

2.1--- in 2013

2.2---- in 2017

3.1----in 2022

3.2----in 2024

persistence.xml

*It is a JPA configuration file which is used to provide the configuration info to JPA to connect with the database server.

*It must be created in **META-INF** folder which has to be created in src/main/resources

*The root tag of this file is <persistence>
*The sub tag of <persistence> is <persistence-unit>
*The child element of <persistence-unit> is <provider> which is used to specify the implementation provider.
*The child element of <persistence-unit> is <properties>which is used to configure the JPA properties.
*The child element of <properties> is <property> which is used to provide the value for a JPA property.

persistence.xml

```
<persistence……>
    <persistence-unit name=”dev”>
        <provider> org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <properties>
  <property  name=”javax.persistence.jdbc.driver” value=”com.mysql.jdbc.Driver”/>
  <property  name=”javax.persistence.jdbc.url”
value=”jdbc:mysql://localhost:3306/jpa_demo?createDatabaseIfNotExist=true”/>
//If you do any mistake you will get UnKnownDatabase as the message.
  <property  name=”javax.persistence.jdbc.user” value=”root”/>
  <property  name=”javax.persistence.jdbc.password” value=”admin”/>
```

```
//Details with respect to hibernate properties
<property  name="hibernate.dialect"
value="org.hibernate.dialect.MySQL55Dialect"/>
<property  name="hibernate.hbm2ddl.auto" value="update"/>
<property  name="hibernate.show_sql" value="true"/>
<property  name="hibernate.format_sql" value="true"/>
 </properties>
</persistence-unit>
</persistence>
```

Execution:- Class TestCFG

```
{
        main()
            {
EntityManagerFactory fac=Persistnce.createEntityManagerFactory(“dev”);
S.O.P(fac);
            }
}
```

Output:- It should print SessionFactoryImplementation reference

For Execution:-To Test the JPA Configuration

Step:1:- Create a Simple Maven Project
Step:2:-Add the dependencies(Hibernate Core-Relocation and Mysql Connector Java)
Step:3:-Create User.java(Entity Class)
Step:4:-Add the JPA Annotations to the entity class.
Step:5:-Create a JPA Configuration file (persistence.xml) in **META-INF** Folder which has to be created in src/main/resources
Step:6:-Copy the declaration for the persistence.xml from
github link
Step:7:-Write the TestCFG class to test the configuration

To save record using JPA

Create a Class by name User

+class User

{

Id|name|phone|age|email|password

Use JPA annotation in the class

}

*No need of hbm file

*No need of hibernate.cfg.xml

*We have persist() to save the object this is present in EntityManager.

*EntityTransaction contains

---begin() To commit the transaction

---commit()

*All these are present in javax.persistence API

javax.persistence.EntityManagerFactory

-It is an interface belongs to JPA

- It is responsible for creating and managing EntityManager instances, which are used to interact with the database in a JPA application.

-It maintains the Second Level Cache

-It is responsible for managing the lifecycle of EntityManager Instances .It Ensures that resources are properly allocated and released when an EntityManager is created or closed.

Syntax:-

```
EntityManagerFactory fac=Persistence.createEntityManagerFactory(String persistence unit name);
```

createEntityManagerFactory(String persistence unit name)

-It is a static factory/helper method present in Persistence class. Which is used to create and return the implementation class object of EntityManagerFactory.

-Hence the return type of this method is EntityManagerFactory.

javax.persistence.Persistence

- It is a Factory/Helper class belongs to JPA.
- Persistence Class reads configuration details from persistence.xml file which is located in **META-INF** directory.
- The Primary Job of Persistence Class is to create an EntityManagerFactory based on provided persistence unit name.
- The EntityManagerFactory is necessary to create **EntityManager instances**, which are used for all database operations.

javax.persistence.EntityManager

-It is an interface belongs to JPA which is used to manage an entity.

-This interface will provide the methods using which we can save, update, fetch and delete the records without any sql statements.

-EntityManager is a lightweight object that represents a Single Unit of work with the database

-The following are the important methods present in EntityManager.

----- persist(Object)
----- merge(Object)
----- find(Class<T>,Object Primary Key)
----- remove(Object)
---- createQuery(String)
---- createNamedQuery(String)
----- createNativeQuery(String)

Syntax:-

```
EntityManager man=fac.createEntityManager();
```

createEntityManager( ):-

- This method is present in EntityManagerFactory interface.
- It is a non-static factory/Helper method which is used to create and return the implementation class object of EntityManager.
- Hence the return type of this method is EntityManager.

javax.persistence.EntityTransaction

-It is an interface belongs to JPA which is used to manage the transaction on an entity.

-Following are the important methods present in this interface EntityTransaction

---- begin( )

---- commit( )

Syntax:-

EntityTransaction tran=man.getTransaction( );

getTransaction():-

-This Method is present in EntityManager Interface.

- It is a non-static factory/Helper method which is used to create and return the implementation class object of EntityTransaction.

-Hence the return type of this method is EntityTransaction.

Note:-

While writing program if you don't use begin( ) method you will get IllegleStateException with the message:-Transaction Not Successfully Started

persist(Object)

-This method is present in EntityManager interface.

-It is used to persist the entity into the persistence system(Database).

-This method throws IllegleArgumentException if the argument is null or If the argument is not an Entity class.

- The return type of this method is void.

Code to save a Merchant(The Person who sell Products) by using JPA

Create Merchant class (Entity Class) with following fields

- id
- name
- gst_number
- phone
- email
- password

---- Use JPA annotations in Merchant class

---- create persistence.xml(Configuration file of JPA)

---- Create SaveMerchant.java class

- Use persist( ) of EntityManager.
- It throws IllegleArgumentException if the argument is not an entity class

Interview Questions

1>Explain Hibernate and its Advantages

2>Explain Hibernate config file and Hibernate Mapping File

3>What is Session and SessionFactory interfaces.

4>Explain configure() of Configuration Class

5>What are the differences between HQL and SQL

6>Differentiate between get() and load() of Session Interface

7>Explain JPA

8>What is @Entity, @Id,@GeneratedValue,@Column and @Table

9>Name the JPA generators [Identity ,Sequence , Auto, Table]

10>Explain persistence.xml

find(Class<T>,Object primary key)

-It is a method present in EntiryManager interface.

-This method is used to **fetch a record** from the table by using the **primary key**

-This method will search for the specified primary key in a table which is mapped with the specified entity class.

-If the primary key is present it will return the **reference of the object** else it will return **null**

-The **return type** of this method is same as the **type of the first argument** .

If it is Employee ----→It Will be Employee

If it is Merchant -----→It will be Merchant

Note:--If you wrongly enter persistence unit name you will get—**PersistenceException**

-It will throw **IllegleArgumentException** with the root cause **UnKnownEntityException** if the first argument is **not an entity class**

-It will throw **IllegleArgumentException** with the root cause **TypeMisMatchException** if the second argument type is different from the type of Primary key in the Entity class.

-It will throw **IllegleArgumentExceptio**n if the second argument is null.

Code to fetch a record from the Merchant Table by using id

FetchMerchantById.java class

```
Merchant m=manager.find(Merchant.class, id);
```

Use if(m!=null)-----→To avoid NullPointerException

Update a record in the database

There are 2 different ways in which we can update a record in the database.

1>Fetch(using find( )) and Update

2>Use merge( ) of EntityManager interface.

Best way is:- Fetch and update

- Any modification on the persistence object will directly affect the record in the table .

merge():-It will consider the generated id while updating ,if the id is present then it will update else it will save the record.

Code To Update a Merchant by using Fetch and Update way

```
UpdateMerchant
    Merchant m=manager.find(Merchant.class,2);
```

merge(Object)

-It is a method present in EntityManager interface.

-This method is used to merge the state of the passed object with the record present in the database server.

-This method is will **save the record** by assigning the generated id or by using the assigned id if the **primary key is not present.**

-It will update the record if the primary key is present.

-The return type of this method is same as the type of argument.

Code to update a record in the Merchant table using merge(object)

- **UpdateMerchantByMerge.java class**

-Create an Object of Merchant class

- Enter the id value.(m.setId(1))//or//- m.setId(sc.nextInt( ))

- Use merge(m) of EntityManager interface

- Don't use @GeneratedValue //To save the assigned id

Note:--While using merge( ) set all the fields of entity class otherwise record will be updated as 0/0.0/null

*** If the given id is not present and if the Generation Strategy is "identity" then it will not throw any Exception but the record will be saved with identity strategy ,assigned id will not be saved**

remove(Object)

-This method is present in EntityManager interface

-This method is used to remove a persistent object(record)from the database server.

-If we pass a detached or transient object to this method it will throw

IllegleArgumentException.

Code to delete a record from Merchant table by using remove(object)

Find the Merchant by using find(Merchant.class , 1)

```
if(m!=null)
        {
            manager.remove(m);
        }
```

Note:-

As there is no built in methods are available in EntityManager to fetch a record without using id

Now we need to go for **Query** interface.

*In **JPA Query** interface is present in ---**javax.persistence** package

*In **Hibernate Query** interface is present in –**org.hibernate.query** package

*JPA uses -------→JPQL

*Hibernate uses----→HQL

*In Hibenate Query interface **accepts Generics** (Query<T>)

*In JPA Query interface **does not accepts** any Generics (Query)

*The initial versions of JPA were designed before Java introduced generics in JDK 1.5. To maintain compatibility with older versions of Java .

JPA's Query interface was designed without generics.

To support the developers who were using the older version of Java, JPA Query interface was designed without Generics.

javax.persistence.Query

-It is an interface belongs to JPA which is used to **execute JPQL query**.

-The implementation object of this interface can be created by using the following methods which are present in EntityManager.

1>createQuery(String)

2>createNamedQuery(String)

3>createNativeQuery(String)

Syntax:

1>Query q=man.createQuery(String JPQL Query)

2>Query q=man.createNamedQuery(String Name of the JPQL Query)

3>Query q=man.createNativeQuery(String SQL Query)

Following are the important methods present in Query interface.

1>Object getSingleResult():-

-If the **query is returning single** result then we need to use this method to fetch that single result.

-It throws **NoResultException** -- if the query is not returning any result.

-It throws **NonUniqueResultException** – if the query is returning more than one results.

-The **return type** of this method is **java.lang.Object**

2>List<T> getResultList()

-If the **query is returning the list of results**, then to fetch that list of results we need to use this method.

-The **return type** of this method is **java.util.List**

3>setParameter(int,Object)

-It is used to assign the value for a numbered parameter.

4>setParameter(String ,Object)

-It is used to assign the value for a named parameter.

For Type Safety we need use TypedQuery interface like below:-

TypedQuery<Merchant> q=man.createNamedQuery("verifyMerchant",Merchant.**class);**

Task:--

1>Verification of Merchant by phone and password

2>Fetching the Merchant details by Name

3>Fetch all the phone numbers of Merchant.

1.VerifyMerchant.java

```
Query q=manager.createQuery("select m from Merchant m where m.phone=?1 and m.password=?2 ");

    try
        {
            Merchant  m=(Merchant)q.getSingleResult();
        }
Catch(NoResultException)
        {
            S.O.P("Invalid Credentials");
        }
```

2.FindMerchantByName.java

```
Query q=manager.createQuery("select  m  from   Merchant m where m.name=?1");
 List<Merchant> merchants =q.getResultList();
          if(merchants.size()>0)
               {
                  for(Merchant  m  :  merchants)
                       {
                            S.O.P(m);
                        }
               }
        else
              {
                  S.O.P("No Merchant with entered name");
              }
```

3.FindPhoneNumbers .java

```
Query q=manager.createQuery("select m.phone from Merchant m");
List<Long> phs=q.getResultList();
for(Long ph:phs)
    {
      S.O.P(ph);
    }
```

Assignments:

1>Find Merchant by gst_number

```
select m from Merchant m where m.gst_num=?1
```

2>Find Merchant by Phone

```
select m from Merchant m where m.phone=?1
```

3>Verify merchant by id and password

```
select m from Merchant m where m.id=?1 and m.password=?2
```

4>Verify Merchant by email and password

```
select m  from Merchant m where m.email=?1 and m.password=:pw
```

5>Fetch All the email ids

```
select m.email from Merchant m
```

6>Fetch all the names

```
select m.name from Merchant  m
```

7>Fetch all the gst_number

```
select m.gst_number  from Merchant m
```

createQuery(String JPQL)

-It uses JPQL query as a parameter.

-If we want to give the name of the query instead of writing the query directly then use the below ie

createNamedQuery().

createNamedQuery(String Name of JPQL Query)

-It uses name of the JPQL query as the parameter.

-To use the above method you need to **use one annotation** in the **Entity Class** just above the **@Entity annotation** ie

@NamedQuery(name=”name of the JPQL query” , query=”JPQL Query”)

Ex:-1

```
@NamedQuery( name="findAll", query="select m from Merchant m" )
@Entity
public class Merchant
{
}
Query q=manager.createNamedQuery("findAll");
```

Ex:-2

If we have parameters then how to use

```
@NamedQuery(name="Verify Merchant", query="select m from Merchant m where m.phone=?1  and m.password=?2");
Query q=manager.createNamedQuery("Verify Merchant");
q.setParameter(1,phone);
q.setParameter(2,password);
```

Ex:-3

```
@NamedQueries(value={@NamedQuery(name="verifyMerchant" , query="select  m
from Merchant m where m.phone=?1 and m.password=?2"),
        @NamedQuery(name="findAll", query="select m from Merchant m")})
```

@NamedQueries is a collection of @NamedQuery

Points:-

@NamedQuery

1>It is a **class level** annotation belongs to JPA present in **javax.persistence** package.

2>It is **used to create a Named JPQL query** for an entity class.

3>Following are the important **attributes** of this annotation

1>name **2>query**

@NamedQueries

-It is **a class level annotation** belongs to JPA and present in javax.persistence package.

-It is a **container annotation** that is **used to aggregate/group** multiple @NamedQuery annotation in an Entity class .

-Instead of having multiple @NamedQuery annotation scattered throughout the entity class ,@NamedQueries provides a cleaner way to define multiple queries in one Place.

-**value attribute** of this annotation is used to **store** multiple @NamedQuery annotation .

value attribute represents-----→array of @NamedQuery annotation

Example Program to understand createNamedQuery(String)

In Merchant.java

Use

```
@NamedQueries(value={@NamedQuery(name="verifyMerchantByEmail"
,query="select m from Merchant m where m.email=?1"),
@NamedQuery(name="findNames" ,query="select m.name from Merchant m )})
```

Use the above annotation Just above the entity class

VerifyMerchant by Email.java

```
man.createNamedQuery("verifyMerchantByEmail");
```

FindNames.java

```
man. createNamedQuery("findNames");
```

createNativeQuery()

→This method uses SQL query to execute
Here we don't pass JPQL query
-It is database dependent

Example Program to understand createNativeQuery

*Here we use SQL not HQL/JPQL
*Use createNativeQuery(pass sql query , **Result Type**)

FindAllMerchant.java

* In this code if we use createNativeQuery(**"select * from Merchant ")** only ,we will get **ClassCastException** → So,pass the **Result type** also as
*createNativeQuery("select * from Merchant ",**Merchant.class**)

Find Merchant by Id using createNativeQuery()

```
Scanner sc=new Scanner(System.in);
System.out.println("Enter Merchant id ");
int mid=sc.nextInt();

EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
EntityManager man=fac.createEntityManager();
Query q=man.createNativeQuery("select * from Merchant where id=?1 ",Merchant.class);
q.setParameter(1,mid);
List<Merchant> m=q.getResultList();
if(m.size()>0)
  {
for (Merchant merc : m)
      {
        System.out.println(merc);
      }


  }
else
  {
      System.out.println("No Merchant found");
  }
```

Note:-

//In this code if we don't use Merchant.class then we will **get ClassCastException so it is mandatory to use result type**

Note:-**If you don’t use result type in createNativeQuery()**

```
Query q=man.createNativeQuery("select * from Merchant");
 List<Object[]> oarray=q.getResultList();
 for (Object[] obj : oarray)
 {
 int id=(Integer) obj[0];                    //You need to do typecasting
 String email=(String) obj[1];
 String gst_num=(String)obj[2];
 System.out.println("Merchant id "+id);
 System.out.println("Merchant email "+email);
 System.out.println("Metchant GST Number "+gst_num);
  System.out.println("----------------");}
```

If you have 2 Merchant information in the table then output will be like below

OutPut:-

Merchant id- 1
Merchant email- guru@gmail.com
Metchant GST Number- ABCD1234

Merchant id- 2
Merchant email- raj@gmail.com
Metchant GST Number- def1234

Association Mapping

-In case of SQL we were having Emp table and Dept table

-Dept no is used to develop(build) relationship between tables.

-Till now we have not built the relationship between the tables in Hibernate/JPA.

The process of establishing the relationship between the database table is called as association mapping.

Association Mapping refers to how **relationships between entities** (Java objects) are **mapped to database tables.**

The association mapping can be broadly classified into 2 types

1>Uni-Directional Mapping

2>Bi-Directional Mapping

1>Uni-Directional Mapping:-

In Uni-Directional mapping **we can access** the child entity with the help of parent entity and vice versa is not possible.(**Only One entity is aware of RelationShip**)

Ex:-One **Person(Parent Entity)** can have one **Aadhar Card(Child Entity)**

We can fetch the Adhar Card information using Person but vice versa is not possible.

2>Bi-Directional Mapping:-

In Bi-Directional mapping **we can access** the child class entity with the help of parent entity ,the parent entity with the help of Child entity .

We can further classify the association mapping into the following types **based on the number of entities associated with each entity.**

1>one-to-one

2>one-to-many

3>many-to-one

4>many-to-many

1>one-to-one

In one-to-one association mapping **one instance(Object) of an entity class** will be associated **with one instance of another entity class**.

Ex:-one Person can have only one Pancard

One person can have only one AadharCard

2>one-to-many

One instance of an entity class will be associated with **many instance of another entity class.**

Ex:-Department –Employees.

Merchant-Products.

One Department can have many Employees.

One Merchant can sell many Products

3>Many-to-one

Many instance of an entity class will be associated with **one instance of another entity class.**

Ex:-Many answers belongs to one Question

Many Shops in one Mall.

4>Many-to-many

Many instance of an entity class will be associated with **many instance of another entity class.**

Ex:-Student-Batch

A student can attend many Batches.Each batch can have many Students.

OneToOne Uni-Directional Mapping

Person class

-id (This is person id)

-name

-phone(long)

PanCard Class

-id(This is pancard id)

-number(String)

-Dob(LocalDate)//Optional

-pincode(int)

There is no relationship here
So we need to build the relationship.
-Between Person and Pancard we need to build the relationship ie one-to-one uni
-We need to **declare a field** in the **Person class** which is of **PanCard type**
-We need to annotate this field with **@OneToOne** annotation in person class.

Because of this 2 Tables are going to create

Person Table

id(PK)|name|phone|card_id(FK)----→This references PanCard(id----ie PK)

PanCard Table

id(PK)|number|dob|pincode

```
@Entity
public class Person
    {
        id
        name
        @OneToOne
         private PanCard card;
    }
```

```
@Entity
public class PanCard
        {
            id
             number
            dob
            pincode
        }
```

*Simply If we use the below

private PanCard card;(It is the card field in Person class which is of PanCard Type)

Without using **@OneToOne** annotation in Person class, then we will get **PersistenceExcception** with the root cause **MappingException**

Now If we use @OneToOne in Person class Just above the private PanCard card; Like below

```
@OneToOne
private PanCard card;
```

Here 2 Tables are going to create (1st it will create Person Table ,next it will create PanCard Table ,finally it is going to alter Person table to add FK---see in console)

Imp Note:-In OneToOneUni-Directional Association Mapping 2 tables are going to create But one table will have FK (Foreign Key)

Note:-

How many tables will be created in one to one uni ?

2 tables will be created in one-to-one uni-directional Mapping

How many table will have FK ?

1 Table(ie in Person Table)

*FK column(card_id) in the Person table is the combination of card variable of PanCard (ie card) and PK field of PanCard class(ie id)

Don't Forget

-In persistence.xml

Change the db name to onetooneuni

Note:-Date and Time API (java.time.LocalDate)

To check Configuration(Whether the table are created or not)

Create 2 classes

persistence.xml

Don't write any relationship

Check the Configuration

Write TestCfg class

```
{
    EMF
    S.O.P(fac);
}
```

//Check ,is there any column in the Person table referring to PanCard table-→No

//Similarly ,check is there any column in the PanCard Table referring to Person Table-→No

//That means the both the tables are not in relationship

*Ie create Person class and PanCard class with no relationship .

*Later you add this (ie private PanCard card in Person Class) and see card_id(FK column in the Person table)

If you want to build the relationship between two tables first we need to have relationship in classes.

manager.persist(p);//only if you save person object it throws **RollBackException** with the root cause **TransientPropertyValueException**(Reason is - save the transient instance ie Pancard before flushing)

So ,You need to persist PanCard also like below

manager.persist(card);//It is mandatory

@OneToOne

-It is an annotation belongs to JPA and present in javax.persistence package
-This annotation is used to **map one instance** of an entity class with **one instance of** another entity class.
(Or)This annotation is used to map onetoone association in the database.
-We will get exception , if we use this annotation on collection

Example Program to understand oneToOne uni-directional mapping

-Project name:-OneToOneUni

-Person.java(Parent Entity)

```
@OneToOne
    private PanCard card;
```

-PanCard.java(Child Entity)

src/main/resource –META-INF folder---persistence.xml

Write TestCfg class--→Now 2 tables will be created

@GeneratedValue(strategy=GenerationType.IDENTITY)

Note :-If you don't use the above in Pancard class, one more table is going to generate ie hibernate_sequence

next_value

-SavePersonAndPanCard

p.setCard(card);//Assign Pancard to the Person

manager.persist(p);//Persist Person

manger.persist(card);//Without this if I execute we will get TransisentPropertyValueException

Because **Person is in Persistent state** and **PanCard is in Transient State**. So it is mandatory to save child entity also.

Assignment(OneToOne-Uni)

1>Find Person by id

Person p=find(Person.class,1)

Use "if" condition to handle NullPointerException.

Note:- Override toString() in both the classes

2>Find Person by name

Use Query interface --→getResultList()

3>Find Person by phone

4>Find Pancard by id

find(PanCard.class,1)

S.O.P(p);

5>Find Pancard by number

While giving input to the find( ) if you do any mistake you will get InputMisMatchException

6>Find Pancard by number and Date of Birth

7>Find Pancard by Person id

Hint:- Find Person first later PanCard

select p.card from Person p where p.id=?

8>Find Pancard by Person Phone

9>Find Person by Pancard Id // select p from Person p where p.card.id=?1

10>Find Person by Pancard number

11>Find Person by Pancard number and date of birth

Note:-

In the **query if the values are not set using setParameter()**

Then you will get **QueryException** With root cause **Ordinal value not found**

Solving the assignments(OneToOneUni)

6>th Question

Select c from Pancard c where c.number=?1 and c.dob=?2

Ask the user to enter the dob in (yyyy-mm-dd) format

Use createQuery(pass the above query here)

q.setParameter(1,number);

q.setParameter(2,dob);//LocalDate dob=LocalDate.parse(sc.next());

Handle NoResultException by using try and catch
O/P:-PanCard[id=1,number=ABCD5678,…..]

7th Question:-

select p from Person p where p.id=?1

This is used to fetch Person by Person id but for the above requirement

select p.card from Person p where p.id=?1

Advantage:- No need to write join query

8th Question:-

select p.card from Person p where p.phone=?1

9th Question:-

select c from Pancard c where c.id=?1

Here the type of c is Pancard

(Or)

Select p from Person p where p.card.id=?1

This kind of query only applicable for onetoone and many to one

10th Question:-

Select p from Person p where p.card.number=?1

11th Question:-

Select p from Person p where p.card.number=?1 and p.card.dob=?2

Question:-

9,10,11----------→Fetching Person from Pancard

7,8----------→Fetching Pancard from Person

Code to fetch PanCard by Person Phone Number

FindPanCardByPersonPhone class

select p.card from Person p where p.phone=?1

Handle NoResultException by try and catch

Code to find Person by Pancard Number

FindPersonByPanCardNumber class

select p from Person p where p.card.number=?1

Person p=(Person)q.getSingleResult();
Handle NoResultException

Cascade Attribute

Why we need to use Cascade?

In one to one mapping we had called persist() for 2 times.

If we have multiple child entities then it is mandatory to use persist() for multiple times which is not a good practice (Which will leads to Boiler Plate code)

-To avoid this we have one attribute ie cascade

Cascade means that actions performed on a parent entity can be automatically applied to its child entities.

-Whenever we modify the state of Parent entity then the state of child entity has to be modified then, at that time we use cascade attribute .

-Using cascade attribute we can save the storage , Less number of code.

-cascade will save the child entity along with the parent entity.

Points:-

-It is an attribute used in association mapping using which **we can modify the state of child entity** whenever the **state of Parent entity is modified.**

-This attribute is present in the following annotations

1>@OneToOne
2>@OneToMany
3>@ManyToOne
4>@ManyToMany

-We can assign the value for cascade attribute by using the enum javax.persistence.CascadeType which is present in JPA.

See the Demo in OneToOneUni Example :---

Just use @OneToOne(cascade=CascadeType.ALL)

Don't persist Pancard→man.persist(card);-------→This is not required

Following are the different CascadeTypes available

1>CascadeType.PERSIST(To Save)

The child entity will get saved along with the parent entity if we use this CascadeType.

2>CascadeType.MERGE(To Update)

The child entity will get updated along with the parent entity if we use this CascadeType.

3>CascadeType.REMOVE(To Delete)

The child entity will get deleted along with the parent entity , if we use this CascadeType

4> CascadeType.DETACH(To Disconnect)

The child entity will get disconnected from the Session(EntityManager),if we use this CascadeType.

5> CascadeType.REFRESH(To Refresh)

The child entity will get refreshed along with the parent entity if we use this CascadeType

6> CascadeType.ALL(Complete)

CascadeType.ALL ensures that all operations (persist, merge, remove, refresh, detach) on an entity are cascaded to its associated entity.

If we use these CascadeType, then any modification made on the state of Parent entity will also affects the child entity .

One-To-One Bi-Directional Mapping

-In this example we will understand the concept of Owning side(JoinColumn) and Non-Owning Side(mappedBy)

Ex:-User and AadharCard

User

-id

-name

-phone(long)

@OneToOne

private AadharCard card;

AadharCard

-id

-number

-dob(LocalDate)---**card.setDob(LocalDate.*parse("1993-02-17")*)**

-address

@OneToOne

private User user;

*Test the configuration:-

Here 2 tables will be created both the tables will have FK key column

User	AadharCard
Id|name|phone|card_id(FK)	id|number|dob|address|user_id(FK)

Note:-

Here in both the tables FK is present

It is **not required to have FK in both the table** to avoid that we need to use the concept of **owning side** and **Non- owning side** in association mapping

To inform the JPA regarding this

*The table in which you want to **retain FK** is called **Owning side**

*The table from which you want **remove FK** is called **Non-Owning side or inverse-side**

We specify owing side of Association Mapping with ----------@JoinColumn annotation

We specify non-owning side of Association Mapping with ---------"mappedBy" attribute

@JoinColumn is an annotation used in Hibernate to **specify** the column that will be used to join two tables in a database.

-It is used to represent the column in the database that acts as the foreign key.

*Owning side and Non-Owning side is only applicable for Bi-Directional Mapping

NOTE:-

-There is no concept of **mappedBy attribute** in **ManyToOne** Association Mapping

-There is **no rule to decide owning and non-owning side** it totally depends on the requirement.

Points:-

@JoinColumn:-

*It is an annotation belongs to JPA and present in javax.persistence package

*We choose one Entity class to manage the relationship we call this as owning side.

*It is used to provide the details of foreign key in association mapping .It can also be used to specify the owning side of association in bi-directional mapping.

*It must used with the annotations which are used to achieve association mapping.

mappedBy:-

-It is an attribute used in association mapping to specify the non-owning side of **association in bi-directional mapping .**

-It indicates that the field it is associated with is not the owner of the relationship, but instead, the relationship is managed by the other entity. This helps Hibernate understand which entity is responsible for managing the relationship

Points:-

mappedBy attribute is not applicable for ManyToOne Bi-directional

Because,ManyToOne Bi-Directional mapping is also known as OneToMany Bi

While doing coding if I use OneToManyBi-----One Extra Table will be created to build relationship

In ManyToOne Bi----→One extra column will be created to build a relationship

So It is good to select ManyToOne

But where do we use @ManyToOne there only I need to use @JoinColumn so there will not be any attribute in this mapping

But if you observe mappedBy attribute will come in OneToManyBi

mappedBy attribute is there in

@OneToOne

@OneToMany

@ManyToMany

Example Program to understand OneToOne Bi-Directional mapping

Change database in persistence.xml

User.java
AadharCard.java
SaveUserAndAadhar.java

Note:-

***Initially 2 tables will have FK's(which is not necessary) so avoid that extra FK using below**

Here 2 tables will be created but only one FK column will be there because of owning side and non-owning side.

@OneToOne(mappedBy="card")
private User user;

*This tells us that Aadharcard is already mapped in user table

Imp :-For Ex:-

```
@OneToOne(mappedBy = “card")
    private User user;
```

This tells “card” field in the “User” class is responsible to build relationship.

- The **mappedBy** attribute in Hibernate annotation indicates that the relationship is already established by another enitity Class(Ie User)

Note:-

- In @JoinColumn(name=“aadharcard_id”)
- name attribute is optional
- Without using **mappedBy** **in AadharCard class** if you use only @JoinColumn in **User Class** Just if you check the configuration still you will not get any error or exception but extra FK column will not be removed
- So it is mandatory to use **mappedBy**

Assignment Questions:-

1>Find User by id

Note:-If you override toString() in both the classes then you will get StackOverFlow Error in case of Bi-Directional Mapping if you use the references

So while overriding don’t include references

2>Find User by name

3>Find User by phone

4>Find User by Aadhar id

```
select u from User u where u.card.id=?1
```

5>Find User by Aadhar number

6>Find User by Aadhar Number and date of birth

7>Find Aadhar by id

```
find(Aadharcard.class,1)
```

8>Find Aadhar by number

9>Find Aadhar by number and date of birth

10>Find Aadhar by User id

```
select a from AadharCard a where a.u.id=?1
```

11>Find Aadhar by user phone

12>Find Aadhar by user name and phone

OneToMany Uni-Directional

Ex:- One Department can have many employees

Department Class

id |name|location

Employee Class **(Employee Class is unaware of Relation that's why it is uni)**

id| name |degn

Note:-

EF codd rule(One cell must always contains only one value)

Primary key can not be duplicated.

→In oneToMany uni directional mapping we will get 3 tables but the 3rd table contains the PK of 1st table and PK of 2nd Table

*3rd Table is called as Join Table(Department _Employee)

*Third Table Columns :------Department_id|emps_id

*We can not avoid the creation of 3rd table in OneToMany Uni

Here 3rd table is used to establish a relationship between other two tables.
@OneToMany is always used on collection .If you use it on the field which is not a collection then it throws AnnotationException

Note:-

In case of OneToOne --------→We will get extra column
ManyToOne to build relationship

In case of OneToMany
ManyToMany--------→We will get extra table
to build relationship

Points:-

@OneToMany

-It is an annotation belongs to JPA and it is present in **javax.persistence.package**

-This annotation is used to map **one instance of an entity class** with **many instance of another entity class.**

-We will get AnnotationException, if we use this annotation on a filed which is not a collection(ie the second entity must be collection type)

Note:-

Exclude the references while overriding the toString() to avoid -→**StackOverFlowError**

-d.setEmps(Arrays.asList(e1,e2,e3));

asList()-----takes variable length argument

asList() converts elements into List and it returns List

Ex:--

print(int a)----------→It takes one parameter

print(int a ,int b)---→It takes 2 parameters

but

print(int...a)-----→takes unlimited parameter

To Show the number of insert statement

Remove format_sql from persistence.xml

Example Program to understand oneToMany Uni-Directional Mapping

Department.java

id|name|location

@OneToMany(cascade=CascadeType.ALL)//**Without this annotation if we execute we will get PersistenceException with the root cause MappingExcepton**

private List<Employee> emps;

Employee.java

id|name|desg|sal

Note:-Show Demo for 3 tables

SaveDeptAndEmps.java

```
Department d=new Department();
Employee e1=new Emplopee();
                e2
                e3
d.setEmps(Arrays.asList(e1,e2,e3));
```

(Or)

```
List<Emplyee> emps=new ArrayList();
              emps.add(e1);
              emps.add(e2);
              emps.add(e3);
              d.setEmps(emps);
```

Code To Fetch Employees by Dept id

```
Select d.emps from Department d where d.id=?1
List<Employee> emps=q.getResultList();
```

```
if(emps.size()>0)
        {
      for(Employee emp:emps)
            {
            S.O.P(emp);
            }
        }
else
        {
        }
```

Assignment(OneToMany Uni-Directional Mapping)

1>Find Department by id

find(Department.class,id)

2>Find Department by name→Use Query interface

3>Find Department by location→UseQuery Interface

4>Find Employee by id

find(Employee,id)

5>Find Employees by name

Select e from Employee e where e.name=?1

6>Find Employees by designation

7>Find Employees by salary

8>Find Employees by department id

select d.emps from Department d where d.id=?1

9>Find Employees by dept id and name

10>Find Employees by dept id and location

In case of

OneToOne

ManyToOne----------→ Extra Column is going to create

OneToMany

ManyToMany----------→Extra Table is going to create

@ManyToOneUni

Ex:-One Question can have many answers-

(Or)

Many answers belongs to one question:-It is a good choice since it creates one column

Let’s see onetomany first

Ie One Question can have many answers

Because of OneToMany relation, 3 Tables are going to create

questiondata,answerdata,qestiondata_answerdata like below

questiondata

id|postedBy|question

answerdata

id|answeredBy|answer

questiondata_answerdata

QuestionData_id|answers_id

Which is unnecessary

We can build a relationship between the tables by using ManyToOne also

Ie

Many answers belongs to one question:-It is a good choice since it creates one column

In this case **one extra column will be created** to maintain relationship like below

questiondata	answerdata
id|postedBy|question	id|answeredBy|answer|question_id

In oneToMany --------→3rd table is created to build a relation

In ManyToOne--→one column is created to build a relation

Ex:-

QuestionData.java

(Child Entity)

-id

-postedBy

-question

AnswerData.java

(Parent Entity)

-id

-answeredBy

-answer

@ManyToOne(cascade=CascadeType.ALL)

private QuestionData question

-It is ManyToOne Uni since there is no **AnswerData** variable in **QuestionData** Class --

-QuestionData is unaware of Relationship

Points :

@ManyToOne

*It is an annotation belongs to jpa and present in javax.persistence package

*This annotation is used to map **many instances of one entity class** with **one instance of another entity class**

*We will get annotationException if we use it on Collection

Example Program to understand ManyToOne uni-directional mapping

SaveQuestionAndData.java

```
QuestionData q=new QusetionData();
```

Note:-

While Writing the JPQL Query if you do any spelling mistake you will get **QuerySyntaxExeption**

```
AnswerData a1=new AnswerData();
        a1.setQuestion(q);
AnswerData a2=new AnswerData();
        a2.setQuestion(q)
AnswerData a3=new AnswerData();
        a3.setQuestion(q)
    manager.persist(a1);
    manager.persist(a2);
    manager.persist(a3);
      tran.commit();
```

Tables:-

questiondata

id|postedBy|question

answerdata

id|answer|answeredBy|question_id

Assignment Question(ManyToOneUni)

1>Find Questiondata by ID

find(QuestionData.class,id)

2>Find QuestionData by questioned by attribute

3>Find QuestionData by question attribute

```
Query q=man.createQuery("select ques from QuestionData ques where ques.question=?1");
q.setParameter(1,que);
try {
QuestionData qe=(QuestionData) q.getSingleResult();
System.out.println(qe);
} catch (NoResultException e)
{
System.out.println("No Data found with respect to this question");
}
```

Here use nextLine () to take input (Question)

4>Find QuestionData by AnswerData id

```
Query q=man.createQuery("select a.question from AnswerData a where a.id=?1");
q.setParameter(1,aid);
try {
QuestionData qs= (QuestionData) q.getSingleResult();
System.out.println(qs);
} catch (NoResultException e)
{
System.out.println("No AnswerData Found For for id");
}
```

5>Find AnswerData by id

find(AnswerData.class,id)

6>Find AnswerData by answer attribute

sc.nextLine()----→To Take answer.

Query q=man.createQuery("select a from AnswerData a where a.answer=?1");

AnswerData ad=(AnswerData) q.getSingleResult();

Handle NoResultException

Output:

AnswerData [id=3, answeredBy=Sunil, answer=It is a ORM tool, question=QuestionData [id=1, postedBy=Guru, question=What is Hibernate?]]

Note:-

I am fetching AnswerData by answer attribute but why it is fetching question also

So we need to use fetch atttribute

7>Find AnswerData by QuestionData id

For One question id ---→we will get List of AnswerData so use getResultList()

```
Query q=man.createQuery("select  a from AnswerData a where a.question.id=?1");
        q.setParameter(1, qid);
        List<AnswerData> lans=q.getResultList();
if(lans.size()>0)
    {
for (AnswerData  answ :  lans)
          {
              System.out.println(answ);


          }
    }
else
   {
            System.out.println("No AnswerData found for the given id");
    }
```

8>Find AnswerData by question attribute

9>Find AnswerData by questionedBy attribute

OneToManyBi-Directional Mapping

(Or) ManyToOneBi-Directional Mapping

-In this case we need to mention the annotation in both the classes

-@OneToMany annotation in one class and **@ManyToOne** in another class

Ex:-One Merchant would sell many Products (OneToMany) **(or)**
Many Products belongs to One Merchant(ManyToOne)

Merchant

-id
-name
-gst_number
-phone
-password

@OneToMany(mappedBy,cascade)

private List<Product>products

Product

-id
-name
-brand
-category
-description

@ManyToOne
@JoinColumn

private Merchant merchant

Here in this case we will get 3 tables (Whenever we establish a relationship)

1>Merchant

id|name|gst_number|phone|password

2>Product

id|name|brand|category|description|merchant_id(FK)(this we will get because of manytoone)

3>merchant_product(We will get this table because of onetomany)

Merchant_id(PK)|products_id(PK)

Already merchant_id column is there in the Product table

No need of 3rd table

How to avoid the creation of 3rd Table? :

-ie with the help of @JoinColumn and mappedBy attribute

- So, the class which contains @OneToMany is responsible to generate 3rd table will be---------- non-owning side(use mappedBy Here)----ie in Merchant class

-The class which contains @ManyToOne is responsible to generate column will be ----------owning side(use @JoinColumn here)----ie in Product class

Example Program to understand OneToMany Bi-Directional Mapping

Merchant.java

Product.java

SaveMerchantAndProduct

```
        Merchant m=new Merchant();
        Product p1=new Product();
p1.setMerchant(m);//Assign merchant to the product
        Product p2=new Product();
p2.setMerchant(m);
```

```
//Assign all the products to one merchant
        m.setProducts(Arrays.asList(p1,p2));
//Save merchant automatically child entity is going to be saved
        manager.persist(m);
```

Note:-Eventhough Product class(Parent Entity)owns association since it is bi-directional mapping

Both the classes are aware of relationship so

persist merchant that is enough

What is the use of owning side and non-owning side ?

Using this we can avoid the creation of multiple FK's we use this

Assignment(OneToMany Bi or ManyToOne Bi)

1>Find Merchant by id

find(Merchant.class,id)

2>Find Merchant by name

select m from Merchant m where m.name=?1

3>Find Merchant by gst_number

select m from merchant m where m.gst_number=?1

4>Verify merchant by phone and password

select m from Merchant m where m.phone=?1 and m.password=?2

5>Verify Merchant by id and password

select m from Merchant m where m.id=?1 and m.password=?2

6>Find Merchant by Product id

```
Query q=man.createQuery("select p.merchant from Product p where p.id=?1");
q.setParameter(1,pid);
try {
Merchant m=(Merchant) q.getSingleResult();
System.out.println(m);
} catch (NoResultException e) {
            System.out.println("No Merchant found since the Product id is
wrong");
}
```

Note:-In the above example //Don't use reference of both the classes *while overriding toString() else you will get StackOverFlowError*

7>Find Product by id

find(Product.class,id);

8>Find Products by name

select p from Product p where p.name=?1

9>Find Products by brand

select p from Product p where p.brand=?1

10>Find Products by category

select p from Product p where p.category=?1

11>Filter Products between a price range

select p from Product p where p.price between p.min=?1 and p.max=?2

12>Find Products by merchant id

select p from Product p where p.m.id=?1

```
Query q=man.createQuery("select m.products from Merchant m where m.id=?1");
q.setParameter(1,mid);
List<Product> lps=q.getResultList();
if(lps.size()>0)
    {
        for (Product product : lps) {
                System.out.println(product);

            }
    }
else
    {
        System.out.println("No Products Found since the merchant id is wrong");
    }
```

13>Find Products by merchant phone and password

select m.products from Merchant m where m.phone=?1 and m.password=?2

14>Find Products by Merchant id and password

```
select  m.products from Merchant m where m.id=?1 and m.password=?2
```

15>Find Products by merchant gst_number

```
select m.products  from Merchant m where  m.gst_num=?1
```

ManyToManyUni-Directional Mapping

Ex:-**Many Batches are attended by many Students**

Batch	Student
-Id	-id
-batch_code	-name
-trainer	-phone
-subject	-perc

```
@ManyToMany(cascade=CascadeType.All)
private List<Student>students;
```

3 Tables are going to create like below

batch

id|subject|batch_code|trainer

student

id|name|perc|phone

batch_student

Batch_id|students_id

*We can not avoid the craeation of 3rd Table since it is required to build relationship between batch and student table

Note:-

We can not avoid the creation of Extra table(3rd Table) in ManyToManyUni

batch_student

Batch_id	Student_id
1	1
1	2
1	3
2	1
2	2
2	3

Note:-

In oneToOneUni----------→2 Tables----→1FK

(Owning Side and Non-Owning side is not required)

In OneToOneBi----------→2 Tables---→2FK's

(To avoid the FK –we need owning side and non owning side)

In OneToManyUni----------→3 Tables,3rd Table is required

In ManyToOne----------→2 Tables-----→1FK

In OneToManyBi----------→3 Tables---→We can avoid the creation of 3rd table we retain one column using owning and Non-owning side

In ManyToManyUni----→3 Tables-→3rd Table is required to build relationship

Points:-

@ManyToMany

-It is an annotation belongs to JPA and present in javax.persistence package.

-It is used to map **many instances of an entity class** with the **many instances of another entity class**

-We will get exception if we use it on a field which is not a collection.

Example Program to understand ManyToMany Uni-Directional Mapping

Here 17 insert statements
To Show Demo Remove format_sql in persistence.xml

Batch.java(Parent Entity)
-id
-batch_code
-subject,trainer
-@ManyToMany(cascade)
private List<Student>students;

Student.java

-id

-name

-phone

-perc

SaveBatchesAndStudents.java

```
Batch b1=new Batch();
b1.setSubject("Java");
b1.setBatch_code("JAVAB1");
b1.setTrainer("Abhishek");

Batch b2=new Batch();
b2.setSubject("J2EE");
b2.setBatch_code("J2EEB2");
b2.setTrainer("Guru");
```

```
Student s1=new Student();
s1.setName("Guru");
s1.setPerc(78.40);
s1.setPhone(9483663883l);

Student s2=new Student();
s2.setName("Raj");
s2.setPerc(80.00);
s2.setPhone(9482205408l);

Student s3=new Student();
s3.setName("Rahul");
s3.setPerc(35.35);
s3.setPhone(9483663889l);
//Since it is uni
b1.setStudents(Arrays.asList(s1,s2));
b2.setStudents(Arrays.asList(s1,s2,s3));
        man.persist(b1);
        man.persist(b2);
       tran.commit();
```

ManyToManyUni (Assignment Questions)

1>Find batch by id

```
find(Batch.class,id)
```

2>Find Batch by subject

```
select b from Batch b where b.subject=?1
```

3>Find Batch By batch_code

```
select b from Batch b where b.batch_code=?1
```

4>Find Batch by trainer name

```
select b from Batch b where b.batch_code=?1
```

5>Find Student by id

```
find(Student.class,1)
```

6>Find Students by name

```
select s from Student s where s.name=?1
```

7>Find students whose percentage is >=60

```
select s from Student s where s.perc>=60
```

8>Filter Students between a Percentage range.

```
select s from Student s where s.perc between ?1 and ?2
```

9>Find Students by batch id

```
select b.students from Batch b where b.id=?1
```

10>Find Students by batch code

```
select b.students from Batch b where b.batch_code=?1
```

ManyToManyBi-Directional Mapping

Batch

@ManyToMany

private List<Student>students

Student

@ManyToMany

private List<Batch>batches

Totally 4 tables are a going to create ----------2 Extra tables

batch

id|subject|batch_code|trainer

student

id|name|perc|phone

batch_student(Use @JoinTable to retain this)

Batch_id|students_id

student_batch

Student_id|batches_id

-Remove one table and retain one table(Any one table is fine)

-Here we are retaining the table so use @JoinTable in association mapping .

joinColumns and inverseJoinColmns are the attributes of @JoinTable.

-joinColumns ---→ it is an attribute of @JoinTable--------→Type is @JoinColumn[]

(It is used to provide the details of column which is referring owning side)

→inverseJoinColumn:-Which is used to provide the details of the column which is refererring to non-owning side

→name is the attribute which is used to change the table name of owning side.

Points:-

@JoinTable

-It is an annotation belongs to JPA and present in javax.persistence package.
-It is used to represents the owning side of association in bi-directional mapping.
-It is used to provide the details of JoinTable(The table which you want to retain) in association mapping .

-Following are the important attributes present in @JoinTable

1>**name:-**It is used to specify the table name

2>**joinColumns:-**It is used to provide the details of the column in the join table which is referring to the owning side of association.

3>**inverseJoinColumns:-**It is used to provide the details of the column in the join table which is referring to the non-owning side of association .

Example Program to understand ManyToManyBi-Directional mapping.

-If you don’t use owing side and non----------owning side 4 tables will be created.

batch

id|batch_code|subject|trainer

student

id|name|perc|phone

batch_student

Batch_id|students_id

student_batch

Student_id|batches_id

Two tables are not required to build relationship so delete one table by using @JoinTable annotation and mappedBy attribute

@JoinTable in ----------→Batch class

mappedBy attribute in------→Student class

Batch.java

-id

-batch_code

-subject

-trainer

@ManyToMany(cascade)

@JoinTable(name="batch_student",joinColumns={@JoinColumn(name="batch_id")},
inverseJoinColumns={@JoinColumn(name="student_id")})
private List<Student>students;

Note:-inverseJoinColumns--→is used to specify the non-owning side component in the table.

Student.java

-id

-name

-phone

-perc

```
@ManyToMany(mappedBy=”students”)
private List<Batch>batches;
```

SaveBatchAndStudents.java

SAME as ManyToMany

```
s1.setBatches(Arrays.asList(b1,b2,b3));
s2.setBatches(Arrays.asList(b1,b2,b4));
s3.setBatches(Arrays.asList(b1,b2,b3,b4));
```

In Batch Class

```
@ManyToMany(cascade=CascadeType.ALL)
@JoinTable(joinColumns= {@JoinColumn},          (or)          only @JoinTable only also fine
        inverseJoinColumns= {@JoinColumn})
private List<Student>students;
```

In Student Class

```
@ManyToMany(mappedBy="students")
private List<Batch> batches;
```

Enough to remove 4th table which is not necessary Now 3rd table will be

batch_student

Bataches_id|students_id

Note:--Only to provide the info of 3rd table which we are going to retain we are using joinColumns and inverseJinColmns attributes that's it where I can remove like above only using @JoinTable

-Both joinColumns and inverseJoinColumns are used to represent the FK's in third table.

One is used to represent the FK of owning side.

One more is used to represent the FK column of non-owning side.

```java
@Entity
public class Batch
{
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private int id;
private String batch_code;
private String trainer;
private String subject;

@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "Batches_Students",
joinColumns = {@JoinColumn(name="Batches_id")},
inverseJoinColumns  = {@JoinColumn(name="Students_id")})
private List<Student>students;
//Getters and Setters
//toString()

}
```

```java
@Entity
public class Student
{
@Id
@GeneratedValue(strategy =GenerationType.IDENTITY)
private int id;
private String name;
private long phone;
private double perc;
@ManyToMany(mappedBy = "students")
private List<Batch>batches;

//Getters and Setters
//override toString()

}
```

```
import java.util.Arrays;
import javax.persistence.*;
public class SaveBatchesAndStudents
{
	public static void main(String[] args)
	{
		EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
		EntityManager man=fac.createEntityManager();
		EntityTransaction tran=man.getTransaction();
		tran.begin();

		Batch b1=new Batch();
		b1.setSubject("Java");
		b1.setTrainer("Abhishek");
		b1.setBatch_code("JavaB1");
```

```
Batch b2=new Batch();
b2.setSubject("J2ee");
b2.setTrainer("Guru");
b2.setBatch_code("J2eeB2");

Student s1=new Student();
s1.setName("Guru");
s1.setPhone(9483663883l);
s1.setPerc(63.45);

Student s2=new Student();
s2.setName("Ram");
s2.setPhone(9483663882l);
s2.setPerc(45.12);

Student s3=new Student();
s3.setName("Rahim");
s3.setPhone(9483663886l);
s3.setPerc(35.45);
```

```
//Assign Students to the batches
b1.setStudents(Arrays.asList(s1,s2,s3));
b2.setStudents(Arrays.asList(s1,s2));

//Assign batches to the Students
s1.setBatches(Arrays.asList(b1,b2));
s2.setBatches(Arrays.asList(b1,b2));
s3.setBatches(Arrays.asList(b1));

man.persist(b1);
man.persist(b2);

tran.commit();



    }

}
```

Output:-

Total 3 tables will be there

batch_student

batch_id	student_id
1	1
1	2
1	3
2	1
2	2
2	3

Assignment Question on :-ManyToManyBi-Directional Mapping

1>Find Batch by Batch id

find(Batch.class,1)

2>Find Batch by Batch_Code

select b from Batch b where b.batch_code=?1

3>Find Batch by Subject

select b from Batch b where b.subject=?1

4.Find Batch by trainer name

select b fromBatch b where b.trainer=?1

5>Find Batches by Student id

select s.batches from Student s where s.id=?1//s.batches means list of batches

select b from Batch b where b.students.id=?1///This is wrong //Since b.students refers to List of Students

Not one student

6>Find Batches by Student id and name

select s.batches from Student s where s.id=?1 and s.name=?2//s.batches means list of batches

select b from Batch b where b.students.id=?1 and b.students.name=?2//This is wrong

7>Find Batches by Student phone

select s.batches from Student s where s.phone=?1//s.batches means list of batches

select b from Batch b where b.students.phone=?1//This is wrong

8>Find Batches by student phone and name

select s.batches from Student s where s.phone=?1 and s.name=?2//s.batches means list of batches

select b from Batch b where b.phone=?1 and b.name=?2//This is wrong

9>Find Student by id

find(Student.class,1)

10>Find Students by name

select s from Student s where s.name=?1

11>Find Student by phone

select s from Student s where s.phone=?1

12>Find Students whose percentage is greater than 35

select s from Student s where s.percentage>35

13>Display the student details whose percentage is more than 60

14>Filter the students between a Percentage range.

select s fro Student s where s.perc between ?1 and ?2

15>Display the student details whose percentage is greater than 85

select s from Student s where s.percentage> 85

16>Find Students by Batch id

select b.students from Batch b where b.id=?1//b.students----→Refers to List of Students

select s from Student s where s.batches.id=?1//It is wrong //s.batches means it is list of batches not one batch

17>Find Students by batch code

select b.students from Batch b where b.batch_code=?1//b.students is List of Students

select s from Student s where s.batches.batch_code=?1//It is Wrong

18>Find Students by subject and batch_code

select b.students from Batch b where b.subject=?1 and b.batch_code=?2

select s from Student s where s.batches.subject=?1 and s.batches.batch_code=?2//It is wrong

9>Find students by trainer and batch_code.

select b.students from Batch b where b.trainer=?1 and b.batch_code=?2

select s from Student s where s.batches.trainer=?1 and s.batches.batch_code=?2//It is wrong

Fetch Attribute

Why its required?

Whenever we load the parent entity whether the child entity has to be loaded or not?

When to use cascade?

Whenever we modify the state of parent entity then the state of child entity has to be modified then we use cascade.

FetchType.Eager:-

If we fetch the parent entity it will fetch child entity also.

FetchType.Lazy:-

If we use this it will fetch parent entity but not child entity (It will fetch or child entity will be loaded on demand)

Note:-

Person and PanCard

In oneToOne association mapping since the default fetchType is eager

Because of which if we try to fetch Person, along with Person PanCard will be loaded.

Imp Note:-

Even though we are not using fetch attribute in association mapping don't you think that some default fetch types are present ?

Yes

Because while in onetoone association mapping even though if I fetch Person, the Pancard information is going to load

In ManyToOne the default fetch type is eager.

Ex:-Many Products belongs to one Merchant

FindProductById.java

```
Product p=manager.find(Product.class,1);
```

If we try to fetch product it will load Merchant also

In ManyToMany-----The default fetch type is lazy

Ex:-Batches and Students

FindBatchById.java

```
Batch b=manager.find(Batch.class,1);//To fetch Parent
S.O.P(b.getStudents());//Since the default fetch type is lazy we need to use this ,on demand only we will get the child entity.
```

Points:-

It is an attribute used in association mapping using which we can decide whether the child entity has to be loaded along with the parent entity or not .

This attribute is present in below annotations

1>@OnToOne

2>@OneToMany

3>@ManyToOne

4>@ManyToMany

-We can assign the values for **fetch attribute** by using the enum javax.persistence.FetchType.

-Following are the two different types of FetchType available

1>FetchType.EAGER:-

Here the child entity gets loaded along with the parent entity if we use this FetchType.

2>FetchType.LAZY:-

Here the child entity will not be loaded along with the parent entity .If we use this FetchType

-But the child entity is going to load on demand

Ex:--Student and Trainer Assignment

Whenever trainer asks for assignment to do some students by default they complete the assignment we can consider them as eager students to do the assignment

In the same way some students are lazy once after some punishment they complete their assignment that's why they are lazy.

These are some default FetchTypes:-

Annotation	Default FetchType
OneToOne	Eager
OneToMany	Lazy
ManyToOne	Eager
ManyToMany	Lazy

Ex:-@OneToOne Uni

User.java

@OneToOne(cascade=CascadeType.ALL,fetch=FetchType.EAGER)

FindUserById.java

User u=manager.find(User.class,1);

//It will fetch User and Pancard Details if I use the above line since the default fetch type is eager

S.O.P(u.getUser());//If we use FetchType as lazy then this line is required.In this case only user will be loaded

Here in this case

If we use the FetchType as lazy

S.O.P(u);//It will fetch only User info not aadhar card info

S.O.P(u.getAadhar());//Now it will load aadhar card info also

In @OneToMany

In case of oneToMany the default fetch Type is lazy so,

Merchant.java

@OneToMany(cascade=CascadeType.ALL,mappedBy=,fetch=

FetchType.LAZY)

Since it is lazy only Merchant is going to load

If you use EAGER then both Merchant and Products are going to load.

FindMerchantById.java:-

Merchant m=manager.find(Merchant.class,1);

//If I use the above only Merchant will be loaded

Since the FetchType is lazy//only Merchant is going to load Product will be loaded on demand like below

If you want the product info here then use

S.O.P(m.getProducts());

@ManyToOne

In ManyToOne the default fetchType is Eager

Product.java

```
@ManyToOne(fetch=FetchType.LAZY)
```

FindProductById.java

```
Product p=manager.find(Product.class,1)
```

@ManyToMany

In ManyToMany the default fetch type is lazy

Batch.java

```
@ManyToMany(cascade=CascadeType.ALL,fetch=FetchType.EAGER)//Default fetch type is lazy
 private List<student>students;
```

FindBatchById.java

```
Batch b=manager.find(Batch.class,1);
S.O.P(b.getStudents());
```

Example Program to understand fetch attribute in OneToOne Uni

Here we have person class and PanCard class

In Case of oneToOne Relationship the default fetch type is Eager

So whenever we try to find Person without using any “fetch” attribute, along with the Person, PanCard will be loaded in the console like below

FindPersonById.java

```
Person p=manager.find(Person.class, 1);

if(p!=null)
{
System.out.println(p);
}
else
{
System.out.println("No Person found");
}
```

OutPut in Console Window:-

```
Hibernate:
    select
        person0_.id as id1_1_0_,
        person0_.card_id as card_id4_1_0_,
        person0_.name as name2_1_0_,
        person0_.phone as phone3_1_0_,
        pancard1_.id as id1_0_1_,
        pancard1_.number as number2_0_1_,
        pancard1_.pincode as pincode3_0_1_
    from
        Person person0_
    left outer join
        PanCard pancard1_
            on person0_.card_id=pancard1_.id
    where
        person0_.id=?
Person [id=1, name=Raj, phone=9482025408]
```

Now add fetch attribute in Person Class like below

@OneToOne(cascade = CascadeType.***ALL,fetch=FetchType.LAZY)***
private PanCard card; //then only Person information will be loaded like below

After adding fetchtype Output in Console window

```
Hibernate:
    select
        person0_.id as id1_1_0_,
        person0_.card_id as card_id4_1_0_,
        person0_.name as name2_1_0_,
        person0_.phone as phone3_1_0_
    from
        Person person0_
    where
        person0_.id=?
Person [id=1, name=Raj, phone=9482025408]
```

Now if you want PanCard information then on demand PanCard will be loaded like below

```
Person p=manager.find(Person.class, 1);

if(p!=null)
{
System.out.println(p.getCard());
}
else
{
System.out.println("No Person found");
}
```

```
Hibernate:
    select
        person0_.id as id1_1_0_,
        person0_.card_id as card_id4_1_0_,
        person0_.name as name2_1_0_,
        person0_.phone as phone3_1_0_
    from
        Person person0_
    where
        person0_.id=?          //Same thing is applied to oneToOneBi(No Changes)
Hibernate:
    select
        pancard0_.id as id1_0_0_,
        pancard0_.number as number2_0_0_,
        pancard0_.pincode as pincode3_0_0_
    from
        PanCard pancard0_
    where
        pancard0_.id=?
PanCard [id=1, number=Raj1234, pincode=577419]
```

In Case of OneToMany Uni/Bi

EX:-One Department can have many Employees

In this oneToMany Uni/Bi the default fetch type is Lazy.

Here if you try to fetch Department using id, since it is lazy only Department class will be loaded Employee class will not be loaded.

So ,in Department class if you make the fetch type as Eager then
Even though you are trying to fetch Department using department id, along with it All Employees info are going to load.

In case of ManyToOne Uni/Bi

Ex:-Many Answers will have one question

Here AnswerData is the Parent Entity

The default fetch type is Eager------so If you if you try to fetch AnswerData by using answer id it will load QuestionData information also

So,in AnwerData Class if you make the fetch type as Lazy Explicitely then
It will not load QuestionData.

In Many To Many Uni

Ex-Student and Batch

One Student can attend many Batches .Each Batch will have many Students

Here the default fetch type is -→Lazy

If you try to fetch Batch by Batch id ---→ Student will not be loaded.

Now If you add the fetch type as eager in the Batch Class, because of this fetch type along with Batch even Student will be loaded

TimeStamp Annotations

What is TimeStamp?

TimeStamp is a datatype used to represent(store) date and time

It is commonly used track the event or operation which are performed based on date and time .

Ex:--FlipKart App or Swiggy

Whenever we order food User will not provide the below information .Application is responsible to provide this Like

- Order id (To generate this Generation Strategy is there)-----It Can not be changed
- Ordered time(Odered time and Delivery time has to generated by the application)---It can not be changed

  ordertime remains the same
- Delivery time(ETA---Estimated Time of Arrival)---This depends on some parameters like (Distance + Time Taken To Prepare food)----It can be changed

If I place an order the application should assign the Current virtual machine date and time and it should assign the value to a field(ie ordered time)

*The value will be assigned for Delivery time whenever we update the oder.(ie multiple time)

*LocalDateTime is the Class present in java.time package.

-It represents a date and time without time zone component.

*It is immutable class and thread safe.

LocaDateTime instances are created by factory or helper methods Including

now()----It is a static method which is used to retrieve current date and time

Of()---It is a static method which is used to create the instance date and time classes by specifying specific values for year,month,day,hour,minute,second and nano second.

Comparison with Other Types:

- **LocalDateTime vs LocalDate**: LocalDateTime includes time components (hour, minute, second), while LocalDate represents a date without time.

- **LocalDateTime vs ZonedDateTime**: ZonedDateTime includes a time zone, making it suitable for handling dates and times across different time zones.

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Example {
    public static void main(String[] args) {
        // Current date and time
        LocalDateTime now = LocalDateTime.now();
        System.out.println("Current LocalDateTime: " + now);

        // Create a specific LocalDateTime
        LocalDateTime specificDateTime = LocalDateTime.of(2024, 7, 15, 12, 30);
        System.out.println("Specific LocalDateTime: " + specificDateTime);

        // Formatting LocalDateTime
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String formattedDateTime = now.format(formatter);
        System.out.println("Formatted LocalDateTime: " + formattedDateTime);
    }
}
```

Whenever we update the data(Distance) there is possibility of changing the delivery time.

But ordered time will not change.

FoodOrder.java(Don’t give the name as Order because Order is the keyword in SQL)

You will get Exception

@Table you can use to give different name

-id (it is order id)

-food_item

-cost

@creationTimeStamp(Both the annotations are from Hibernate not from JPA)

-private LocalDateTime ordered_time;//The value will be assigned only once.

@UpdateTimeStamp

-private LocalDateTime delivery_time;//value will be assigned every time whenever you update the order.

Points:-

-These are the annotations which are used in Hibernate to generate the date and time to assign the value for annotated field.

-These annotations are present in org.hibernate.annoatations package.

-Following are the important TimeStamp annotations.

1>@CreationTimeStamp:-

2>@UpdateTimeStamp:-

1>@CreationTimeStamp:-

-This annotation will assign the annotated filed within the entity class with the current virtual machine date and time whenever we save the entity for the first time .

Ie the field in the entity class which is annotated with @CreationTimeStamp will hold the current virtual machine date and time whenever we try to persist the entity into database.

*The time and date which is inserted into the database server will not be updated or modified afterwards. Eventhough if you update the other fileds of entity .

2>@UpdateTimeStamp:-

This annotation will assign the annotated filed with the current virtual machine date and time when we save or update the entity.

Ie the field which is annotated with @UpdateTimeStamp in the entity class can be updateable later.

(ie When we place an order and when we update the distance by default delivery time will be updated)

Note:-

LocalDateTime:- It is not supported by MySQL 5.5 --→You will get SQLSyntaxException.

If so change the MySQL 5Dialect to MySQL8Dialect in persistence.xml

Note:-If we update the oder(distance,food) ,then the delivery time should change but not ordered time.

To update the order use UpdateOder.java class

To update first you fetch---→Using find() you update it from Biryani to Chicken Biryani

Example Program to understand @CreationTimeStamp and UpdateTimeStamp

@Entity

FoodOrder.java //Hibernate WorkSpace//ZomatoApp

-id

-food_item

-cost

@CreationTimeStamp(Ordered time will not change)

-private LocalDateTime ordered_time;

@UpdateTimeStamp(delivery time would update)

-private LocalDateTime delivery_time;

//Getters and Setters//

PlaceOder.java

main()

EMF(If you give persistence unit name wrongly you will get PersistenceException

EM

ET

FoodOrder order=new FoodOrder();

order.setFood_item(“Biryani”);

order.setCost(150);

manager.persist(order);

UpdateOrder.java

```
  main()
      EMF
      EM
      ET
//Before you update first you fetch
FoodOrder oder=manager.find(FoodOrder.class,1));
//Update the food_item
  Order.setFood_item(“Panner Biryani”);
transacation.begin();
transaction.commit()
```

Assignment Questions:-

1>Find Food order by id

find(FoodOrder.class,1)

2>Find FoodOrders by food item

select f from FoodOrder f where f.food=?1

Use getResultList();

3>Find FoodOder by id and food

select f from FoodOder f where f.id=?1 and f.food=?2

Use getSingleResult()

4>Find Food Orders by cost

select f from FoodOrder f where f.cost=?1

5>Filter food orders between a range of cost.

select f from FoodOder f where f.cost between ?1 and ?2

```
import java.time.LocalDateTime;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

@Entity
public class FoodOrder
{
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int id;
        private String food_item;
        private int cost;

        private LocalDateTime deliverytime;

        @CreationTimestamp
        private LocalDateTime  ordered_time;

        @UpdateTimestamp
        private LocalDateTime OrderUpdateTime;
```

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

EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
EntityManager man=fac.createEntityManager();
EntityTransaction tran=man.getTransaction();
   tran.begin();

   FoodOrder od=new FoodOrder();
   od.setFood_item("Pulav");
   od.setCost(50);
//To get the current system date and time
   LocalDateTime now = LocalDateTime.now();
//add 20 minutes to the current time that will be the delivery time
   LocalDateTime deliverytime=now.plusMinutes(20);

   od.setDeliverytime(deliverytime);
   man.persist(od);
   tran.commit();
}
}
```

```
public class UpdateOrder {
public static void main(String[] args)
{
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter order id to update");
    int oid=sc.nextInt();

    EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
    EntityManager man=fac.createEntityManager();
    EntityTransaction tran=man.getTransaction();
    tran.begin();
    FoodOrder od=man.find(FoodOrder.class, oid);
     if(od!=null)
         {
     od.setFood_item("RiceBath");
     od.setCost(185);//Not Mandatory
     LocalDateTime updateddeliverytime=LocalDateTime.now().plusMinutes(20);
//It is mandatory to update delivery time whenever you update the food
     od.setDeliverytime(updateddeliverytime);//od.setDeliverytime(LocalDateTime.now().plusMinutes(20));

     tran.commit();
         }
     else
        {
     System.out.println("Order Can not be updated since order id is wrong");
        }
```

Composite Key

The PK is used to identify the record uniquely.

The combination of two or more columns to form a PK is known as Composite Key

Composition means combining more than one thing

In a table it is always recommended that only one PK must be there

It is a combination of two or more columns (fields) that together uniquely identify a record in a table.

Each combination of values in the composite key must be unique within the table.

This means no two rows can have the same combination of values across all columns that make up the composite key.

In Primary key -----column means single column but

Composite Key------ Column means multiple columns

When we can use this?

Composite keys are often used when no single column can uniquely identify a record, but the combination of several columns can.

Take One Example where one column is not enough to identity the record uniquely in the table

Lets consider One Training Institution -----BTM Jspiders-----→One DataBase as BTM
In BTM Databse Lets consider 3 tables

Student Table

sid	sname
1	Guru
2	Raj

Course Table

cid	cname
101	J2EE
102	Java

Master Table

sid	sname	cid	cname
1	Guru	101	J2EE
2	Raj	102	Java
1	Guru	102	Java
2	Raj	101	J2EE

Here the combination of sid and cid forms composite key

If you want to save the data in Master Table then you Should create an entity class (Master) like bleow
Now Our Master Class is Composite key class .
It is a rule that Composite Key class must implement Serializable but it violets the rule of POJO

@Entity
class Master implements Serializable
@Id
-sid
@Id
-cid
-sname
-cname Output

```
create table Master (
       cid integer not null,
        sid integer not null,
        cname varchar(255),
        sname varchar(255),
        primary key (cid, sid)
    ) engine=MyISAM
```

It is not recommended to have 2 PK's in a Entity class for Below Reasons

- **Normalization**: Having two separate primary keys implies that each one independently identifies a unique record, which can violate normalization principles. Proper normalization usually involves ensuring that a table has a single, unique primary key.

- **Referential Integrity**: Foreign key constraints and other relational database features are designed to work with a single primary key. Having multiple primary keys would complicate these relationships and potentially lead to data integrity issues.

```
import java.io.Serializable;
import javax.persistence.Embeddable;

@Embeddable
public class MasterId implements Serializable
{
        private int sid;   //Embeddable Class which includes the  all the fields  which are responsible for composite Key
        private int cid;  //No need of @Id

     //Setters nad Getters
    //Override toString()
}
```

Note:-If you don't implement this MasterId class with Serializable then you will get PersistenceException with a root cause MappingException

```
Message:-Composite-id class must implement Serializable
```

Note:-Hibernate WorkSpace/CompositeKeyFinal

```
@Entity          //Don't Forget this
public class MasterStudentInfo
{
private String sname;
private String cname;
@EmbeddedId                 //In this line it shows error ignore it //It will force you
private MasterId mid;       //to override equals() and hashcode()//Not Mandatory
//Above Declare the variable of MasterId
//      Setters and Getters  //
//       Override toString()  //

}
```

Composite key instances are often used in collections such as HashSet or as keys in HashMap . These collections rely on equals() and hashCode() to determine object equality and to manage object storage efficiently.

```
public class SaveMasterStudentInfo
{
public static void main(String[] args) {
EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
EntityManager man=fac.createEntityManager();
EntityTransaction tran=man.getTransaction();
tran.begin();
MasterId mi=new MasterId();
mi.setSid(2);
mi.setCid(102);

MasterStudentInfo msi=new MasterStudentInfo();
msi.setSname("Raj");
msi.setCname("Java");
msi.setMid(mi);//
man.persist(msi);
tran.commit();
```

```java
public class FetchStudentInfo {
public static void main(String[] args) {
EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
EntityManager man=fac.createEntityManager();

MasterId mid=new MasterId();//Composite Key Class
//Find the Student info based on Composite Key
mid.setSid(1);
mid.setCid(101);

MasterStudentInfo msinfo=man.find(MasterStudentInfo.class, mid);
if(msinfo!=null)
{
System.out.println(msinfo);

}
else
{
System.out.println("either student id or course id is wrong");
}
```

Composite Key

If you want to give the PK column by combining more than one column then we use Composite Key concept

Ex:-Lets consider User table

Id|name|phone

-Here if I make the id column as PK column then duplicates will not be allowed here

-If I make the phone column as PK column again duplicates will not be allowed here.

Lets consider User table

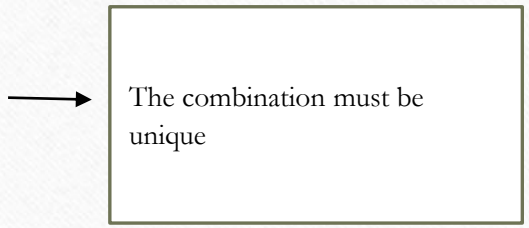
Name | Phone|Email--→Composite Key

In case of Composite Key

The combination of more than one column must be unique in each record .

User

Name	Phone	Email
Xyz	888	xyz@gmail.com
Xyz	888	abc@gmail.com



Note:-

(Composite key class must implement Serializable)

Why the composite key class should implement Serializable?

By implementing Serializable, you ensure that instances of **CompositeKey** can be serialized and used in scenarios where serialization is required, enhancing the flexibility and compatibility of your Hibernate-based application.

Implementing Serializable allows instances of the composite key class to be serialized, which is essential for various use cases, such as caching, distributed computing, or passing objects between different layers of an application.

```
@Entity
public class User implements Serializable
   @Id
      private int id;          //No this class contains 2 PK's
   @Id
      private String phone;
      private String name;
      private String password;
```

There are 2 different way in which we can create composite key

Note:-

1st way:-

We need to make the class which includes the composite key as Serializable

The first way would violets the rule of POJO since the pojo class must not implement any interface.

But here it is implementing Serializable interface

2nd way:-

Write one class(UserId), there have all the attribute of user class which are responsible to create composite key(ie email and phone)

Now the UserId class will be embedded class it should implements Serilaizable.

Embedded class means class is fixed or Embedded in another class

Points:--

- The combination of more than one column to form a PK(Primary Key) is called as Composite Key.

Note:-

- If we combine more than one column to form a composite key the combination of those columns must be unique in every record of the table.

- Lets consider user table having the columns email,phone,name and password where the the combination of email and phone number are considered as Primary key(Composite key)

The below table represents the working of Composite Key

Composite Key

Email	phone	name	password
abc@gmail.com	888	ABC	abc123
abc@gmail.com	888	XYZ	xyz123
abc@gmail.com	777	XYZ	abc123
pqr@gmail.com	777	ABC	xyz123
pqr@gmail.com	777	XYZ	abc123

These two combination should not repeat

Points:

@Embeddable

-It is a class level annotation belongs to JPA and present in javax.pesistence package.

@Embeddable annotation is used to indicate that a class can be embedded(fit) within an entity

-It is used to mark the class as Embeddable class.

-An Embeddable class will have all the fields which are responsible for the creation of Composite Key.

-An embeddable class must implements java.io.Serializable interface.

@EmbeddedId

-It is an annotation belongs to JPA and present in javax.persistence package.

-This annotation is used to mark the filed as Embedded id(Composite Key)

-The field which is annotated with @Embeddable must be of Serializable type else we will get exception.

Demo:-

Class User implements Serializable

-name

@Id

-phone

@Id

-email

-password

If I don’t use serializable

we will get MappingException

Don’t use generated value

-If I implement we will violet the rule of POJO

+class Test

EMF

S.O.P(factory);

Example program to understand @Embeddable and @EmbeddedId

@Embedddable
+ class UserId implements Serializable

-phone
-email

] these two attributes of User class IS contributing to the composite key so write here not in user

@Override
toString()

@Entity

User.java

-name

-password

@EmbeddedId

private UserID userid;

//To support POJO rule in user class use this

//Getters and Setters

//toString()//

SaveUser.java

```
 main()
     UserId userid=new UserId();
                userid.setEmail("xyz@gmail.com");
                 userid.setPhone(9999);
 User user=new User();
user.setName("ABC");
user.setPassword("abc1234");
user.setUserId(userId);
EMF
EM
ET
```

```
manager.persist(user);
tran.begin();
FetchUserByPrimaryKey.java
  main()
    UserId userid=new UserId();
    userid.setEmail("abc@gmail.com");
userid.setPhone(888);
EMF
EM
User user=manager.find(User.class,userid);
If(user!=null)
    S.O.P(user);
```

else

S.O.P(Invalid phone number or email)

Output:-

User[name=ABC,password=ABCD1234,userId[phone=9999,email=xyz@gmail.com]

Hibernate LifeCycle

How Human beings are having our own life cycle .Butterfly is having its own life cycle

Is there any lifecycle for Hibernate objects means --->yes

Previouly I told you about transient State,Persistent State,etc still some more states are there

What is the difference between each state and the task performed by an object in each state is different

In Hibernate we have saved the record ,we have up

-Whenever we create object by using new keyword then the object will be there in transient state.

-Whenever it is connected with session or EntityManager then it will enter into Persistent state.

-When the object goes from Transient State to Persistent State means ?

If we call save()or update() or saveOrUpdate() or persist() or merge()

-In transient State object will not represents any record in the table any modification will not affect the record

-The object in persistent state will represents a record in the table

We were using find() or get() to fetch the record -->That is Persistent State

-If the object is connected to the session then it is said to be Persistent state.

- If the object is disconnected from the session –it will go to the Detached state.
- If you close EntityManager or Session it goes to detached State or if you call detach() the object in the persistent state will go to detached state.
- -Is there any possibility to move the object which is there in Detached State to Persistent State?---yes by calling find()

Note:-

Once we close the program or Execution of the Program Completes

Garbage collector will remove all the object from the heap ie it removes the implementation class object of Entity Manager from heap that means EntityManager is going to close.

Now the record is there in the database but it is not connected with Session

-

How To move the object from detached State to Persistent state ?For that we can use below methods

get()

load() or find()

-If we change the values of Transient object it will not affect the record in the table.

When do we call an object is in detached State?

When object is disconnected from the Session.Bou

We have one More State of Hibernate Object ie RemovedState

When the object goes to removed State?

Whenever we delete an object it goes to removed State. To delete a Persistent object

we have we can use delete() or remove()

delete() can delete both Persistent object and Detached Object

but remove() can delete Only delete persistent object //

So if you try to delete using detached object using remove() -→you will get IllegleArgumentException

So What are the States of an object in Hibernate LifeCycle

-Whenever we use delete() or remove() then the object will go to Removed State

-In Detached State object will be there in Database but not connected with Session

-We can not delete the object which is there in detached State because it is not connected with session.

-find() is used to move the object from detached state to persistent state.

ST:-

-Hibernate LifeCycle is used to depict(Represent) the different state of an object in Hibernate.

-Following are state of an object in Hibernate LifeCycle.

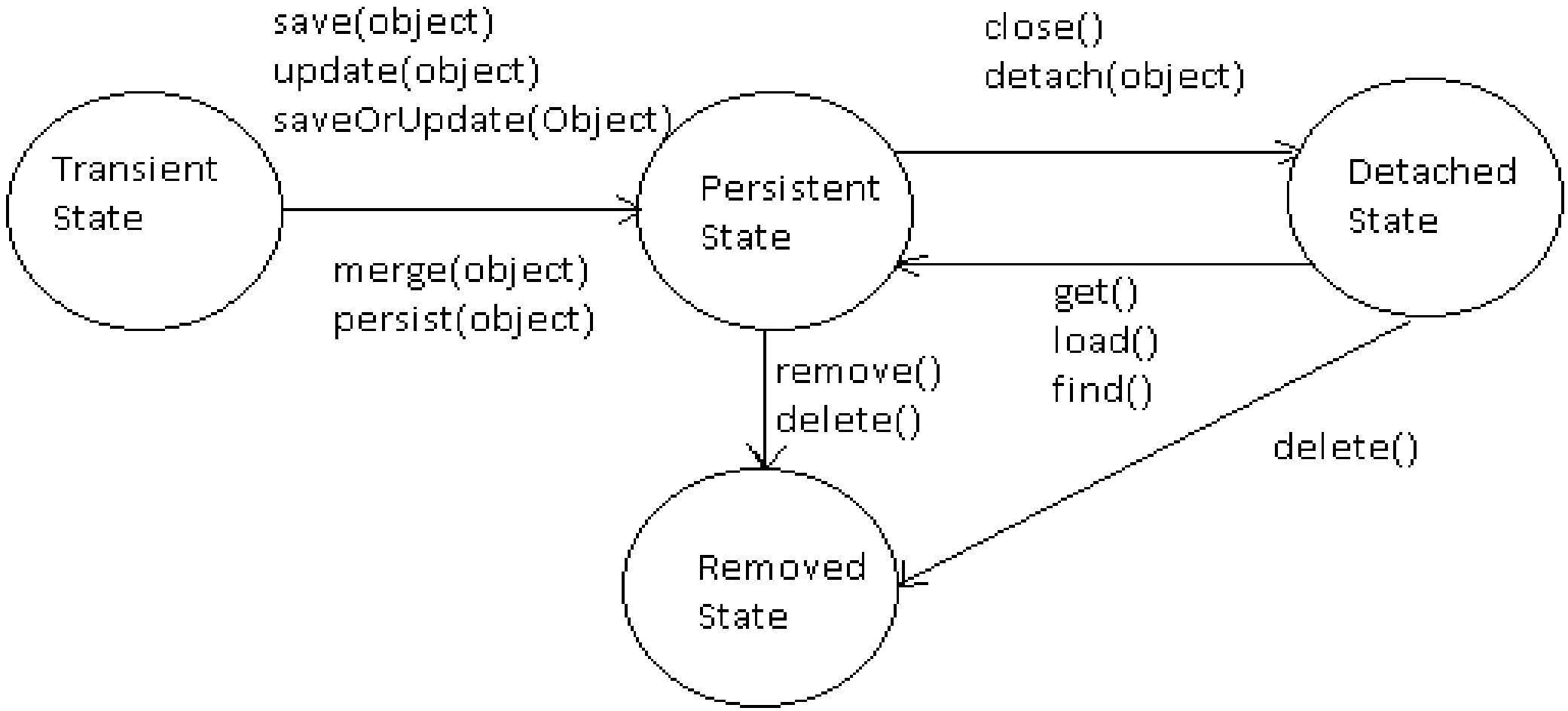
1>Transient State

2>Persistent State

3>Detached State

4>Removed State

Hibernate LifeCycle



1>Transient State

-An object is said to be transient state when it is newly created

-An object in the transient state will not represent any record in the table ,modification on the transient object will not have any effect on the record in the table.

2>Persistent State

-An object is said to be in persistent state once it is connected with the Session.

-An object in the persistent state will represents a record in the database server.

-The modification on the state of a persistent object will change the record it is representing.

3>Detached State:-

-An object is said to be in detached state once it is disconnected from the session(Entity Manager).

-An object in the detached state will not represent any record in the database server.

So modification of detached object will not affect any record in the database server.

4>Removed State:-

-An object is said to be in removed state once it is deleted from the database server.

-We can not delete an object in the detached State by remove(object)

-remove(object) can only delete persistent object.

HibernateLifeCycle_Proj

Example Program to understand Hibernate LifeCycle

Lets Track the Hibernate LifeCycle

Test HLC

-EMF

-EM

ET

Person p=new Person();//Transient State

p.setAge(25);

p.setName(“ABC”);

manager.persist(p);//Persistent State//It is connected with Session//

DeletePerson

-EMF

-EM

-ET

```
Person p=manager.find(Person.class,1);
if(p!=null)
     {
       manager.remove(p);
       transaction.begin();
       transaction.commit();

    }
}
```

Note:-

-Here if 1 is not present then find() will return null .Now p will hold null

-If I directly use it will be

manager.remove(p);//

//Now it will throw IllegleArgumentException

When you will get NullPointerException?

p.setAge(25);

p is null so any operation on null reference throws an exception called NullPointerException.

So use if(p!=null)

9:23 Pending

Cache Mechanism

- Cache Mechanism is use to improve the performnace of an application.

-Lets consider there is no concept of Cache.

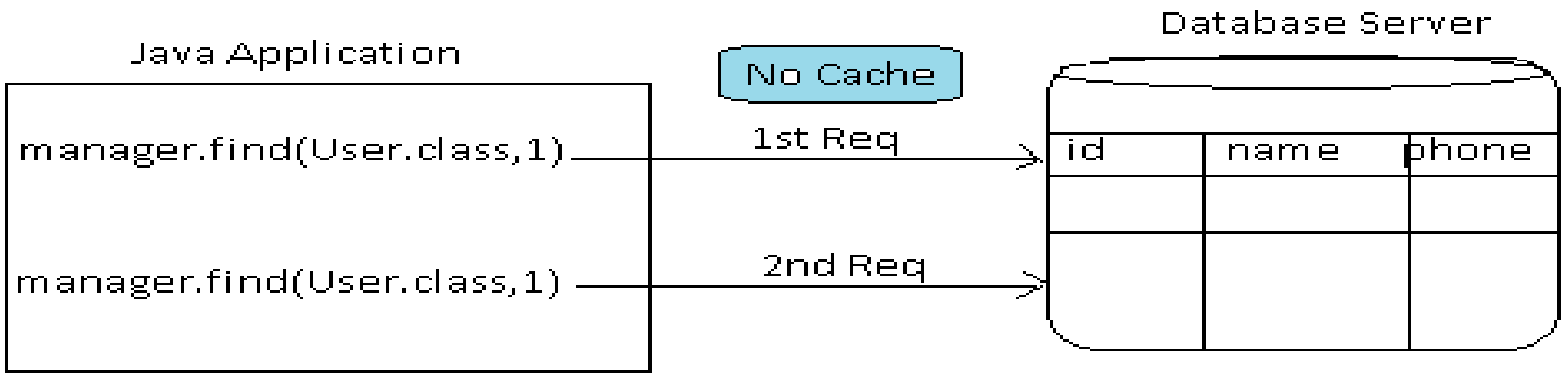
-If I want to fetch the data of an user from the database server where id=1 then SQL query has to be excecuted.

If you don’t use the traffic increases between java application and the database server.

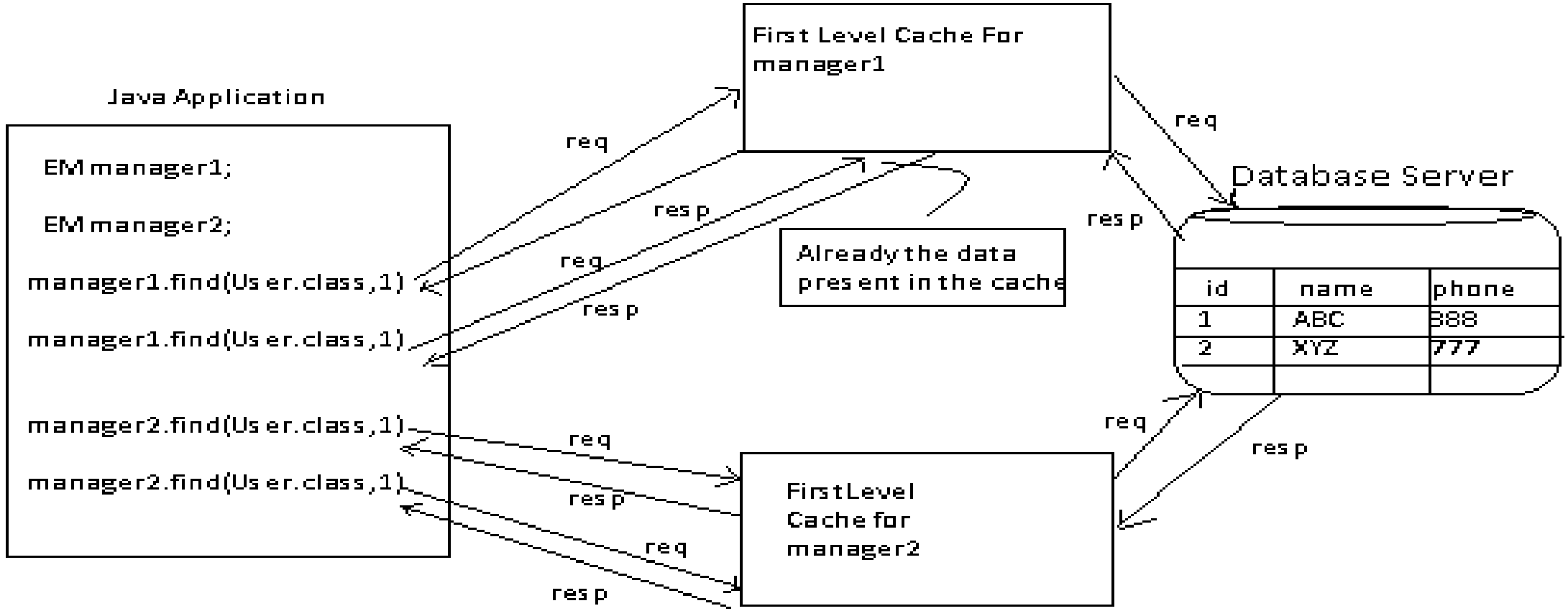
*By default hibernate supports First Level Cache.

*2nd Level cache is not enabled

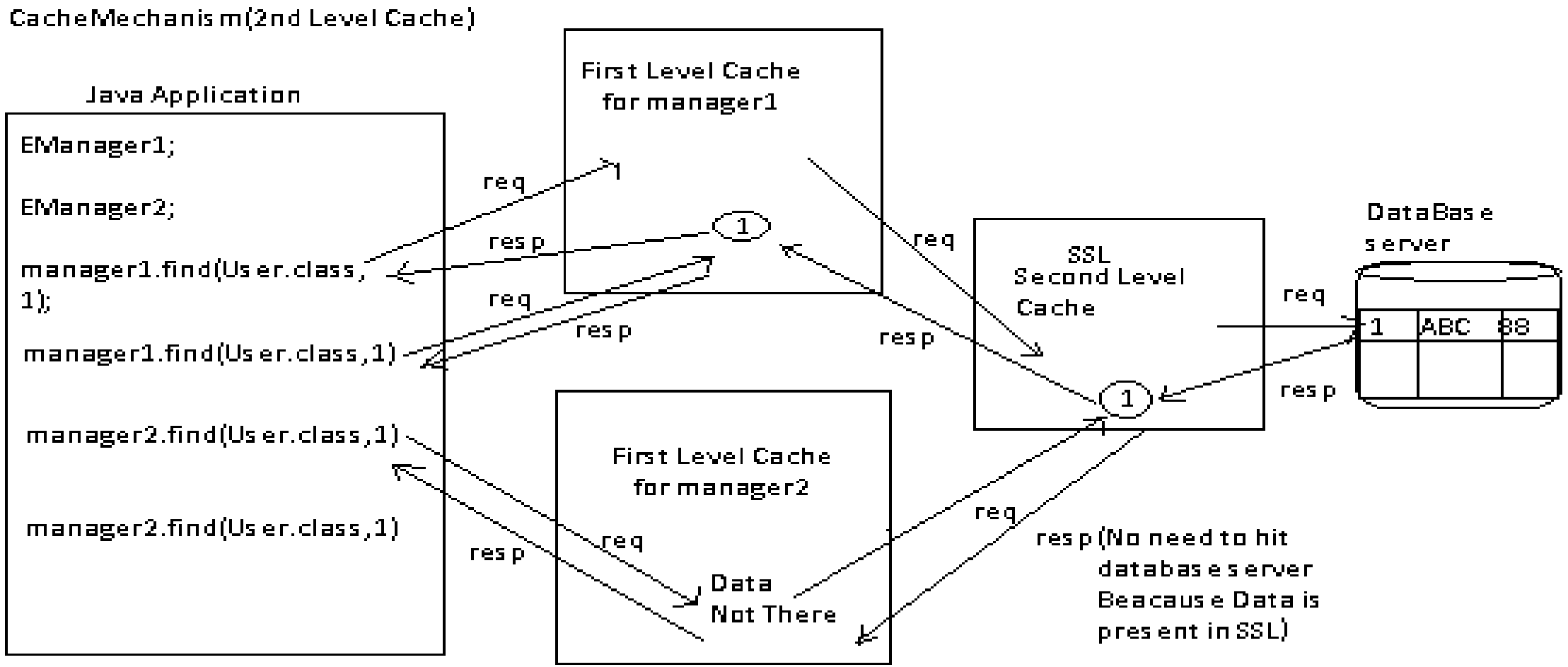
*Hibernate allocates 1st level cache for each Session.


Java Application
manager.find(User.class,1)
manager.find(User.class,1)
No Cache
1st Req
2nd Req
Database Server
id
name
phone

The use of 1st level cache



Usage of SecondLevel Cache



For 2 EntityManager ----2First Level Cache Will be provided by Hibernate.

Note:-

To enable the 2nd Level cache ---→we need to add one dependency

i e EHCache Relocation in pom.xml

-The Version of EHCache relocation and Hibernate Core-Relocation must be same

-Just go to persistence.xml.txt in GitHub(Sathish)

Copy <shared-cache-mode> tag and paste it in persistence.xml of your program inside <persistence-unit>

Then add <property name=”hibernate.cache.use”>

<property factory class>

Annotate your entity class with @Cacheable

Points:-

Cache:-

-It is a temporary layer of storage which is used to store the data to serve future request.

-Hibernate Supports 2 levels of cache mechanism using which we can reduce the traffic between the Java application and the database server and increases the performance.

-Following are the 2 Levels of Cache Supported by Hibernate

1>First Level Cache

-It is a layer of storage assigned by the hibernate for every session(EntityManger) to store the data to serve the future request.

-Every Session will have a dedicated first level cache memory and all of them will be connected with Second Level Cache.

2>Second Level Cache

- It is a layer of storage which is shared by all the sessions(Entity Managers) created by a SessionFactory(EntityMangerFactory).

- By default only First Level cache is supported by Hibernate and the 2nd level cache has to be enabled explicitely.

The property hibernate.cache.region.factory_class in your Hibernate configuration specifies the class that Hibernate should use to manage the second-level cache regions. The value org.hibernate.cache.ehcache.EhCacheRegionFactory indicates tha

Steps to enable 2nd Level Cache

Step1:-Add the hibernate EHCacheRelocation dependency in pom.xml

Step2:-Add the following code in <persistence-unit/>

Just above <properties> tag Below the <provider>

<shared-cache-mode>Enable_SELECTIVE</shared-cache-mode>

Note:<shared-cache-mode>Enable_SELECTIVE</shared-cache-mode>, it means that the second-level cache is enabled only for those entities and collections that are explicitly marked as cacheable

Step 3:-Add the following properties in persistence.xml

<proprty name=”hibernate.cache.use_second_level_Cache” value=”true”/>

<property name=”hibernate.cache.region.factory_class” (To Store)
value=”org.hibernate.cache.ehcache.EhCacheRegionFactory”/>

-The property hibernate.cache.region.factory_class in your Hibernate configuration specifies the

class that Hibernate should use to manage the second-level cache regions.The value org.hibernate.cache.ehcache.EhCacheRegionFactory indicates that Hibernate should use EHCache as a Caching Provider.

Step 4:-Annotate your entity class with @Cacheable

Program To Understand Cache-Mechanism

@Cacheable

User

-id

-name

-phone

id	name	phone
1	Guru	9483663883
3	Raj	9482205408

//Setters and Getters

//Override toString()

```java
public class FindUser1
{
public static void main(String[] args)
{
EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
EntityManager man1=fac.createEntityManager();//one 1st Level cache for man1
EntityManager man2=fac.createEntityManager();//one 1st Level cache for man2 //Toatally 2 First –Level caches will be given


//Without using Second Level Cache if you execute you totally 4 times it hits the database server
//2-1st Level caches are going to create for man1 and man2 here
//Here 2 --1st level caches will be there but not 2nd level cache
man1.find(User.class, 1);//It Hits the database server
man1.find(User.class, 2);//It Hits the database server                    // Without Using 2nd Level Cache
                                                                          //It will hit the database server 4 times
man2.find(User.class, 1);//It hits the database server
man2.find(User.class, 2);//It hits the database server

//Since the data is already present in 2-1st level caches again it will not hit the database server.

man1.find(User.class, 1);//It will not hit the database server
man1.find(User.class, 2);//It will not hit the database server

man2.find(User.class, 1);//It will not hit the database server
man2.find(User.class, 2);//It will not hit the database server

}

}
```

```java
public class FindUser2
{

public static void main(String[] args)
{
EntityManagerFactory fac=Persistence.createEntityManagerFactory("dev");
EntityManager man1=fac.createEntityManager();
EntityManager man2=fac.createEntityManager();

//Here in this code Since we have enabled 2nd level cache only 2 time it will hit the database server
//2 times it will hit the database server
man1.find(User.class, 1);//It will hit the database server,object is going to store in 1st and 2nd level cache
man1.find(User.class, 2);//It will hit the database server,object is going to store in 1st and 2nd level cache
//It will not hit the database server
man2.find(User.class, 1);//Nothing is there in 1st Level cache but object is present in 2nd level cache
man2.find(User.class, 2);//Nothing is there in 1st Level cache but object is present in 2nd level cache
}

}
```

//After enabling 2nd Level cache
//only 2 times it will hit the database server

Association Mapping Summary

*In oneToOneUni Mapping -→ Totally 2 tables are going to create-→1 table will have FK

*In Bi-Directional Mapping owning side and non owning side will come into picture.

Note:-

In Maven Project if you create 2 packages and in each package if you try to create same class Maven will not allow you will get Exception----DulicateMappingException
The [org.jsp.onetoonebi.PanCard] and [org.jsp.associationpractice.PanCard] entities share the same JPA entity name: [PanCard] which is not allowed!

*In oneToOneBi-Directional Mapping 2 table are going to create but both the table will have FK which is not required so we need to use owning side and non owning side

*While writing the code in OneToOneBi-Directional Mapping
p.setCard(card);
//card.setPerson(p);//If you don't use this line then in PanCard table person_id(FK) will be null
But you will not get any exception.

Use cascade attribute to avoid writing multiple persist( );

*OneToManyUni Ditectional mapping ------→3 Tables are going to create but the 3rd Table Contains the PK's of first two tables..
We can not avoid the creation of 3rd table in oneToManyUni
*3rd Table is called as JoinTable ----Which is used to build the relationship
*when you do not use a direct foreign key in the child entity. This approach ensures that the relationship is managed via a separate table,

Explanation:-

```
@JoinTable(    name = "Department_Employee",
 joinColumns = @JoinColumn(name = "departmentId"),
 inverseJoinColumns = @JoinColumn(name = "employeeId"))
```

- **name = "Department_Employee"**: Specifies the name of the join table.
- **joinColumns = @JoinColumn(name = "departmentId")**: Defines the foreign key column in the join table that references the primary key of the owning entity (Department)
- **inverseJoinColumns = @JoinColumn(name = "employeeId")**: Defines the foreign key column in the join table that references the primary key of the target entity or Non –Owining Entity(Employee).

•**name = "Department_Employee"**: Specifies the name of the join table.
•**joinColumns = @JoinColumn(name = "departmentId")**: Defines the foreign key column in the join table that references the primary key of the owning entity (Department)
•**inverseJoinColumns = @JoinColumn(name = "employeeId")**: Defines the foreign key column in the join table that references the primary key of the target entity or Non –Owining Entity(Employee).