Course :: Hibernate with JPA
duration :: 70 to 80 Sessions
course fee :: Rs: 3500/-
pre-requsistes :: core Java , Basics of JDBC , Basics of SQL, Common Sense
Timings : 4pm to 5:30 pm (weekly 6/7 classses)
Free Tools :: Maven,Gradle , log4j ,slf4j, junit, mockito , GIT/GITHUB,
docker, jenkins,JIRA and etc... (15+ tools)
free common topics :: Resume preparation, Interview tips,
How to show gap as exprieiecen, Initial days
in the software company , AGILE-SCRUM Process
and etc.. (10+)
Mini Projects :: 5 + Mini Projects by applying design patterns, realtime coding standards
in the form layered applications.

Faculty Details ::
==========
FB Group name :: natarazjavaarena (private group)
FB group url :: https://www.facebook.com/groups/388095825162910
To collect material :: https://www.facebook.com/groups/388095825162910/files
email id :: natarazjavaarena@gmail.com
For New Batches Info :: https://nareshit.com/new-batches-hyderabad/
youtube url :: https://www.youtube.com/c/NareshIT

Admin details ::
==========
Mr.Srikanth:+91 6302968665 (whats up only)

Whose needs this course ?
==========
Java Freshers (optional -But added advantage) ,
Java experienced (1+,2+3+,4+ and etc...)

-> Learning only Spring and spring boot (good)
=> Learing sping and spring boot with Hiberante + JPA (very good)

note:: To understand and to use spring ORM module, Spring data JPA module
effectively we need hibernate with JPA knowledge.

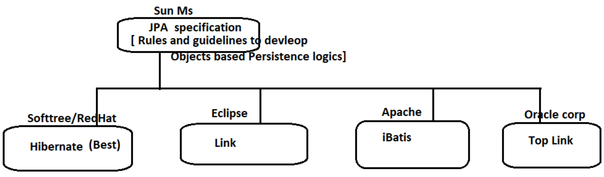
=>if u r planing to learn only 1 framework then learn spring ,spring boot
=>if u r planing to learn only 2 frameworks then learn spring,spring boot and hiberante + JPA
=>if u r planing to learn only 3 frameworks then learn spring,spring boot , hiberante + JPA
and WebServices (Restful webServices)

What is the different b/w JPA and Hiberante?

JPA :: Java Persistence API

JPA is plain or software specification to devlelop objects based Db s/w
independent (No SQL queries) persistence logic (insert,update,delete, Select operations)

=>Hibernate , iBastic , Eclipse Link and etc.. are the Implementation softwares /Tools that are given
based on the rules and guidelines of JPA plan or specification.

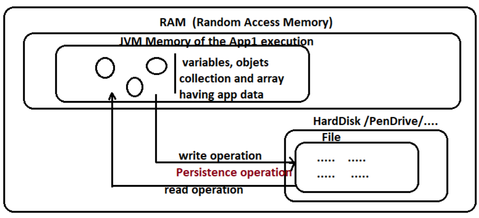


Persistnece
==========
=>It is the process of storing and managing data for long is called persistence
=> The Persistecne activities take place by taking the support of files in Computer
File System (All drives of computer (like c:, d:, e: and etc..) where the data can saved
and reused is called File System)
=> All files and folders managed by Operating system of the computer is called
File System.

What is the need of persistence seperately when we are alredy storing data of
of App using variables, objects, arrays ,collections and etc..?

Ans) The variable, collections, objects, arrays of the Java App allocates memory in
the JVM Memory of the RAM (Even stack and heap memories are part of JVM Memory)
This JVM Memory of the RAM will given to our Java App only during the execution of
the App.. Once the App execution is completed the JVM memory of the RAM will be
vanished , So we loose the data kept in variables, objects, array and collections, Due to this
the inputs and outpust of the App in one execution can not be used across the multiple
executions.

To overcome this problem we need to write Application data (inputs and outputs) to File
System (files of the computer that allocates memory in Harddisk or in some other
permement storage unit like Pendrive, CD,DVD and etc..) using the cocept called
Persistence.



PErsistence Store
==========
=>The place in computer file system where data will be saved and managed for long time is called
Persistence store
eg:: Files in File System
DB s/ws (internally uses the files of File system)
(oracle,mysql, postgresql and etc..)

Persistecne operations
==========
=>The insert ,update, delete and select opeations performed on the
data stored in Persistnece store are called Persistence opeations.. these
are alos called as CURD /CRUD/SCUD operations.

CURD/CRUD
==========
C--->create (insert)
U -->Update (modify)
R --->Read (select)
D ---> Delete (remove)

SCUD
=====
S--->Select
C--->create
U --->Update
D -->Delete

Persistnece logic

=> The Logic that is used to perform Persistence operations is called PErsistence logic
eg:: I/o stream code (core java) , jdbc code (adv.java) ,
hibernate code , itbatis code, and etc..

Pesistence technology/Framewrok
==========
=>The Technology /framework that is given to develop persistence logic
is called Pesistence technology or framework

eg: JDBC (technology), hiberante (framework), iBatis (framework)
and etc..

Conclusion:: Pesistence technology /framework is given to develope
persistence logics for performing persistence opeations(CURD Operations)
on the data of Persistence Stores (files or DB s/w)

Java Learning means
a) Language learning (Core Java)
(raw material)
b) Technologies learning (adv.java)
(Semi-finished material)
c) frameworks (spring,spring boot
hibernate and etc..)
(fully finished material)

One Course for All Java Frameworks
==========
Spring Basics, Spring boot , Spring Rest(web service) , MicroServices (Spring Stack)
with Free Hiberante with JPA or angular or ReactJs
(starts in the month of jan /feb in 7am slot)
[150 sessions] --> daily 1.30 to 1.45 hours
[90 sessions] --> daily 3 to 3.5 hours

fee: 8000/-- (Including free topics)
fee : 6000/- (only spring stack)

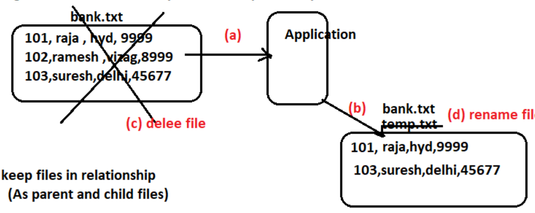
IO streams
Java App ----------> Files

JDBC Code +SQL Queries
Java App ----------> RDBMs DB s/w (oracle,mysql, postgresql and etc..)

o-r mapping code
Java App ----------> RDBMs DB s/w (oracle,mysql, postgresql and etc..)
(like hibernate code)

Limitations of Files as Persistence store

a) Security is not there
b) can not store huge amount data
c) No constraints can be applied (No primary key, unique key and etc..)
d) Data Redundency problem (Data duplication)
e) NO SQL support
f) Searching and reading data by applying multiple conditions is very complex
g) Performing DELETE and UPDATE operations is quiet complex



h) Can not keep files in relationship
(As parent and child files)

and etc..

To overcome the above problems use DB s/w as the Persistence store
(RDMS Db s/w)

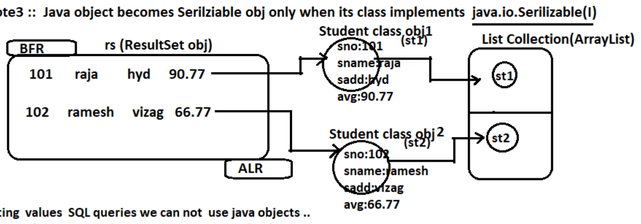
Limitations of JDBC to develop persistence logic

a) JDBC persinstence needs the SQL queries support and SQL queries are DB s/w dependent SQL Queries , So indirectly JDBC Persistence logic DBs/w dependent persistence logic.

b) JDBC persistence logic is not portable across the multiple Db s/ws .So changing DB s/w in the middle of development or production is quite complex process.

c) The Select SQL query execution gives Resultset object which is not Serializable obj So we can not that object over the network .. For this we need to convert ResultSet obj records in to Collection of Java class objects

note1:: We can send only Serilizable objs data over the network
note2:: All java collections (List or Set or Map) are Serilizable objs by default
note3 :: Java object becomes Serilziable obj only when its class implements java.io.Serilizable(I)



d) While setting values SQL queries we can not use java objects .. Though data is there in Java object we must convert the data to simple values in order to set Query Input values.

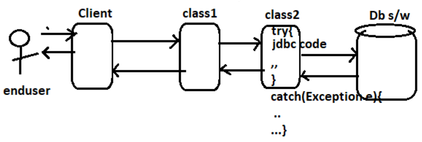
```
PreparedStatement ps=con.prepareStatement("INSERT INTO STUDENT VALUES(?,?,?,?)");

Student st=new Student(101,"raja","hyd",56.77f);

ps.setInt(1,st.getSno());
ps.setString(2,st.getSname());
ps.setString(3,st.getSadd());
ps.setFloat(4,st.getAvg());

int result=ps.executeUpdate();
```

Here object data is converted into simple values in order to set query params

Student class obj(st)
101
raja
hyd
56.77f

e) Throws SQLException for all problems i.e there are no detailed Excpetion classes.

f) SQLException is checked Exception , So we are forced to catch and handle exception This kills exception propagation possibility in layered Application.



if exception is raised in class2 jdbc code .. the control goes to catch back of class2 and the exception will be eaten /suppressed there.. so the raised does not go to enduser through class1 ,client clases..

g) Lots of Boilerplate code is avaiable in JDBC Programming

Jdbc App1 to insert record		JDBC App2 to select record	
=>Load jdbc driver classs => Establish the connection => create JDBC statement obj	common logics	=>Load jdbc driver classs => Establish the connection => create JDBC statement obj	common logics
=>Send and execute INSERT SQL query =>Gather INSERT SQL query result and process it	Application specific logics	=>Send and execute SELECT SQL query =>Gather SELECT SQL query results and process it	Application specific logics
=>catch and Handle exceptions => Close jdbc objs	common logics	=>catch and Handle exceptions => Close jdbc objs	common logics

=>Here common logics are called Boilerplate code

note:: The code that repeates in muliple parts of the Project either with no changes or with minor changes is called boilerplate code ..

h) Multiple services like TxMgmt , versoning, timestamping ,caching and etc.. should implemented manually which is burdern to the Programmers.

TxMgmt (Transaction Management)

=>It is the process of combining related operations into single unit and executing them by applying do every thing or nothing principle

task:: transfer money
- 1.widraw amt from source account
- 2.deposite amt into dest account

Needs TxMgmt support to execute withdraw, deposite logics by applying do everything or nothing principle

Versoning
of
=>It keeps track how many time the record is modifying using Persistence logic

Time stamping
It keeps track of when the record is lastly updated

Student Db table in oracle Db s/w

(pk) sno	sname	sadd	avg	(n) ver	lastupdated (timestamp)
101	raja	~~hyd~~	~~99.88~~	~~0~~	~~17-DEC-21 17:25:45~~
		vizag	90	~~1~~	~~17-DEC-21 17:25:54~~
				2	17-DEC-21 17:26:10
				version	

and etc..

Serialization
==========

=> The Process of converting java object data into bits and bytes is called Serizaltion..
These bits and bytes can be sent over the network and can be written to file
=> We can send only Serializable object's data either to file or over the network
=> To make the java object as Serilizable object the class of that object must implement
java.io.Serializable(I) (Marker interface -Empty interface)

=> The JVM makes the object of class as Serilizable object by seeing the impl of
java.io.Serilizable(I) .. In that process JVM becomes ready to conver tSeriable object's into
bits and bytes.

=>The marker interface (empty interface) Does not do any thing on this own directly.. by Seeing
that marker interface implemetation the JVM will create object as the special object having
special behaviour.

```
public class Emp1 {
  private int eno;
   private String ename;
   ...
   ..
}
```

Emp1 e1=new Emp1(101,"raja");

e1
101
raja

(Normal object becoz Emp1 class is not implenting any Marker Interface)

```
public class Emp2   implements java.io.Serilizable{
  private int eno;
   private String ename;
   ...
   ..
}
```

Emp2 e2=new Emp2(101,"raja");

e2
101
raja

[Special obj (Serilizable obj) becoz the class is implementing the marker interface calls java.io.Serilziable.So this object data can be written to file or can be sent over the network in the for bits and bytes.

=>The interface that makes the undlerying JVM/Server/Container to provide special run time capabilities to impl class object is called marker interface/Tag interface

eg:: java.io.Serilziable (I)
java.lang.Cloneable(I)
java.rmi.Remote (I)
javax.servlet.SingleThreadModel(I)

These are empty interface.

To overcome the limitations of JDBC, use o-r mapping Persistence logic where we write peristence logic using objects with out SQL Queries as DB s/w independent persistence logic.

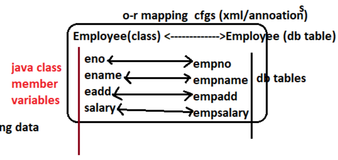
What is ORM (o-r mapping)?

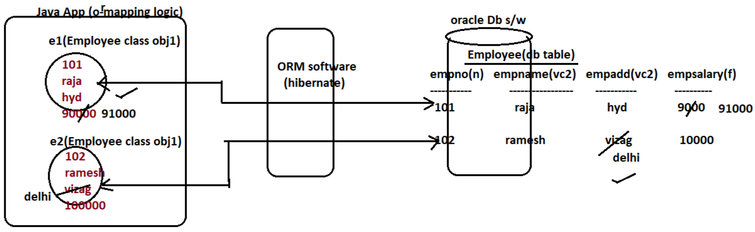
The Process mapping/linking java classes with db tables , the member vairables of java classes with Db table cols and making the objects of java classes representing the records of db tables having synchronization among them is called o-r mapping.

1 java class with 1 db table
1 memberiable of java class with 1 col of db table
1 object of java class with 1 record of db table

==========
java class/Entity class/Persistence class/Model class
==========

```
public class Employee{
  //properites/attributes/member variable
   private int eno;
   private String ename;
   private String eadd;
   private float salary;

  //setters && getters
     ....        //setters methods are for setting/modifying data
     ...         //getter methods for reading data
}
```





=> The modifications done objects of java class reflects to db table row and vice-versa. this is called synchronization b/w objects and db table rows.

=>In o-r mapping all persistence operations take place through objects of java class /Entity class with out SQL Queries
So we can this objects basaed Persistence logic is DB s/w independent persistence logic.

=>The ORM software like hibernate recived objects based Persistence instructions from the Java Apps and generates the JDBC code +SQL queries dynamically to complete persistence operation in the Db s/w.

=> In 0-rm mapping

=>save object means inserting the record in to db table by collecting the object's data and generting Jdbc code + INSERT SQL query internally.

=>update object means updating the record in db table that is represented by the object data by generting Jdbc code + UPDATE SQL query internally

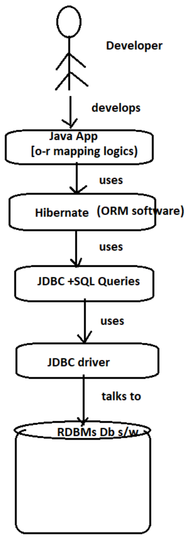
=>load object means selecting the record from db table by generting Jdbc code + SELECT SQL query internally

=>Deleting object means deleting/removing the record from db table represented by the object by generating jdbc code+ DELETE SQL query.

(hidling the details)
note:: All ORM software internally provides abstraction on JDBC i.e they use JDBC code + SQL querery internally by hiding from programmers to complete the objects based persistence instructions given by Java App having 0-R mapping logic.

List of ORM softwares (Java based)
==========

hibernate ----------> from softtree/RedHat (1)
iBatis ----------> form apache (3)
Eclipse Link----------> from Eclipse (2)
Toplink----------> from Oracle corp (4)
MyBatis ----------> from apache
JDO ----------> from Adobe
and etc...



=>In o-r mapping the obejcts of java class /entity classes /model class are like actors playing the role of insert ,update, delete and select persistence operations..

=>For example object is representing db table record.. if i used that object to delete record the ORM s/w generates jdbc code + DELETE SQL Query and deletes the record but does not deletes the object ..In fact same object can be used to perform other perisstence operations on other recorrd.. It is like Actor plays role/character in movie.. if the role/character died in the moive it does not mean actor is died.. The same actor can perform another role either is same movie or different movie.

The objects in o-r mapping are called objects/ Entity objects

Advantages of o-r mapping
==========

=>Allows to develop objects basaed Db s/w independent persistence logic
=> o-r mapping persistence logic is portable across the multiple Db s/ws , So changing Db s/w in the middle of develomentn, production is easy

project is developnent is software company	Development mode of Project
Project is released and client is using the Project	Production mode of the Project

=> No need of writing SQL Queries

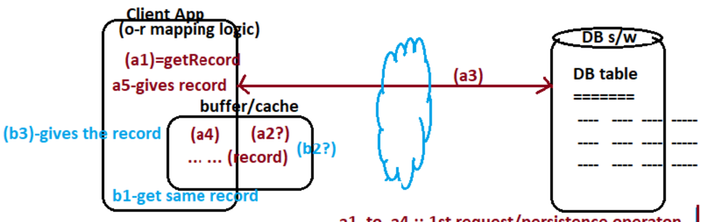
=> All Persistence operations can be done through objects .. So no need of converting java obejcts data into simple values.

=> By taking entity classes as Serilizable classes we can send db table records over the network in the form of Serilizable Entity class objs
(For this Entity classes should implement Serializable (I))

=> Gives lots built-in services support like versoining, timestamping, Caching and etc..

Versoning :: keeps track of how many times the record is modified
timestamp:: keeps track when the object is lastly updated.
caching :: keeps records/objects collected from db table in buffer/cache and uses them across the multiple same requests/persistence operations to reduce the network round trips b/w java app and db s/w.

cache/buffer is a temporary that holds data for temporary period.. In Client - Server App the cache at client side holds server supplied results and uses them across the multiple requests to reduce network round trips b/w client app and server App.



a1 to a4 :: 1st request/persistence operaton
b1 to b3 :: 2nd request/persistence operation

(both are asking for same record)

note:: To take the benifit of caching we need to empty the cache at regular intervals. (like emptying cache for every 1 min or 30 secs and etc..)

=> o-r mapping is capable of dynamic schema generation
(Based on the details of entity classes the db tables will be generted dynamically)
=> o-r mapping provides generators to generate PK column values dynamically
PK :: Primary key
es the
=> Address mismatches b/w Java features and RDBMS DB s/w features.

For example :: we can keep entity classes in the inheritence but we can not keep related db table in inheritence becoz java supports inheritence and RDBMS Db s/w does not support inheritence .. So o-r mapping should address this mismatch should make classes of Inheritence relationship pointing to 1 or more db tables.

and etc..

Is the JDBC technology outdated? if not outdated tell me where should i use JDBC and where should i use o-r mapping like hibernate?

Ans) JDBC is not outdated and will not be outdate in future also becoz the bottom code the required in Java to interact with Db s/w is JDBC code + SQL Quries irrespective of top layeres are using.

=>if App is getting huge amount of data batch by batch for processing like 30k to 2lac records at time then prefere using JDBC becoz it creates single ResultSet object representing all the records for Processing , So less memory and lesss cpu is required..

if we use o-r mapping like hibernate here , the app has to create 30k to 2 lac objs at a time representing 30k to 2lac records that came at a time as a batch .then there is possiblity of system crash/Project crash while creating those objects at a time

eg:: Census (population count) , Survery Apps , marketing apps and etc..
(use JDBC here)

=> if App is getting less amount of data to process (at a time 30 to 40 records) then prefer working with o-r mapping like hibernate , So that we can take benifit various features of o-r mapping like DB portability , Generators, versioning, timestamping , caching and etc..

eg: Banking Apps , mediume scale e-commerce apps, college /University Apps and etc..

if App is getting very much huge amount data which is beyond storing and processing capacity of regular DB s/ws then prefer storing and managing that data with the support of BigData Frameworks like Hadoop or spark (both java based)

1024 bytes = 1kb
1024 kb => 1mb
1024 mb => 1gb
1024 gb = 1tb
1024 tb = 1 pb
1024 pb = 1 eb
1024 eb =1zb
1024 zb =1yb

and etc..

=> youtube generates 13PB per day
=> FB generates 10 TB per hour
and etc..

This amount data can not stored and procesed using the regular RDBMS DB s/w like oracle

For this kind huge amount data use BigData Systems like hadoop or spark which use collection of ordinary computers in a network to store process this kind of BigData..

Hibernate
=========

Type :: Java based ORM framework
version :: 5.6 (compitable with jdk1.8+)
vendor :: SoftTree (RedHat)
Open Source (Free and source code is avaiable)
Creator : Gavin king

To donwload hibernate :: https://sourceforge.net/projects/hibernate/
(download as zip file)

(or)
we can get hibernate librarries (jar files)
directly from internet while working maven/
gradle tools

Why the hibernate is named as hibernate?

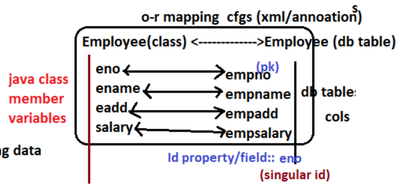
Ans) The process of holding things and using things for long time in rest mode is called hibernation or hibernate mode .. We can see this behaviour polar bear or kumbkarna (eats for six months and sleeps for sixmonths -during sleeping it uses the reseverd food in the body) .. Since the objects representing db table records holds the data of the records in the objects helps to manipulate db table records through objects .. So we can say the objects are in hibernatenation mode .. So the framework is named as hibernate.

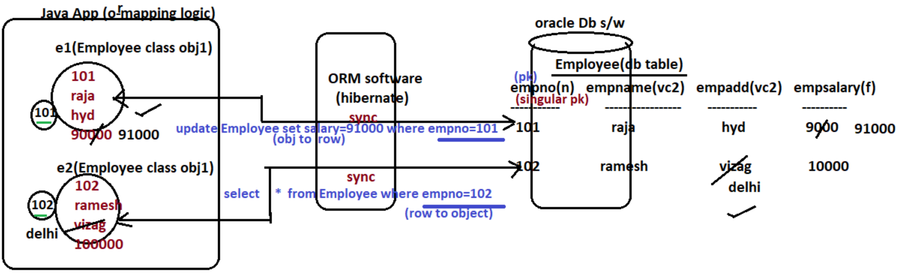
How the synchronization between objects of java class and db table records takes place?

Ans) synchronizatio means the modifications done in db table records reflects to objects and vice-versa.

==========
java class/Entity class/Persistence class/Model class
==========

```
public class Employee{
  //properites/attributes/member variable
    private int eno;
    private String ename;
    private String eadd;
    private float salary;

  //setters && getters
     ....        //setters methods are for setting/modifying data
     ...         //getter methods for reading data
}
```





The ORM software like hibernate generates update SQL query internally to synchronize the changes done in object to db table row .
The ORM software like hibernate generates SELECT SQL query internally to synchronize the changes done in db table row to Object.

=>The UPDATE , SELECT SQL Queries of Synchroization will be generated by taking Id property value of the Entity class objs as the criteria/condition value as shown in the diagram.

Strategies to pick the Entity class property/properties as Identity Property/Field
==========

strategy1:: if db table is having singular PK constraint then take that constraint single column related property in Entity class and make it as singular Id Filed using <id> tag or @Id Annotation

strategy2:: if db table is having composite PK constraint then take that constraint multiple columns related multiple properties in the Entity class and make them as composite Id Filed using <compsite-id> tag or @EmbeddedId Annotation

strategy3:: if db table is given with out PK constraint or unique constraint then reject that db table to use o-r mapping operations becoz it gives lots problems towards synchronization.

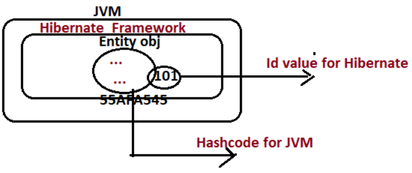
if PK costraint is applied on 1 col of db table then it is called singular PK
empno(PK)
if PK costraint is applied on multiple cols of db table togather then it is called composite PK
empno+ename(composite PK)

if one property of Entity class is taken as the Id property is then it is called singular ID property

if multiple properties of Entity class togather taken as the Id property is then it is called composite ID property

=>As part of o-r mapping cfgs which we do using annotations or xml the id filed cfg is mandatory.. becoz the id property value is very useful for ORM s/w to identity each object of Entity class and to even perform Sync b/w objects and db table rows.

=>JVM identifies each object by using its hashCode
=>Hibernate identifies each object Entity class by using its Id value.



Definitation of hibernate
==========

Hibernate is an open source light weight java based ORM framework to develop the objects based DB s/w independent o-r mapping persistence logic in Java ,Jee Applications.

open source
==========
=> Free s/w whose source code to exposed to outsiders/devleopers is caled open source
=> We get hibernate as free s/w and gives source code to devleopers as needed, So hibernate is called open source software.

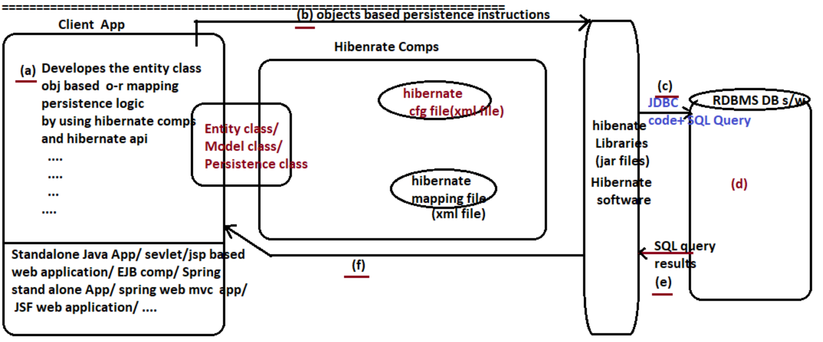
java, spring, hibernate and etc.. are open sources

Light weight
==========
=> hibernate f/w comes in the form zip file having small size
=> hibernate libraries(jars) can be downloaded dynamically from internet using maven/gradle tool support
=> To Execute code we just need
JRE/JVM + hibernate libaries (jar files) i.e
no need of having big setups like webservers , application servers and etc..
=> hibernate code (o-r mapping logic) can be integrated with any Java, JEE and Java frameworks Apps to make them talking to Db s/w.

note:: All above features make hibernate as ligth weigtht software.

How OpenSource organizations survive?
a) By offering certication exams
b) By offering training for certfincation exams
c) By conducting corporate tranings
d) By offering software services to companies like fixing complex problems
e) By publishing books and etc..

Hibernate App development setup or components of Hibernate App development
==========



note :: Here Client App means , not client to Hibernate .. It is client to Db s/w that uses hibernate based o-r mapping persistence logic to locate and interact with Db s/w. This Client App can any java App , JEE Technologies App, JEE framework App having hibernate code as the persistence logic.

w.r.t the diagram
==========

(a) client App uses hibernate api and the hibernate comps to develop entity class objects based o-r mapping persistence logic

(b) Client App gives objects based persistence instructions to ORM s/w (hibernate)

(c) The ORM s/w internally uses its libraries (jar files) and lots built-in comps to generate JDBC code +SQL query for obejcts based Persistence instructions.

(d) SQL goes to Db s/w and executes in Db s/w to complete the Persistence operation.

(e) DB s/w gives SQLQuery execution results back to ORM s/w (hibernate)

(f) hiberante convers the SQL Query results as required Client App an sends to Client App

The comps of Hibernate App devleopment
==========

(a) hibernate cfg file(s) (xml file) (b) hibernate mapping file(s) (xml file)
(c) Entity class(es) d) Client App(s) e) Hibernate Libraries (hibernate jar files)

hibernate cfg file (xml)
==========

=>Contains jdbc driver details helping the hiberante f/w to locate the Db s/w to establish the connection with Db s/w.
=> Also contain other details like dialect to use , dynamic shema generation mode and etc...

Dialect :: It is built-in comp of Hibernate whose actually generates the SQL query by taking objects based persistence instruction
Dynamic shechema Generation:: This tells hibernate wheather to create or not db table based on mapping file/entity class details

=> contains the names and locations of hibernate mapping file(s)

=> if we do not specify name of the hibernate cfg file in the Application then the app takes hibernate.cfg.xml file of classpath as default hibernate cfg file name and location.
=> any <file-name>.xml can be taken as the hibernate cfg file .. if u do not specify any file name the above said default file name (hibernate.cfg.xml) will be taken as the cfg file.

=>The information in hibernate cfg file will be given in the form of keys and values where the keys are fixed hiberante cfg properties and values will be given by programmer bsaed on the DB s/w and its version he is using..

sample hibernate.cfg.xml file
==========

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
    <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">manager</property>

    <property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>

    <mapping resource="comt/nt/hbm/Employee.hbm.xml"/>

  </session-factory>
</hibernate-configuration>
```

property name(key)
property value (value)
jdbc properties to locate and connect the Db s/w
Dialect comp to use to generate SQL queries
name(s) of mapping file(s)
the word "hibernate" in key/property name is optional

=>The hibernate App internally reads and store the infomation collected from hibernate cfg file and specified mapping file in a special object Sessionfactory object

To get all hibernate cfg properties (keys) use the following url
https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/session-configuration.html

=>Nearly 200 + hiberante cfg properties are avaiable as fixed keys ...to provide different inputs..

=>We generally take hibernate cfg file in the Apps on one per Db s/w basis.. if App wants to talk with one Db s/w then we take one hibernate cfg file .. similary if App wants to talk with two Db s/ws then we need take two hiberrnate cfg files.

Entity class /Model class/ Persistence class
==========

=>This class must be develop Java bean class (class with setter methods and getter methods)
=> This class will be mapped with Db table and this properties/attributes/fileds will be mapped with Db table cols.

=>The Client Apps use the objects of Entity class to provide objects based Persistence instructions to ORM software(hibernate) as part o-r mapping logic devleopment..

=> It will be taken on 1 per db table table basis..
if project is having 10 db tables then we take 10 Entity classes.
=> While developing these entity classes we can apply all java features like inheritence, composition and etc..

eg::

```
public class Employee{
  private int eno;
  private String ename;
  private String eadd;
  private float salary;

  //4 setter methods
    ....
    ....
    ....
  //4 getter methods

    ...
    ....
    ..

}
```

Hibernate Mapping file (xml file)

=>any <file-name>.xml can be taken as hibernate mapping file .. There is no default file name for hibernate mapping file

=> we generally take hibernate mapping file on 1 per db table/Entity class basis.. if the project is having 20 db tables then we take 20 Entity classes and 20 mapping files.

=> All mapping file names and locations must be specified in hibernate cfg file

=>Each mapping contains o-r mapping cfgs like mapping class with db table , member variables with db table cols and also cfg Identity property/Field cfg

=>The recomanded notation of hibernate mapping file name is <filename>.hbm.xml

=>The infomation read from mapping files, hibernate cfg file will be used to construct HB SessionFactory object For this reason we see <session-factory> tag in hibernate cfg file.

Sample hibernrate mapping file

Employee.hbm.xml

```
<hibernate-mapping>
  <class name="<pkg>.Employee"  table="EMPLOYEE">
     <id name="eno"  column="EMPNO"/>  <!-- Id Field cfg -->
     <property name="ename"  column="ENAME"/>
     <property name="eadd"  column="EADD"/>
     <property name="salary"  column="SALARY"/>
  </class>
</hibernate-mapping>
```

entity class mapped with dbtable

properties mapped with db table colums

Employee (db table in oracle)

=> EMPNO (pk) (n)
=>ENAME (VC2)
=>EADD (VC2)
=>SALARY (FLOAT)

if Hibernate App is taking single DB s/w with db table names Employee , Department the

The HB cfg file name is :: hibernate.cfg.xml
The mapping file names are :: Employee.hbm.xml
Department.hbml.xml
Entity class names are :: Employee , Depepartment (java beans)

=>Naming Conventions to fallow

Client App

=> It is not Client to HB framework /software .. It is client to Db s/w having o-r mapping Persinstence logic

=> The client App can be any Java , JEE App or Java Frameworks App having hibernate api /libraries based o-r mapping persistence logic using the obejcts of Entity classes.

=> In layered Apps we generally hibernate persistnece logic in DAO class (DataAccessObject class)

=>The java class that contains only Persistence logic and seperate persistence logics from other logics and also makes persistence logic as reusable .loosely coupled logics (gives flexiblity to modify) is DAO class.

note:: we can place o-r mapping persistgence logic hibernate either directly in Client App (main(-,-) of main lass) or in DAO class if the App is Layered App.

Hibernate Libaries (jar files)

=>Represents Hibernate F/w or Software

=> these jar file can be gathered from hiberante software that is downloaded in the form of zip file

<hibernate_home>\lib\required folder jar files (16 jar files)

=> if u r working with Maven/Gradle build tools in the development of Hbiernate Apps then we can get jar files directly from Internet Repositories with out downloading any hibernate software from the internet.

=> These jar files contains api form Programmers which can be used devleop the o-r mapping persistence logic

API (Application Programming Interface)

=> In Java API means set of classes,enums, annotations, interfaces that are given in the form pkgs
=> In c++ API means set of classes and functions that are given in the form header files

while developing hibernate's o-r mapping persistence logic we can use

a) pre-defined hibernate api (collected from hibernate jar files)
b) user-defined apis (programmer devleoped pkgs having classes, interfaces and etc..)
c) third party apis (like hikari cp , apache dbcp,lombok api and etc..)

Library= set of APIs

We can develope hibernate Apps in two styles

a) Using xml driven cfgs
[seperate mapping files(xml), seperate entity classes(java bens))
=>The o-r mapping cfgs takes in mapping files called hibernate mapping files

b) Using mapping annotation driven cfgs (Best)
[No mappiping file (xml files) , In Entity classes itself annotation will be added for o-r mapping cfgs]

Different types of Projects

a) Product type Projects
[requirements will be gathered through market survey and the software company developes the Project as product and keeps the market for sale]
eg:: Tally , window OS , MS- office , Anto-Virus s/w and etc..

b) Sevices based Projects
[The project will be developed for certain Client organization by gathering requirements from the Client organization]
eg:: AMEX project by CTS , Airbus project by capgemini
TS E-governance by TCS , Central Food For Work by TCS
Aadhar project by Infosys and etc..

The Developers work modes in Projects /Products

a) Software engineer in Scratch Level Development (0 to 100% development)

b) Software engineer in Maintainance Projects (Bug fixing and issue tracking)

c) Software enginner in Migration Projects (Chaging Project technologies or servers or proceses and etc)

=>We can place hibernate's o-r mapping persistence logic in any java , jee , java framework app either directly in the App or in the DAO class (layered App) of the Application. while writing this Persistence logic we need 3 important comps.

a) Configuration object
b) SessionFactory object
c) Session object

In JDBC persistence logic development we need Connection, Statement, Resultset and etc.. objects similarly hibernate's persistence logic development also needs 3 more objects

==========
Configuration object
==========

=>Activates hiberante f/w by using the jar files (libraries) that are added to CLASSPATH/BUILDPATH
=>Reads and holds the given hibernate cfg file or default hibernate file content..
=> This object is called Bootstrap object of hibernate becoz it is bootstrapping (starting) hiberate f/w in the required Java app based on the jar files added to CLASSPATH/Build path.

```
Configuration cfg=new Configuration();
cfg.configure("com/nt/cfgs/hiberante.cfg.xml");
```

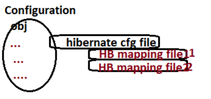
Activates/Bootstraps the hibernate framwork by taking given hibernate cfg file from the specified location

(or)

```
Configuration cfg=new Configuration();
cfg.configure();
```

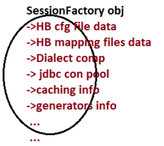
Activates /Bootstraps the hibernate framework by taking hibernate.cfg.xml file of CLASSPATH/ BUILDATH Location. (src folder of Eclipse Project)

note:: The "src" folder of eclipse Project will be there in CLASSPATH /BUILD Path by default..



SessionFactory object (Factory to create Session objs)
==========

=>Created from Configuration object
=>Collects hibernate cfg file info, also hibernate mapping files info , prepares neccessary other objects like dialect , data soruce pointing to jdbc con pool , caches and etc. Lastly creates SessionFactory object having all those objects and infogration.



=>It is heavy weight becoz it contains multiple details that is required for whole App based on the instructions collected from mapping files(xml files) , cfg file (xml file)
=> It is immutable object (Once the object is created with data.. it can not be modified)
=> It is Thread safe object i.e though multiple threads are acting on single object.. no one can modify the data
=> It is Long lived object in Hibernate App...
=> It is the object of Java class that implements org.hibernate.SessionFactory(I).

note :: if any object name is called with Interface.. then it is not called object of interface.. It is called object of the java class that implements the Interface.

```
code::
        SessionFactory factory=cfg.buildSessionFactory();
```

Session object
==========

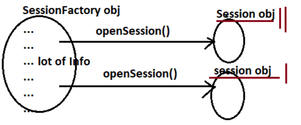
=>Created using SessionFactory object
=> Collects con object from jdbc con pool and Mapping file info and creates Session object

Session object= con++
(jdbc con obj + Mapping file information)

=>Session object is bridge b/w Java App and ORM software(Hibernate framework) to given objects based Persinstence instructions to ORM software(Hibernate framework).
=> The Applications uses HB session object to given Entity class objects based PErsistence instructions to Hibernate software (framework).. The Hiberante software internally uses Dialect comp support to generate the equalent SQL Queries based on the recieved Objects based Persistence instructions..

=> It is light weight object becoz it contains very little information like con + Mapping file data..
=> It is not thread safe object by default..
=> It is mutable objects (Data can be changed)
=> We generally create more Session objs in one App one 1 Session per each persistence opeation, So we can say these are short lived objects.

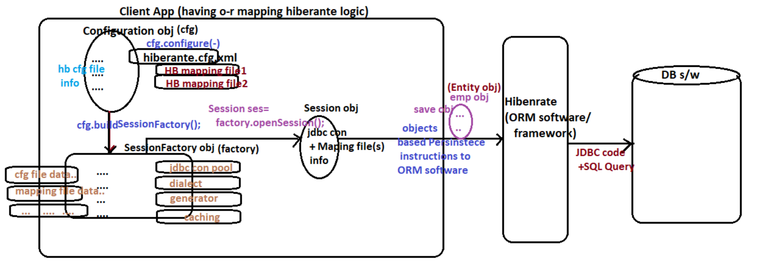
=>Session object means the object of java class that implements org.hibernate.Session(I).




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

In client App / DAO class
==========

```
//Activate or BootStrap Hiberante f/w
   Configuration cfg=new Configuration();
   cfg.configure("com/nt/cfgs/hiberante.cfg.xml");

// Create SessionFactory object
   SessionFactory factory=cfg.buildSessionFactory();

//create Session object
      Session ses=factory.openSession();
```



=>Generally the above 3 objects will be created in the Hibernate App as shown below
=> Configuration obj will be created on 1 per Db s/w basis
=> SessionFactory obj will be created on 1 per DB s/w basis
=> Session object will be created on 1 per persistence operation basis.

Persistence opeations in hibenrate based o-r mapping logic

single row operation (Manipulates one record at a time)

- ses.save(-) / ses.persist(-) — For save obj opeation (Inserting record)
- ses.load(-,-) / ses.get(-,-) — For Load object opeation (selecting a record)
- ses.update(-) — for update object opeation (for updating a record)
- ses.delete(-) — For delete object operation (for deleting a record)
- ses.saveOrUpdate(-) / ses.merge(-) — For save or update obj operation) (for saving or updating the record)

and etc..

Bulk operations (Manipulates more than 1 record at a time)

a) HQL /JPQL
b) Native SQL
c) HB Criterial API
d) JPA Criteria API

HQL :: Hiberante Query Language
JPQL :: Java PErsistence Query Language

first Hiberante App to save the object (inserting Object data as the record into db table)
==========

step1) keep the following software setup ready

=>Eclipse IDE (2019 /2020/2021 releases)
=>Jdk s/w (java 8+)
=>Oracle any version (11g)

SQL Developer for oracle (optional)

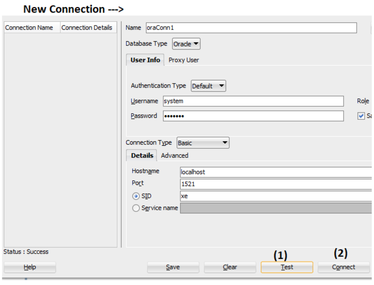
To download SQL developer (seperate download and installation)
---
https://www.oracle.com/tools/downloads/sqldev-downloads.html

=>GUI DB tool simplifies the DB Operations though we are not strong in SQL Querites
eg:: SQL developer for oracle
mysql workbench for mysql
TOAD for oracle
TOAD for mysql
an etc..

step2) Create db table in oracle using SQL Developer

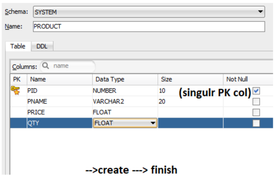
a) Connect to oracle from SQL developer

New Connection --->



b) create db table Product having columns pid(n)(pk),pname(vc2), price(float) ,qty(float)

Right click on oraConn1 ---> create table --->



-->create ---> finish

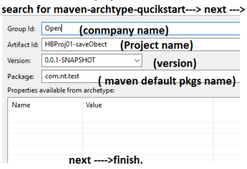
Generated SQL query

```
CREATE TABLE "SYSTEM"."PRODUCT"
(    "PID" NUMBER(10,0) NOT NULL ENABLE,
     "PNAME" VARCHAR2(20 BYTE),
     "PRICE" FLOAT(126),
     "QTY" FLOAT(126),
     CONSTRAINT "PRODUCT_PK" PRIMARY KEY ("PID"))
```

step3) Launch Eclipse IDE by chooing workspace folder
(G:\Worskpaces\hibernate\NTHB412)

step4) create Maven Project using "maven-archetype-quickstart" as the archetype (Project Template)

right cick -->new --> maven project ----> next --->
type and search for maven-archtype-qucikstart---> next --->



next ---->finish.

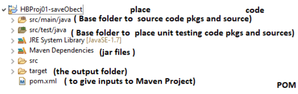
=>Maven, Gradle and etc.. are the Project build tools who simplifies project development , deployment activities.

=>In MAve archetypes are the Project Template took kid who given different structures of directories for different projects.

popular archetypes are :: maven-archetype-quickstart (for standalone app
maven-archetype-webapp (for web application app)
and etc..

In maven Project/ jar file/ plugin is identified with 3 details
a) groupId (company name)
b) Artifact Id (project name/ jar file name/plugin name)
c) Version (project version/jar file version/plugon version)

step5) Understand maven's stanalone Project's directory structure..



POM :: Project Object Model

step6) Perform the following operation to chanage the java version

a) In pom.xml

```
                                15
<maven.compiler.source>1.7</maven.compiler.source>
<maven.compiler.target>1.7</maven.compiler.target>
                                15
```

(we can give any number aabove 8)
(In latest versions of eclipse we can java version with out installing it explicitly)

b) perform maven update project
right click on project --->maven -->update -> ....

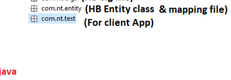
step7) add The following the jar file to project classpath/build path by adding the following entries in pom.xml file.

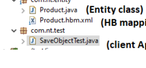
=>In maven jar files are called dependencies , so we need to add the following <dependent> tags under <dependencies> tag.

```
    <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.6.3.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.oracle.ojdbc/ojdbc8 -->
<dependency>
    <groupId>com.oracle.ojdbc</groupId>
    <artifactId>ojdbc8</artifactId>
    <version>19.3.0.0</version>
</dependency>
```

collect these details from mvnrepository.com by searching for hibernate-core , ojdbc8 .

step8) create the following pkgs in "src/main/java" folder





Product.java
==========

```
//Entity calss or BO class or Model class or Persistence class
package com.nt.entity;
```

Java bena class.

```
public class Product {
        private  Integer pid;
        private String pname;
        private Float price;
        private  Float qty;

        //setters & getters  (Alt+shift+S ,R )
        public Integer getPid() {
                return pid;
        }
        public void setPid(Integer pid) {
                this.pid = pid;
        }
        public String getPname() {
                return pname;
        }
        public void setPname(String pname) {
                this.pname = pname;
        }
        public Float getPrice() {
                return price;
        }
        public void setPrice(Float price) {
                this.price = price;
        }
        public Float getQty() {
                return qty;
        }
        public void setQty(Float qty) {
                this.qty = qty;
        }


    //toString()  (Alt+shift+S, S)
        @Override
        public String toString() {
                return "Product [pid=" + pid + ", pname=" + pname + ", price=" + price + ", qty=" + qty + "]";
        }

}
```

step8) Develope hiberrnate mapping file

Product.hbml.xml

step10)Develope hiberrnate mapping file

Product.hbml.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class  name="com.nt.entity.Product"  table="PRODUCT">
   <!-- o-r mapping -->
   <id name="pid" column="PID"/>  <!--  singular Id Field cfg -->
   <property name="pname"  column="PNAME"/>
   <property name="price"  column="PRICE"/>
   <property name="qty"  column="QTY"/>
  </class>
</hibernate-mapping>
```

xml prolog -Indicates this is an xml document

importing of hibernate mapping related dtd rules [get this import statement either from internet or from sample mapping files]

hibenrate mapping file containing o-r mapping cfgs

note:: To make all the developers of certain technology or framework developing xml files using same tags and attributes with different values , the vendor of the technology supplies set rules and guildelines in the form of DTD or XSD documents /rules..

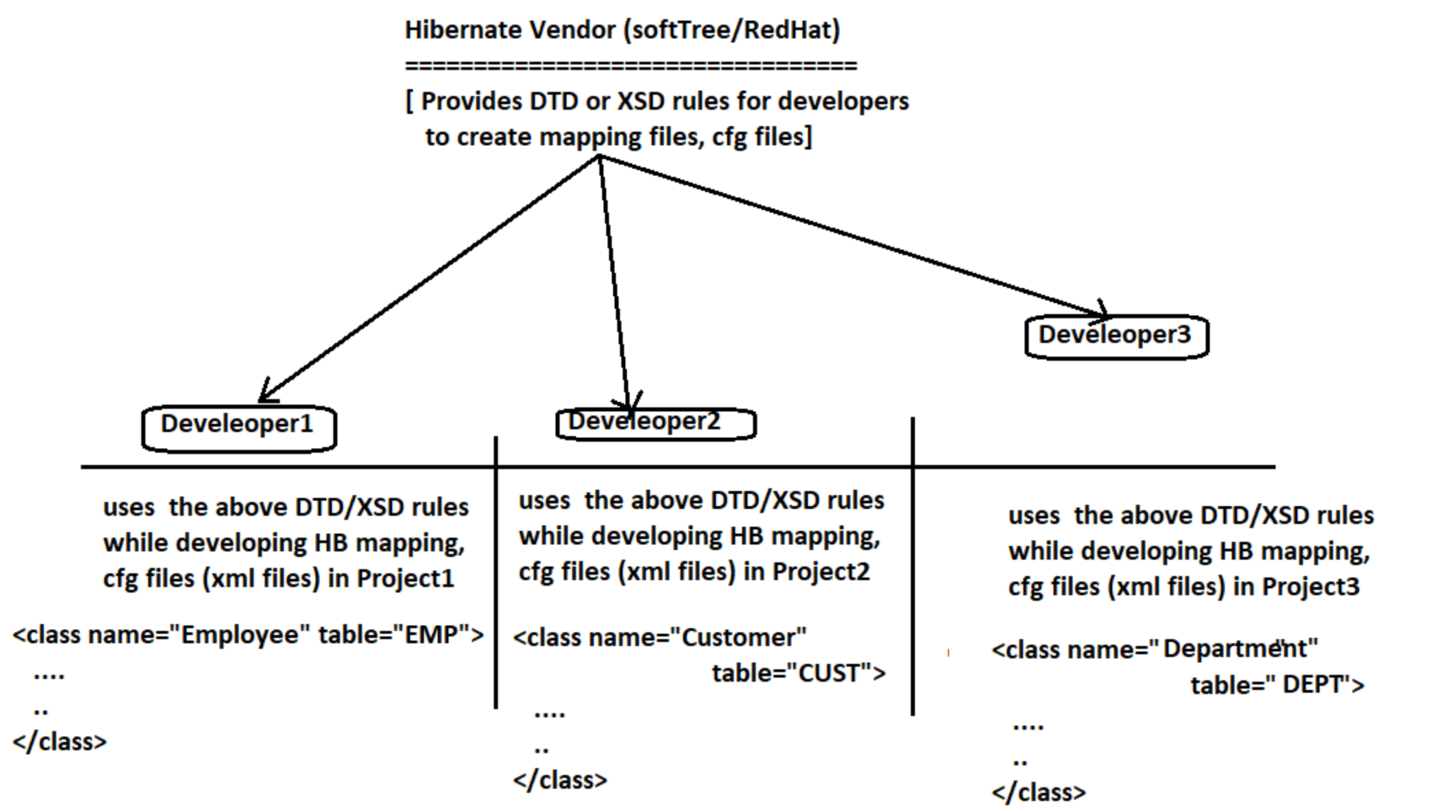
DTD :: Document Type Definitation (old style)
XSD :: Xml Schema Definitation (new style)

=>DTD rules will come in the form of .dtd file inside the jar files of Technology/framework apis
=>XSD rules will come in the form of .xsd file inside the jar files of Technology/framework apis

hibernate's mapping file(*.hbm.xml) DTD , XSD rules are coming in the form hibernate-mapping-3.0.DTD/ ,hibernate-mapping-4.0.xsd files in hibernate-core-<ver>.jar file (in org.hibernate pkg)

hibernate's cfg file(*.cfg.xml) DTD , XSD rules are coming in the form hibernate-configuration-3.0.DTD ,hibernate-configuration-4.0.xsd files in hibernate-core-<ver>.jar file (In org.hibernate pkg)

=> We can say XSD/DTD documents are the building blocks of the xml files i.e the xml files constructed by the using the tags and attributes of given XSD/DTD rules.



---

step11) Develope the hibernate cfg file

hibernate.cfg.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
 <session-factory>
  <!-- JDBC properties -->
   <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
   <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
   <property name="hibernate.connection.username">system</property>
   <property name="hibernate.connection.password">manager</property>

   <!-- mapping file name -->
   <mapping resource="com/nt/entity/Product.hbm.xml"/>
 </session-factory>

</hibernate-configuration>
```

xml prolog

importing DTD rules (collect from internet or form sample files)

hibernate configurations

The name and location of the hibernate mapping file

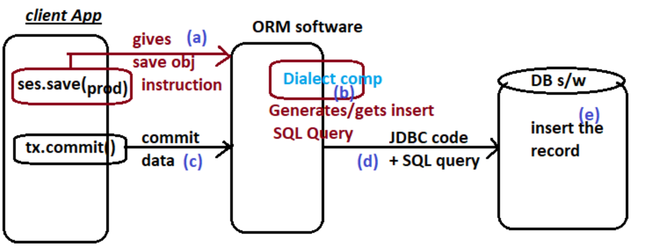
step12) Develop the Client App having Hibernate Persistence logic

=>For saving object we need to ses.save(-) method having the object of Entity class as the arg value
=> All non-select persistence operations in hibernate must be executed in Transactional env.. i.e
we should be in a position to commit or rollback persistence activity..

SaveObjectTest.java
==========

ctrl+shift+o :: importing java pkgs

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class SaveObjectTest {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        //specify the hibernate cfg file name and location
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
    //build SessionFactory having all services specified in cfg file and mapping file
        SessionFactory factory=cfg.buildSessionFactory();
        // create Session obj
        Session ses=factory.openSession();
        Transaction tx=null;
        try {
            // begin Transaction
              tx=ses.beginTransaction(); //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
                //prepare entity object
                  Product prod=new Product();
                  prod.setPid(1001);prod.setPname("table"); prod.setPrice(5678.5f);
                  prod.setQty(1.0f);
          //save object
                    ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
         tx.commit(); // internally calls con.commit() method to make insertion executin result perminent
         System.out.println("Object is saved(Record is inserted)");
        }
        catch(HibernateException he) {
            he.printStackTrace();
            tx.rollback(); //internally calls con.rollback() method to rollback the results of query execution.
            System.out.println("Object is not saved(Record is not inserted)");
        }

        //close Session obj
        ses.close();
        //close SesssionFactory obj
        factory.close();
    }//main
}//class
```



step13) Run the Client App

i) use Run button in the menu bar.

(or)

ii) Right click on client App -->run as ---> Java App

(or)

iii) ctrl +F11

In order see the Hibrnate f/w dialect comp generated SQL queries for the given persistence instructions on console of the app ,we can place the following instructions to hibernate from hibernate cfg file (xml)

In hibernate.cfg.xml

```
<!-- hibernate properties -->
  <property name="hibernate.show_sql">true</property>
  <property name="hibernate.format_sql">true</property>
```

=> The word hibernate is optional in the properties of hibernate cfg file

In hibernate.cfg.xml

```
<!-- JDBC properties -->
  <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
  <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
  <property name="connection.username">system</property>
  <property name="connection.password">manager</property>

  <!-- hibernate properties -->
  <property name="show_sql">true</property>
  <property name="format_sql">true</property>
```

note:: if u r using jdbc4 version based jdbc driver software which supports autloading of jdbc driver class then there is no need of placing

```
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
```
property

=> ojdbc6/7/8/...jar files are given based on JDBC4 version so specify driver class name while working with these jdbc drivers is purely optional.

signature of ses.save(-) method

Serializable save(Object object);

This mehtod generates id value , keeps in id property , gives save object persistence instruction to Hiberante f/w and returns the generated id value back to App.

note:: if no generator is cfg ... the value kept in identty property will be taken and will be return as the id value in form of Serializable obj

note:: All wrapper class objs,String objs are Serializable objs in Java by default Integer, Float , Long,String, Short and etc..

eg1:: With out generator cfg

```
//prepare entity object
        Product prod=new Product();
        prod.setPid(1352);prod.setPname("spoon"); prod.setPrice(35678.5f);
        prod.setQty(2.0f);
        //save object
      Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to
                                                 hibernate to save object (insert object data as the record)
  System.out.println("The Generated id value ::"+idVal);
```

Gives 1352 as id value in the form of Serialization obj

eg2:: With Generator cfg

In Mapping file (Product.hbm.xml)

```
<id name="pid" column="PID">  <!-- singular Id Field cfg -->
    <generator class="increment"/>
  </id>
```

uses max val + 1 formulae to generate the id value

In client app

```
//prepare entity object
        Product prod=new Product();
        prod.setPid(1352);prod.setPname("spoon"); prod.setPrice(35678.5f);
        prod.setQty(2.0f);
        //save object
      Integer idVal= (Integer) ses.save(prod); // Gives persistence
                                instruction to hibernate to save object (insert object data as the record)
  System.out.println("The Generated id value ::"+idVal);
```

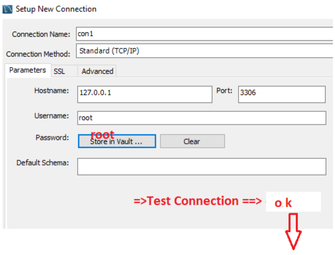
Let us assume max value in pk col of db table is 4561 then the "increment" generator generates 4561+1 as id value ,keep in id property returns to client App

Gives 4562 as the id value

Making HB first App interacting with mysql Db s/w

step1) make sure mysql latest version is installed in u r computer giving mysql work bench as built-in GUI DB tool

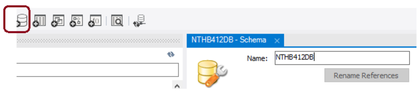
step2) lanuch mysql workbench connecting to mysql Db s/w



=>Test Connection ==> o k

Launch con1

step3) create Logical DB in mysql Db s/w

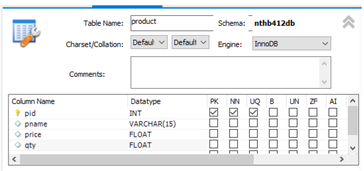


Logical DB is logical partiation of physical Db s/w which will be created on 1 per Logical DB basis.

-->apply --->apply ->finish

step4) create db table "product" in the above logic DB of mysql Db s/w

Expand NTHB412DB --->Tables ----> create table --->



-->apply ---> apply -->finish

step5) Gather mysql Connector/J JDBC driver details (jdbc driver for mysql)

jdbc driver class name :: com.mysql.cj.jdbc.Driver

jdbc url :: jdbc:mysql:///<logical DB> (for Local Mysql DB s/w)

jdbc:mysql://<host (or) ipadress>:<port>/<logical DB> (For remote mysql Db s/w)

step6) add mysql connector/j JDBc driver jar file (dependency) in pom.xml file of the Project

In pom.xml

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>8.0.27</version>
</dependency>
```

Collect the information from mvnrepository.com

step7) Modify the JDBC properties in hibernate.cfg.xml file

hibernate.cfg.xml

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <property name="connection.url">jdbc:mysql:///nthb412db</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>

    <!-- mapping file name -->
    <mapping resource="com/nt/entity/Product.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

step8) run the client App

=>if the Xml document /file is satisfying only syntax rules .. then it is called well-formed Xml document

Xml syntax rules

=>every open tag should have closing tag

=>The tags must be nested property

=>The tag , attribute names are case-sensitive

=>first tag of xml file/doc is called root tag/element.. once the root tag closed.. no other content should placed

and etc...

=>if the xml document is satisfying the imported DTD/XSD rules then it is called Valid Xml document

DTD:: Document Type Definitation

XSD :: Xml Schema Definitation

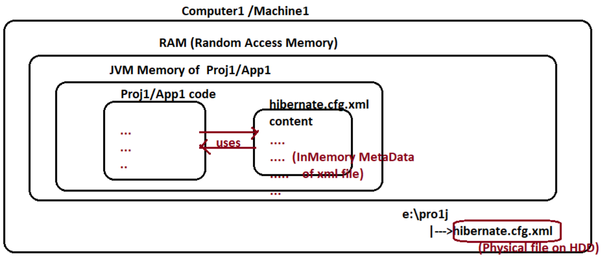
=> Xml parser is a software comp/App performing the following operations on given xml document/file

a) Loads the given xml file/doc from HDD

b) checks wellformed doc or not (if not throws exception)

c) checks valid doc or not (if not throws exception)

d) reads xml doc content and prepare Inmemory MetaData having xml file content in the JVM Memory of the RAM where current App is running for the faster reutlization of the content for multiple times



MetaData :: data about about (contains more details about existing details)

Example of Xml parser in java

=>SAX parser (Simple API for Xml Processing)

=>DOM parser (Document Object Model)

=>JDOM parser (Java Document Object Model)

=> DOM4J parser (DOM for Java)

and etc..

---

if Hibernate persistence logic is placed in the main(-) of standalone App then the flow begins with main(-) method and ends with main(-) method.

```
//Bootstarp / Activate the hibernate
Configuration cfg=new Configuration();
```

=>Activates /Bootstraps the hibernate framework (sofware) based on the HB jar files /libraries that are added to CLASSPATH/Build path.

```
//specify the hibernate cfg file name and location
cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
```

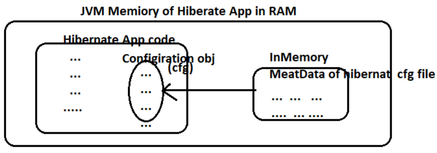
=>This line performs the following operations

a) Loads given hibernate cfg file from the specified location of the HDD

b) checks wheather hibernate cfg file(xml file) is well-formed or not and valid or not (if not exception will be thrown)

c) reads hibernate file content and prepare InMemory MetaData in the JVM memory of RAM belonging to current App (refer aabove)

d) stores the hibernate cfg file info and addtional info (MetaData) in the Configuration object




```
cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
System.out.println(cfg.getProperties());
```


{hibernate.format_sql=true, java.specification.version=16, sun.cpu.isalist=amd64, sun.jnu.encoding=Cp1252, sun.arch.data.model=64, java.vendor.url=https://java.orac
x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin, sun.java.command=com.nt.test.SaveObjectTest, jdk.c
2021-04-20, java.home=G:\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20
line.separator=
, java.vm.specification.vendor=Oracle Corporation, java.specification.name=Java Platform API Specification, user.script=, connection.password=root, sun.management.c
user.name=Nareshit, file.encoding=UTF-8, show_sql=true, java.io.tmpdir=C:\WINDOWS\TEMP\, java.version=16.0.1, connection.username=root, java.vm.specification.
\Softwares\Eclipse\eclipse-jee-2021-06-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205\jre\bin;C:\W
\WINDOWS;G:/Softwares/Eclipse/eclipse-jee-2021-06-R-win32-x86_64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-12
64/eclipse//plugins/org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_16.0.1.v20210528-1205/jre/bin;C:\Program Files\Java\jdk-11.0.3\bin;.;G:\Softwares\Eclip
sun.io.unicode.encoding=UnicodeLittle, java.class.path=G:\Worskpaces\Hibernate\NTHB412\HBProj01-saveObect\target\classes;C:\Users\Nareshit\.m2\repository\or
\Nareshit\.m2\repository\org\jboss\logging\jboss-logging\3.4.2.Final\jboss-logging-3.4.2.Final.jar;C:\Users\Nareshit\.m2\repository\javax\persistence\javax.persister
\bytebuddy\byte-buddy\1.11.20\byte-buddy-1.11.20.jar;C:\Users\Nareshit\.m2\repository\antlr\antlr\2.7.7\antlr-2.7.7.jar;C:\Users\Nareshit\.m2\repository\org\jbo
transaction-api_1.2_spec-1.1.1.Final.jar;C:\Users\Nareshit\.m2\repository\org\jboss\jandex\2.2.3.Final\jandex-2.2.3.Final.jar;C:\Users\Nareshit\.m2\repository\com\
\javax\activation\javax.activation-api\1.2.0\javax.activation-api-1.2.0.jar;C:\Users\Nareshit\.m2\repository\org\hibernate\common\hibernate-commons-annotations\
\repository\javax\xml\bind\jaxb-api\2.3.1\jaxb-api-2.3.1.jar;C:\Users\Nareshit\.m2\repository\org\glassfish\jaxb\jaxb-runtime\2.3.1\jaxb-runtime-2.3.1.jar;C:\Users\
\Nareshit\.m2\repository\com\sun\istack\istack-commons-runtime\3.0.7\istack-commons-runtime-3.0.7.jar;C:\Users\Nareshit\.m2\repository\org\jvnet\staxex\stax-
\FastInfoset\1.2.15\FastInfoset-1.2.15.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\ojdbc8\19.3.0.0\ojdbc8-19.3.0.0.jar;C:\Users\Nareshit\.m2\repositor
\repository\com\oracle\ojdbc\oraclepki\19.3.0.0\oraclepki-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\osdt_cert\19.3.0.0\osdt_cert-19.3.0.0.ja
\osdt_core-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracle\ojdbc\simplefan\19.3.0.0\simplefan-19.3.0.0.jar;C:\Users\Nareshit\.m2\repository\com\oracl
\mysql-connector-java\8.0.27\mysql-connector-java-8.0.27.jar;C:\Users\Nareshit\.m2\repository\com\google\protobuf\protobuf-java\3.11.4\protobuf-java-3.11.4.jar
java.vm.specification.version=16, os.name=Windows 10, sun.java.launcher=SUN_STANDARD, user.country=US, sun.cpu.endian=little, user.home=C:\Users\Nareshit, us
10.0, java.runtime.name=OpenJDK Runtime Environment, hibernate.connection.url=jdbc:mysql:///nthb412db, format_sql=true, java.vm.name=OpenJDK 64-Bit Server
java.vendor.url.bug=https://bugreport.java.com/bugreport/, user.dir=G:\Worskpaces\Hibernate\NTHB412\HBProj01-saveObect, os.arch=amd64, hibernate.connection
hibernate.bytecode.use_reflection_optimizer=false, connection.url=jdbc:mysql:///nthb412db, java.class.version=60.0}


```
cfg.configure();
```
(Not a good pratice)

This method takes hibernate.cfg.xml file from the CLASSPATH folder (In Maven project is src/main/java folder) and makes it as the hiberante cfg file.

note:: if u do not specify name and location of hibernate cfg file in the configure(-) method then it takes hiberante.cfg.xml file of classpath folder (src/main/java folder) as the default hibernate cfg file.. if the name or location of the hiberante cfg file is changed we must specify it explicity.. in the cfg.configure(-) method

```
cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
```
(good pratice)

note:: cfg.configure(-) does not attempt to load and read the contentn of hibernate mapping file thoguh their names are specified in hiberanat ecfg file

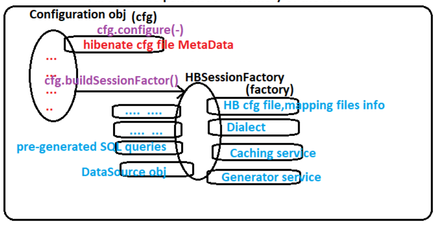
---

```
//build SessionFactory having all services specified in cfg file and mapping file
SessionFactory factory=cfg.buildSessionFactory();
```

The cfg.buildSessionFactory() method performs the following opeations

a) gets the name and location of hiberratne mapping file(s) from the configuration obj (Indirectly from hibernate cfg file) , Loads the mapping file(s), checks mapping file(s) are well-formed or not? , valid or not?) if not valid or if not well-formed they throw Exception) --> creates InMemory Meta data for mapping files -->Based on info placed in HB cfg file, HB Mapping files MetaData it will create following specialies services like dialect , datasource obj with jdbc con pool (Connection provider), Generator services , cache service, pre-generated SQL queries and etc.. ---> having all these services one heavy weight bigger object will be created that is SessionFactory object.

All these are part of JVM Memory of the RAM



Session ses=factory.openSession

=>get one jdbc con object from jdbc con pool , collects InMemoery meta data of mapping files and creates HB Session object having them , so we can HB session obj as con++

HB Session obj= jdbc con obj ++

=>SessionFactory obj is the object of Java class that implements SessionFactory(I) and that class will be given HB framework software (In the jar files Hibernate framework)

```
System.out.println("session factory obj class name::"+factory.getClass());
```

gives org.hibernate.internal.SessionFactoryImpl

SessionFactory(I)
↑ extends
SessionFactoryImplementor(I)
↑ implements
SessionFactoryImpl(c)

Session obj is the object of java class that implements Session(I) and that class will be given by HB framewokr software in the jar files represeting hibernate software.

```
System.out.println("Session obj class name::"+ses.getClass());
```

gives org.hibernate.internal.SessionImpl

Session(I)
↑ *extends*
SessionImplementor(I)
↑ *implements*
SessionImpl (c)

In eclipse IDE to open source code
=>For the class typed in editor :: f3
=>For the class not typed in editor :: ctrl +shit+ T -->type the class name
=> To get methods of class :: ctrl+o
=> To see the hierarchy of the class :: f4

Transaction tx=ses.beginTransaction();

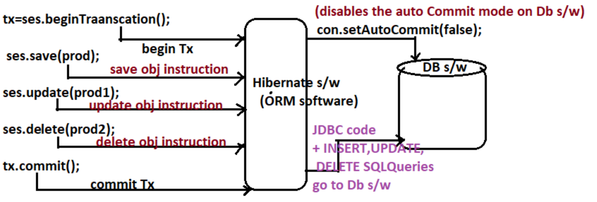
=>Intenrall calls con.setAutoCommit(false) method to disable autocommit mode on DB s/w to begin the Transaction.. In this process it returns Transaction obj reprpesenting the Trasaction that is started (Begining of the work)

Integer idVal= (Integer) ses.save(prod);

=>This method (ses.save(-)) takes the given Entity obj and passes to Hibernate f/w to provide persistence instruction of saving object .. In the process eigher value assigned property or the value generated by cfg generator will be returned back to Client App as the generrated id value in the form of Serilziable object

tx.commit();

=>Takes support dialect comp associated with Session obj either to generate dynamic insert SQL Query for save object persistence instruction or to use pre-generated inser SQL query , reads the given obj data to set to insert Query parameter values (?) and to send , execute the SQL query in Db s/w with the support of JDBC code.. At the end it calls con.commit() method to commit the INSERT SQL Query execution results



---

tx.rollback()

=>internally calls con.rollback() method and rollbacks the already executed SQL Query results.. (Hibernate programming calling this method is optional)

---

ses.close();

=>close the session by releasing the connection with Db s/w and vanishes the InMemory Metata and other resoruces that are associated with Session obj

factory.close();

=>Closes the SessionFactory object by releasing the associated services like jdbc con pool, generators, cachings and etc.. This method call also dectavities the Hibernate f/w that is in running mode..

When does the HB f/w will be activated and Deactivated ?

Ans) When Configuration obj created the Hiberante f/w will be bootstrapped (activated) and same hibernate f/w will be deactivated when SessionFactory obj is closed..

What is the difference b/w SessionFactory obj and Session obj?

SessionFactory obj	Session obj
a) It factory to create Session obj	a) It is mediator obj b/w Client App and Hibernate f/w to provide objects based persistence instructions to Hibernate f/w
b) It is heavy weight obj becoz it contains multiple other objects and services	b) It is light weight object just containing jdbc con object + few other objects
c) It is immutable obj	c) It is mutable object
d) It is thread safe object (immutable objs are thread safe by default)	d) It is not thread safe object (Mutable objs are not thred safe objs)
(e) It is long lived object	(e) It is short lived object
(f) It is generally taken on 1 per Db s/w basis	(f) It is generally taken on 1 per each persistence operation basis
g) It is implemented using "Factory Design Pattern"	g) No Design Pattern is involved
h) This object is created using Configuiration obj	h) This obj is created using SessionFactory obj
i) Contains pre-generated SQL Queries for Entiy classes based db tables that are specified in mapping files.	i) Does not contain any pre-generated SQL Queries .. But captable generated dynamic SQL queries as needed..
j) Contains Level2/SecondLevel Cache	j) Contains Level1/FirstLevel Cache
k)if Hiberante framework's SessionFactory obj is closed the actvation hiberaten framework will be stopped.	k) if HB session obj is closed , we can say , just connection with Db s/w will be closed.

ses.persist(-)
==========

=>This method is given to give save objet /persist object persistence instruction to Hibernate ORM software .. But can not use generators to generate the id value to identity property

=> This method is added to hibernate supporting JPA Specification .. but recomanded method to perform save object operation is ses.save(-) method becoz it can work generators to generate id value to identity property.

```
singnature ::
        public   void    persist(Object obj)
```

note:: ses.persist(-) method supports generators to generate id value only when if we do not set explicit value id property where as ses.save(-) method supports the generators to generate the id value in any situation (irresepective of wheather u set value to id property or not)

example code
==========

```
public class PersistObjectTest {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
    //build SessionFactory having all services specified in cfg file and  mapping file
        SessionFactory factory=cfg.buildSessionFactory();
        // create Session obj
        Session ses=factory.openSession();
        Transaction tx=null;
        try {
            // begin Transaction
            tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
            //prepare entity object
             Product prod=new Product();
             //prod.setPid(345);
             prod.setPname("spoon"); prod.setPrice(35678.5f);
             prod.setQty(2.0f);
        //save object
             ses.persist(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
        tx.commit(); // internally calls con.commit() method to make insertion execution result permineent
        System.out.println("Object is saved(Record is inserted)");
        }
        catch(HibernateException he) {
            he.printStackTrace();
            tx.rollback();  //internally calls con.rollback() method to rollback the results of  query execution.
            System.out.println("Object is not saved(Record is not inserted)");
        }

        //close Session obj
        ses.close();
        //close SesssionFactory obj
        factory.close();
    }//main
}//class
```

```
Product p=new Product();
p.setName("table");
p.setPrice("955");
p.setQty(4.5f);
ses.persist(p);      =>In this setup  it can work with cfg generators
```

```
Product p=new Product();
p.setPid(456);
p.setName("table");
p.setPrice("955");
p.setQty(4.5f);
ses.persist(p);      =>In this setup  it can not work with cfg generators
```

What is the difference b/w ses.save(-) and ses.pesist(-) method?

ses.save(-)	ses.persist(-)
(a) can work with the cfg generators to genearate id value to Id property in all situations	(a) can work with the cfg generators to genearate id value to the Id property only when values is not set id property..
(b) returns the generated value back to App after keeping in the id property becoz the return type Serializable	(b) Does not return the generated Id vlaue back to App after keeping the id property becoz the return type is void
(c) It is direct method in Hibernate Specification	(c) This method is placed hinberate by collecting from JPA Specification
(d) can override the existing value of id property when the generator generates the value	(d) can not override the existing value of id property when the generator is generated id vlaue .. Throw exception..
(e) This method based o-r mapping code is not portable across the multiple orm s/w.	(e) This based code works across the multiple ORM softwares..
(f) signature : public Serializable save(Object entity)	(f) signature : public void persist(Object entity)

Best of way of writing Client App code using try/catch and finally blocks
==========

SaveObjectTest.java
==========

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class SaveObjectTest {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        //specify the  hibernate cfg file name and location
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
    //build SessionFactory having all services specified in cfg file and  mapping file
        SessionFactory factory=cfg.buildSessionFactory();
        // create Session obj
        Session ses=factory.openSession();
        Transaction tx=null;
        boolean flag=false;
        try {
            // begin Transaction
            tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
             //prepare entity object
             Product prod=new Product();
             prod.setPid(1234);
             prod.setPname("spoon"); prod.setPrice(35678.5f);
             prod.setQty(2.0f);
        //save object
             Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
             System.out.println("The Generated id value ::"+idVal);
             flag=true;
        }
        catch(HibernateException he) {
            he.printStackTrace();
            flag=false;
        }
        finally {
            if(flag) {
                tx.commit();
                System.out.println("Object is saved");
            }
            else {
                tx.rollback();
                System.out.println("Object is not saved");
            }
            //close session obj
        try {
            if(ses!=null)
             ses.close();
            }
            catch(HibernateException he) {
                he.printStackTrace();
            }

            try {
                if(factory!=null)
                    factory.close();
            }
            catch(HibernateException he) {
                he.printStackTrace();
            }
        }//finally

note:: working with try,catch and finally blocks is legacy  model ..  if  the resources/objects like hibernate objs or
jdbc objs or  streams  are Autoclosable we need to close  them manually by  opening then with the support
of  try with Resource (TWR)

        syntax in java7/8
        ---------------------
        try( obj creation){
            ...
            ...
            }   // here the obj will be
        catch(....){        closed automatically
            ..
            ..
        }

        syntax in java9
        ---------------------
        object creation
        try(obj){
            ...
            ...
        }   // here the obj will be
        catch(....){        closed automatically
            ..
            ..
        }

        note: Hibernate's
        SessionFactory , Session objs are
        AutoClosable  obj becoz they are implementing
        java.lang.AutoClosable() directly or indirectly.

        while work with  Try with resource (TWR)  .. generally we do not need  finally block
```

Example Client App using try with resource syntax of java7/8
==========

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class SaveObjectTestUsingTWR1 {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        //specify the  hibernate cfg file name and location
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
    //build SessionFactory having all services specified in cfg file and  mapping file
        Transaction tx=null;
        try(SessionFactory factory=cfg.buildSessionFactory();
                                    // create Session obj
            Session ses=factory.openSession();){
            // begin Transaction
            tx=ses.beginTransaction();
             //prepare entity object
             Product prod=new Product();
             prod.setPid(1234);
             prod.setPname("spoon"); prod.setPrice(35678.5f);
             prod.setQty(2.0f);
        //save object
             Integer idVal= (Integer) ses.save(prod); // save obj
             System.out.println("The Generated id value ::"+idVal);
             tx.commit();
             System.out.println("Object is saved");
        } // The SessionFactory, SSession objs will be closed here automatically
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getRollbackOnly() && tx.getStatus()!=null) {
                tx.rollback();
                System.out.println("Obj is not saved");
            }
        }

    }//main
}//class...
```

Example Client App using try with resource syntax of java 9
==========

SaveObjectTestTWR2.java

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class SaveObjectTestUsingTWR2 {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        //specify the  hibernate cfg file name and location
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
    //build SessionFactory having all services specified in cfg file and  mapping file
        SessionFactory factory=cfg.buildSessionFactory();
        // create Session obj
    Session ses=factory.openSession();
        Transaction tx=null;
        try(factory;ses){   //form java 9
            // begin Transaction
            tx=ses.beginTransaction();
             //prepare entity object
             Product prod=new Product();
             prod.setPid(1234);
             prod.setPname("spoon"); prod.setPrice(35678.5f);
             prod.setQty(2.0f);
        //save object
             Integer idVal= (Integer) ses.save(prod); // save obj
             System.out.println("The Generated id value ::"+idVal);
             tx.commit();
             System.out.println("Object is saved");
        } // The SessionFactory, SSession objs will be closed here automatically
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getRollbackOnly() && tx.getStatus()!=null) {
                tx.rollback();
                System.out.println("Obj is not saved");
            }
        }

    }//main
}//class
```

Dialect comp /SQL Dialect comp
==========

=>This comp is capable of generating SQL Queries based on the Persistence instructions that are given
=> This comp is a class extneding from org.hibernate.dialect.Dialect(c) .. This class will be instantiated while building SessionFactory object and will be passed to Session object..
=>The Dialect comp either generates dynamic SQL Queries or uses the pre-generated SQL Query when we give persistence instructions ORM software (hibernate)

=>There are 2 operations performed by the Dialect comp
a) Generates optimized SQL Queries as needed to for the given persistence instruction
b) Assigns intelligent , sensible default values for the properties of Hibernate cfg which are not specified by the programmers based Db s/w and jdbc driver we are using

=> The Dialect comp is the class name whose name will change based on DB s/w and its version we use..

Cache71Dialect	Caché 2007.1 dialect.
DataDirectOracle9Dialect	
DB2390Dialect	An SQL dialect for DB2/390.
DB2400Dialect	An SQL dialect for DB2/400.
DB2Dialect	An SQL dialect for DB2.
DerbyDialect	Hibernate Dialect for Cloudscape 10 - aka Derby.
Dialect	Represents a dialect of SQL implemented by a particular RDBMS.
FirebirdDialect	An SQL dialect for Firebird.
FrontBaseDialect	An SQL Dialect for Frontbase.
H2Dialect	A dialect compatible with the H2 database.
HSQLDialect	An SQL dialect compatible with HSQLDB (HyperSQL).
Ingres10Dialect	A SQL dialect for Ingres 10 and later versions.
Ingres9Dialect	A SQL dialect for Ingres 9.3 and later versions.
IngresDialect	An SQL dialect for Ingres 9.2.
InterbaseDialect	An SQL dialect for Interbase.
JDataStoreDialect	A `Dialect` for JDataStore.
MckoiDialect	An SQL dialect compatible with Mckoi SQL
MimerSQLDialect	An Hibernate 3 SQL dialect for Mimer SQL.
MySQL5Dialect	An SQL dialect for MySQL 5.x specific features.
MySQL51InnoDBDialect	
MySQLDialect	An SQL dialect for MySQL (prior to 5.x).
MySQLInnoDBDialect	
MySQLMyISAMDialect	
Oracle10gDialect	A dialect specifically for use with Oracle 10g.
Oracle8iDialect	A dialect for Oracle 8i.
Oracle9Dialect	**Deprecated.** *Use either Oracle9iDialect or Oracle10gDialect instead*
Oracle9iDialect	A dialect for Oracle 9i databases.
OracleDialect	**Deprecated.** *Use OracleiDialect instead.*
PointbaseDialect	A `Dialect` for Pointbase.
PostgresPlusDialect	An SQL dialect for Postgres Plus
PointbaseDialect	A `Dialect` for Pointbase.
PostgresPlusDialect	An SQL dialect for Postgres Plus
PostgreSQLDialect	An SQL dialect for Postgres For discussion of BLOB "support" in postrges, as of 8.4, have a peek at http://jdbc.postgresql.org/documentation/84/
ProgressDialect	An SQL dialect compatible with Progress 9.1C Connection Parameters required: hibernate.dialect org.hibernate.sql.ProgressDialect hibernate.driver com.progress.sql.jdbc.JdbcProgressDriver hibernate.url jdbc:JdbcProgress:T:host:port:dbname;WorkArounds=53687 hibernate.username username hibernate.password password The WorkArounds parameter in the URL is required to a Progress 9.1C JDBC driver related to PreparedStatements.
RDMSOS2200Dialect	This is the Hibernate dialect for the Unisys 2200 Relational Database (RDMS).
ResultColumnReferenceStrategy	Defines how we need to reference columns in the group-by, having, and order-by clauses.
SAPDBDialect	An SQL dialect compatible with SAP DB.
SQLServer2008Dialect	A dialect for Microsoft SQL Server 2008 with JDBC Driver 3.0 and above
SQLServerDialect	A dialect for Microsoft SQL Server 2000 and 2005
Sybase11Dialect	A SQL dialect suitable for use with Sybase 11.9.2 (specifically: avoids ANSI JOIN syntax)
SybaseAnywhereDialect	SQL Dialect for Sybase Anywhere extending Sybase (Enterprise) Dialect (Tested on ASA 8.x)
SybaseASE15Dialect	An SQL dialect targetting Sybase Adaptive Server Enterprise (ASE) 15 and higher.
SybaseDialect	**Deprecated.** *use `AbstractTransactSQLDialect`, `SybaseASE15Dialect` or `SQLServerDialect` instead depending on need.*
TeradataDialect	A dialect for the Teradata database created by MCR as part of the dialect certification process.
TimesTenDialect	A SQL dialect for TimesTen 5.1.

Activate W
Go to Setting

java.lang.Object
org.hibernate.dialect.Dialect
org.hibernate.dialect.OracleBiDialect
org.hibernate.dialect.Oracle9iDialect
org.hibernate.dialect.Oracle10gDialect
org.hibernate.dialect.Oracle12cDialect
(for all latest versions of oracle)

java.lang.Object
org.hibernate.dialect.Dialect
org.hibernate.dialect.MySQLDialect
org.hibernate.dialect.MySQL5Dialect
org.hibernate.dialect.MySQL55Dialect
org.hibernate.dialect.MySQL57Dialect
org.hibernate.dialect.MySQL8Dialect
(for all latest version of mysql)

Current setup is oracle11g, so use org.hibernate.dialect.Oracle10gDialect

current setup for mysql Mysql 8 , so use org.hibernate.dialect.MySQL8Dialect

To cfg dialect in hibernate cfg file for oracle 11g DB s/w
In hibernate.cfg.xml

<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect </property> ,

To cfg dialect in hibernate cfg file for mysql 8.x DB s/w
In hibernate.cfg.xml

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect </property>

note:: Dialect cfg in hibernate cfg file is optional becoz it will be pickup automatically by Hibernate f/w based on the JDBC driver details that we place in hibernate cfg file

Conclusion:: Dialect Cfg in hibernate cfg file is optional but recomanded to cfg..

Two types of DB s/ws (Modren broad categorization)

SQL Db s/w (RDBMS DB s/w)

=> if data is structured i.e fixed no.of vlaues are possible in every record then go for SQL Db s/w
eg:oracle ,mysql, postgreSQL Db s/w

Employee db table with 10 colums
---
20 records as emp details haivng <=10 details always
(here all the records allocates memory for all the cols though we give less than 10 values)

Usecase :: Banking Apps, E-commerice Apps

No SQL Db s/w

=>if data is not structucted i.e there is no quarantee that similar infomation will be there in multiple records then go for NoSQL Db s/w

eg : MongoDB . HBase, Cassendra , Neo4J and etc..

Employee record1 with 10 details
Employee record2 with 5 details
Employee record3 with 20 details

There is no quantee that all these details will be having same fileds information

=> Each record allocates seperate amount of memory based no.of values that are in the record.

usecases:: orphan registration, Search criteria regsitration

note:: Hibernate is given only to inteact with RDBMs Db s/ws (SQL Db s/w) i.e it is not captable interacting with NoSQL DB s/w

Tommorrows timings:: 10:45am to 12:20/25 pm

In java app
=========

=>To place numeric value we have numeric data type variables like int,short,byte and etc.. data type variables.
=>To place floating values we have floating data type variables like float,double and etc.. data data type variables
=> To place text content ,we use String class objs
=> Similarly to hold class or interface or enum or annotation in the Java App use the object of of java.lang. Class

To create object java.lang.Class we can use one built-in property clalled ".class"

=>Two built-in properties of java App
.class (To create obj of java.lang.Class hold the class or interface or enum)
.length (To get the size of the array)

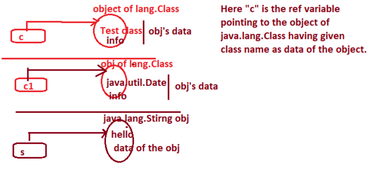
=>While compiling java class soruce code .. the Java compiler (javac) automatically many things to generated .class file .. In that list .class, .length properties will be there.

int type

ref variable of type java.lang.Class

Class c=Test.class;

Calss c1=java.util.Date.class;



Here "c" is the ref variable pointing to the object of java.lang.Class having given class name as data of the object.

String s=new String("hello");

=>Now onwords to pass class name or interface name or enum name or annotation name as inputs in the Java App we can place them in the object of java.lang.Class and we can use them.

Class c=Test.class;
String s1="Test";

What is the difference?
=>Here "s" is holding "Test" as the string value. So we can do only string manipulations on "Test"
=> Here "c" holding "Test" ass the real class name ,so using "c" we perform instantiation(obj creation) on "Test" class.

=>built-in properties of java App :: .class , .length
=>built-in threads of java App :: main , gc
=>built-in rer variables of java app :: this,super

Test.class
outside .java file
it is .class file

Inside Java app it is not .class file
it is passing "Test" class name in form of java.lang.Class obj's data

In feb month -- spring stack course -- 7 am

spring basics
spring boot
spring/spring all moules
microservices

Core, Jdbc
(pre-requistes)

Sunday class timing :: 10:45 am
(same link)

Q) How to perform partial Save object operation?

Ans) partial save obj means inserting values only for few columns outof multiple cols /all cols

(1) =>For this we should not take simple type properties in entity class
becoz simple data type properties contain default values 0 or 0.0 which will inserted to db table col when no values are set..

To solve this problem use wrapper data type properties in the Entity class whose default values are null i.e no value will be inserted in db table cols when values are not set.

note: null value in java is no value in db s/w cols
note:: null in java is presents one valid value..
null in Db s/w represents no value

```
//Good Designing
public class Product {
    private Integer pid;
    private String pname;
    private Float price;
    private Float qty;
}
```

```
// Bad Designing
public class Product {
    private Int r pid;
    private String pname;
    private float price;
    private float qty;
}
```

(2) set only few vlaues to the object of Entity class before ses.save(-) method

```
//prepare entity object
    Product prod=new Product();
    prod.setPid(1234);
    prod.setPname("spoon");
     //prod.setPrice(35678.5f);
   // prod.setQty(2.0f);
   //save object
      Integer idVal= (Integer) ses.save(prod); // save obj
```

=>contains null for price,qty properties if no values are set (for wrapper types)
=>contains 0.0 for price,qty properties if no values are set (for simple types)

The generated SQL query is ::

```
insert into PRODUCT (PNAME, PRICE, QTY, PID) values (?, ?, ?, ?)
```

Pre-generated SQL Queries
==========

=> In the process of creating SessionFactory object for all db tables specified in the mapping files the basic and regularly used SQL queries will be generated .by dialect as pre-generated SQL queries .. these will be used while completing persistence operations for tx.commit() call.

These pre-generated query are
->Insert SQL query inserting values to all the cols
->update SQL query updating all cols based on given id value as criteria value
->delete SQL query deleting the record based on given id value as criteria value
->Select SQL query selecting all col values of a record based on given id value as criteria value

=>Generally ses.save(-) method uses this pre-geneated query always to save the object..though we set less values to given entity object .. as shown above..

```
//prepare entity object
    Product prod=new Product();
    prod.setPid(1234);
    prod.setPname("spoon");
     //prod.setPrice(35678.5f);
   // prod.setQty(2.0f);
   //save object
      Integer idVal= (Integer) ses.save(prod); // save obj
```

=>contains null for price,qty properties if no values are set (for wrapper types)
=>contains 0.0 for price,qty properties if no values are set (for simple types)

The generated SQL query is ::

```
insert into PRODUCT (PNAME, PRICE, QTY, PID) values (?, ?, ?, ?)
```

=>To make the dialect comp to genrate dynamic insert SQL query only by involving those properties whose values non-null then we need to place dyamic-insert="true" in <class> tag of mapping file

```
<hibernate-mapping>
  <class name="com.nt.entity.Product" table="PRODUCT" dynamic-insert="true" >
   <!-- o-r mapping -->
   <id name="pid" column="PID">  <!-- singular Id Field cfg -->
     <generator class="increment"/>
   </id>
   <property name="pname" column="PNAME"/>
   <property name="price" column="PRICE"/>
   <property name="qty" column="QTY"/>
  </class>
</hibernate-mapping>
```

```
//prepare entity object
    Product prod=new Product();
    prod.setPid(1234);
    prod.setPname("spoon");
     //prod.setPrice(35678.5f);
   // prod.setQty(2.0f);
   //save object
      Integer idVal= (Integer) ses.save(prod); // save obj
```

=>contains null for price,qty properties if no values are set (for wrapper types)
=>contains 0.0 for price,qty properties if no values are set (for simple types)

The generated SQL query is ::

```
insert into PRODUCT (PNAME,PID) values (?, ? )
```

Configuring Lombok api with Eclipse IDE

=> It is third party generating commonly required basic code for java class dyamically with the support of annotations that are applied.

To cfg lombok api with Eclipse IDE
==========

step1) download lombok api.jar file from the internet
(https://projectlombok.org/download)

step2) run lombok.jar file and cfg with Eclipse IDE by supplying the name and location of eclipse IDE

Using Lombok api the Eclipse Project
==========

step1) make sure that lombok.api.jar file is added project built path
If is maven project add using pom.xml file

In pom.xml
=======

```
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.22</version>
  <scope>provided</scope>
</dependency>
```

step2) add lombok api annotations in required java class

```
@Getters
@Setters
@EqualsAndHashCode
@Data  (@Getter +@Setter + @ToString +@EqualsAndHashCode +.. )
@NoArgConstructor
@AllArgConstructor
@NonNull
@RequiredArgConsructor
@ToString
   and etc..
```

```
Example Entity class
==============
//Entity calss or BO class or Model class or Persistence class
package com.nt.entity;
import lombok.Data;
@Data
public class Product {
    private Integer pid;
    private String pname;
    private float price;
    private float qty;

}
```

@Data takes of generting code like seters ,getters, toString(), equalsAndHashcode(), no-param constructor dyamically for deleted , newly added , modified properties.

different ways to wrting hibernate mapping file
==========

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Product" table="PRODUCT" dynamic-insert="true" >
  <!-- o-r mapping -->
  <id name="pid" column="PID">  <!-- singular Id Field cfg -->
    <generator class="increment"/>
  </id>
  <property name="pname" column="PNAME"/>
  <property name="price" column="PRICE" />
  <property name="qty" column="QTY"/>
 </class>
</hibernate-mapping>
```

(or)

```
<hibernate-mapping>
  <class name="com.nt.entity.Product" table="PRODUCT" dynamic-insert="true" >
  <!-- o-r mapping-->
  <id name="pid" >
    <column name="PID"/>
    <generator class="increment"/> <!-- singular Id Field cfg -->
  </id>
  <property name="pname">
    <column name="PNAME"/>
  </property>
  <property name="price" >
    <column name="PRICE"/>
  </property>
  <property name="qty" >
    <column name="QTY"/>
  </property>
  </class>
</hibernate-mapping>
```

(or)

if the db table name is matching entity class name and db table col names are matching with the properties of entity class then specifying db table name and col names in the mapping file is completely optional

```
<hibernate-mapping>
  <class name="com.nt.entity.Product" dynamic-insert="true" >
    <!-- o-r mapping -->
    <id name="pid" >
      <generator class="increment"/> <!-- singular Id Field cfg -->
    </id>
    <property name="pname"/>
    <property name="price" />
    <property name="qty" />
  </class>
</hibernate-mapping>
```

(Best)

Loading object operation

=>This is nothing but selecting a record from db table based on the given id value and storing that record into Entity class object .

=>Loading object means loading the record into the object (Entity class obj)

=>For this operation hibernate uses the pre-generated SQL Query that select col vlaues of db table record based on the given id value.

select pid,pname,price,qty from Product where pid=?

=>Load object operation can be done in two ways
a) using ses.get(-,-) method [Performs eager/early Load object operation]
b) using ses.load(-,-) method [Performs lazy/late Load object operation by default
But can also perform earge/early loading if needed]

Loading object operation

=>This is nothing but selecting a record from db table based on the given id value and storing that record into Entity class object .

=>Loading object means loading the record into the object (Entity class obj)

=>For this operation hibernate uses the pre-generated SQL Query that select col vlaues of db table record based on the given id value.

select pid,pname,price,qty from Product where pid=?

=>Load object operation can be done in two ways

a) using ses.get(-,-) method [Performs eager/early Load object operation]

b) using ses.load(-,-) method [Performs lazy/late Load object operation by default But can also perform earge/early loading if needed]

Loading object using ses.get(-,-) method

signature of ses.get(-,-) method

```
public <T> T get(Class <T> clazz, Serilizable id)
```

type decl

return type

expecting class name in the form java.lang.Class obj

id value as the Serializable obj

=>All wrapper classes Serilziable classes i.e we can pass id vlaues as the Wrapper class obj..

=> Though id values as the simple values they will be converted wrapper objs using autoboxing concept.

Loading object operation

=>This is nothing but selecting a record from db table based on the given id value and storing that record into Entity class object .

=>Loading object means loading the record into the object (Entity class obj)

=>For this operation hibernate uses the pre-generated SQL Query that select col vlaues of db table record based on the given id value.

select pid,pname,price,qty from Product where pid=?

=>Load object operation can be done in two ways

a) using ses.get(-,-) method [Performs eager/early Load object operation]

b) using ses.load(-,-) method [Performs lazy/late Load object operation by default But can also perform earge/early loading if needed]

Loading object using ses.get(-,-) method

signature of ses.get(-,-) method

```
public <T>  T  get(Class <T> clazz, Serilizable id)
```

type decl — return type — expecting class name in the form java.lang.Class obj — id value as the Serializable obj

=>All wrapper classes Serilziable classes i.e we can pass id vlaues as the Wrapper class obj..
=> Though id values as the simple values they will be converted wrapper objs using autoboxing concept.

=>This method takes entity class and Serilizable id value as the parameters to get id value based record from db table and stores that record values in to object of Entity class by creating object for it.

=>This method returns null .. if given id value based record is not avaiable in the db table.

```
Product prod= ses.get(Product.class ,101);
  if(prod==null)
      S.o.println("record not found");
  else
    System.out.println(prod.getPid()+" "+prod.getPname()+" "+prod.getQty()+" "+prod.getPrice());
```

Hibernate app /Project development involving lombok api

==========

step1) configure lombok api with Eclipse IDE

=>download lombok-api.jar --> double click on the jar file --->select eclipse IDE installation location --->install/update.

step2) create Maven Project by taking maven-archetype-quickstart 1.4 as the archetype.

step3) add lombok api dependency in pom.xml file along with reguarl hibernate-core , ojdbc8 dependenceis

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.6.3.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.oracle.ojdbc/ojdbc8 -->
<dependency>
  <groupId>com.oracle.ojdbc</groupId>
  <artifactId>ojdbc8</artifactId>
  <version>19.3.0.0</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.22</version>
  <scope>provided</scope>
</dependency>
```

step4) develop resources of the application.

```
src/main/java
  |----> com.nt.cfgs
          |--->hibernate.cfg.xml   (same as first app)
  |--->com.nt.entity
          |---->Product.hbm.xml  (same as first app)
          |----->Product.java  (improve it by adding lombok annotations)

  |--->com.nt.test
          |----->LoadObjectTestingUsingGetMethod.java
```

Product.java (using lombok api)

```
package com.nt.entity;

import lombok.Data;

@Data
public class Product {
  private Integer pid;
  private String pname;
  private Float price;
  private Float qty;
}
```

```
//Client App
==========
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class LoadObjectTestUsingGetMethod {

    public static void main(String[] args) {
        //Configutration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        try(factory;ses){  //try with resource
            //Load object using get(-,-) method
            Product prod=ses.get(Product.class, 1452);
            if(prod==null)
                System.out.println("Product not found");
            /* else
                System.out.println(prod); */
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
        }

    }//main
}//class
```

Simple int to wrapper Integer obj to Serialzible obj

=>simple int to Wrapper obj (auto boxing)
note: All wrapper objs are Serializable objs

1) Select persistence operations in hibernate can be developed as non-transactional persistence operations becoz they do not modify db table data
2) if we pass reference variable in the S.o.p(-) method then toString() will be called automatically on that reference variable .. the Internal code is

3) ses.get(-,-) method performs eager Loading i.e one ses.get(-,-)is called the Hibernate F/w hits the Db s/w by generating or using select SQL Query and tries gets the record from DB and puts that record into an given entity class obj irrespective of wheather that object is used or not in further lines of coding.. (So it is called eager/early loading),

inside System.out.println(-) method

```
public void println(Object x) {
    String s = String.valueOf(x);
    if (getClass() == PrintStream.class) {
        // need to apply String.valueOf again since first invocation
        // might return null
        writeln(String.valueOf(s));
    } else {
        synchronized (this) {
            print(s);
            newLine();
        }
    }
}
```

System.out.println(prod); --> java code

F3

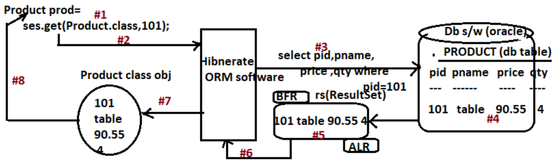
F3

```
public static String valueOf(Object obj) {
    return (obj == null) ? "null" : obj.toString();
}
```

Diagramatic representation of ses.get(-,-) method showing eager Loading

---

Client App
=========



Internal code of Ses.get(-,-) method
==========

```
public   <T> T  get(Class<T> clazz,  Serilziable id){

    try{
    PreparedStaement ps=
            con.prepareSatatement("SELECT PID,PNAME,PRICE,QTY WHERE   PID=?");
      ps.setInt(1,id);
    ResultSet rs=ps.executeQuery();
      if(rs.next()){
            Object obj=clazz.newInstance();
             Product prod=(Product) obj;
             prod.setPid(rs.getInt(1));
             prod.setPname(rs.getString(2));
             prod.setQty(rs.getFloat(3));
             prod.setPrice(rs.getFloat(4));
            return prod;
           }
     else
       return null;
     }
   }
   catch(Exception e){
      throw new HiernteException(e.getMessage());
      }
}
```

Exception rethrowing concept.

==========
working with ses.load(-,-) method
==========

=> this method can peform both eager loading and lazy loading of record

=>This method performs lazy loading by default..

=> When this method is performing lazy loading ..it first returns Proxy object (dummy object) that is created for InMemory Proxy class .. when start using proxy object then the code in the proxy class talks to Db s/w .. gets the record keeps in real object created for entity class and returns required data from real object

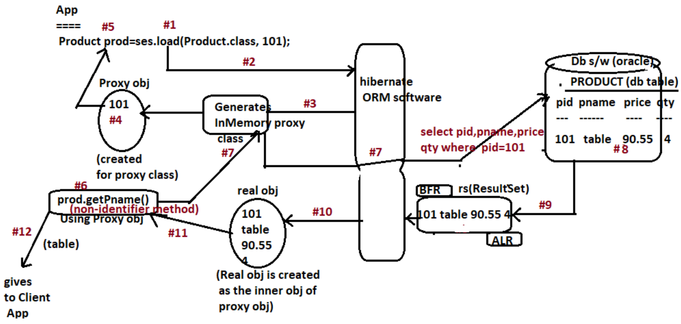
signaure of ses.load(-,-) method
---

public <T> T load(Class<T> clazz , Serializable id)

=>The dummy object that looks like real object and internally uses real object is called proxy object.

=>Working with proxy object gives the feling real object .. but it is different obj having addtional logics .. if needed proxy object internally uses real object.

=> ses.load(-,-) performs lazy loading only with the support of proxy object .. first returns proxy object when app show starts using proxy object then it creates real object having record collected from the db table.



=> ses.load(-,-) method generated InMemory Proxy class is always the sub class of Entity class

=> ses.load(-,-) always return Proxy object by creating proxy class irrespective of given id value based record is available or not in db table becoz first returns proxy object then it tries to hit the Db s/w when we invoke methods on Proxy object.

(other than id property methods like getPname(),getPrice(), getQty() ...)

=> when we call non-identifier methods on proxy object then only interaction Db s/w takes place by generating SQL select Query. In this process , if the record is found then creates real object having record inside the proxy obj otherwise it throws ObjectNotFoundException.

=>we can findout wheather record is available or not only through Exception handling..

Example App using ses.load(-,-) method
==========

```
package com.nt.test;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import com.nt.entity.Product;

public class LoadObjectTestUsingLoadMethod {

        public static void main(String[] args) {
                //Configutration obj
                Configuration cfg=new Configuration();
                cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
                //build SEssionFactory obj
                SessionFactory factory=cfg.buildSessionFactory();
                //build Session
                Session ses=factory.openSession();
                try(factory;ses){  //try with resource
                        //Load object using get(-,-) method
                        Product prod=ses.load(Product.class, 1422);
                        System.out.println(prod.getClass()+"   "+prod.getClass().getSuperclass());
                        System.out.println("--------------------");
                        System.out.println(prod.getPid()+"  "+prod.getPname()+"  "+prod.getPrice()+"  "+prod.getQty());
                }//try
                catch(HibernateException he) {
                        System.out.println("Record not found");
                        he.printStackTrace();
                }

        }//main
}//class
```

Normal java class

.java(source code) ----------> .class (byte code) ----------> JVM Loads the class ----------> JVM executes class
[HDD] [HDD] (JVM Memory of RAM) (JVM Memory of RAM)

InMemory Proxy class
==========

Source code Generation ----------> Source code compilation ----------> JVM Loading the class ----------> JVM executing the class
(JVM Memory of RAM) (JVM Memory of RAM) (JVM Memory of RAM) (JVM Memory of RAM)

=>For dynamic code generation the premgem memory of JVM memory will be used.

=>For dynamic code generation the premgem memory of JVM memory will be used.

==========
sample code of InMemory Proxy class (Generated for Product prod=ses.load(Product.class,101);
==========

```
public class  Product$abc145Proxy  extends  Product{
  private   Product  real;
  Object idVal;

    public  Product$abc145Proxy(){
      //Intialize idval property with the second
        arg gathered from the  ses.load(-,-) method  (101)

      ... .... ....
    }

  public  String getPname(){
     //logic  to record from DB table based on given id value
    PreparedStatement ps=con.prepareStatement("SELECT PID,PNAME,PRICE,QTY FROM PRODUCT
                                                                                    WHERE PID=?");
      //set arg value
    ps.setInt(1,idVal);
      //execute the Query
     ResultSet rs=ps.executeQuery();

     if(rs.next()){
          //real object creation
        real=(Product) Product.class.newinstance();
          //Maping RS record  real obj
          real.setPid(idVal);
          real.setPname(rs.getString(2));
          real.setPrice(rs.getFloat(3));
          real.setQty(rs.getFloat(4));
      }
     else
       throw new ObjectNotFoundException( idVal+" object is not found");

        return real.getPname();
    }

  public   String  getQty(){
     // get the record from DB based on id value and put into real obj or throw exception if
          record is not available
        ...
       ...
       return  real.getQty();

    }

  public  Float getPrice(){
     public   String  getQty(){
     // get the record from DB based on id value and put into real obj or throw exception if
          record is not available
        ...
       ...
     return real.getPrice();
    }

  public   Integer getPid(){
       return idVal;
    }

   ....
   ....
   ....
}  //class
```

=>This InMemory Proxy class is generated at run time --compiled at run time --executed at runtime in the JVM Memory of RAM

How to make ses.load(-,-) method performing eager loading of object
==========

Approach1) By taking lazy="false" in <class> of of in hibernate mapping file

```
<class name="com.nt.entity.Product"  table="PRODUCT"  lazy="false">
  ....
  ...
</class>
```

Approach2) *By making Entity class as final class*
(Final classes can not have sub classes .. So InMemory Proxy class will not generated for Entity class as the sub class ..due to this though we take lazy="true" the ses.load(-,-) performs eager loading)

```
@Data
public final class Product {
  private Integer pid;
  private String pname;
  private  Float price;
  private  Float qty;

  public Product() {
        System.out.println("Product:: 0-param constructor"+this.getClass());
  }

}
```

note:: the ses.get(-,-) always performs eager loading irrespective of wheather class is taken final or not and lazy="true" / lazy="false" is kept <class> or not

How can we make ses.load method(-,-) performing lazy loading though we have take entity class as the final class?

Ans) can be done with the support of proxy Interface .. i.e make ORM software to generate the proxy class by implementing the programmer supplied Interface (Proxy Interface) not by extending from Entity class.

step1) create ProxyInterface having getter and setter methods declaration based on Entity class properties.

IProduct.java

```
package com.nt.entity;

public interface IProduct {
      public Integer  getPid();
      public String  getPname();
      public  Float   getPrice();
      public Float getQty();
      public void  setPid(Integer pid);
      public void  setPname(String pname);
      public void   setPrice(Float price);
      public void setQty(Float qty);

}
```

Internally IProduct Interface will be implemented by the dynamically generated InMemory Proxy class.. So it is called Proxy Interface.

step2) Make Entity class implementing ProxyInterface

Product.java

```
package com.nt.entity;

import lombok.Data;

@Data
public final class Product implements IProduct{
  private Integer pid;
  private String pname;
  private  Float price;
  private  Float qty;

  public Product() {
        System.out.println("Product:: 0-param constructor"+this.getClass());
  }

}
```

step3) Specify the Proxy Interface name in the <class> tag of mapping file

```
<class name="com.nt.entity.Product"
        table="PRODUCT"  lazy="true"  proxy="com.nt.entity.IProduct">
  <id name="pid" column="PID" >
    <generator class="increment"/>
  </id>
  <property name="pname" column="PNAME"/>
  <property name="price" column="PRICE"/>
  <property name="qty"  column="QTY"/>
</class>
```

step4) Develop the Client App code as shown below

```
public class LoadObjectTestUsingLoadMethod {

      public static void main(String[] args) {
            //Configuration obj
            Configuration cfg=new Configuration();
            cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
            //build SEssionFacotry obj
            SessionFactory factory=cfg.buildSessionFactory();
            //build Session
            Session ses=factory.openSession();
            try(factory,ses){  //try with resource
                  //Load object using get(-,-) method
                  IProduct prod=ses.load(Product.class, 1452);
                  System.out.println(prod.getClass().getSuperclass()+"  "+Arrays.toString(prod.getClass().getInterfaces()));
                        System.out.println("-------------------");
                        System.out.println(prod.getPid()+"  "+prod.getPname()+"  "+prod.getPrice()+"  "+prod.getQty());
            }//try
            catch(HibernateException he) {
                  System.out.println("Record not found");
                  he.printStackTrace();
            }

      }//main
}//class
```

note:: once we specify proxy interface name in the proxy attribute of <class> tag ..the Hibernate makes ses.load(-,-) method to perform always lazy loading irrespective of lazy="true" or lazy="false" that is taken.

==========

Sample InMemory Proxy class code when it is implementing given Proxy Interface

==========

```
public class  Product$abc145Proxy   implements IProduct
  private  IProduct  real;
  Object idVal;

    public  Product$abc145Proxy(){
      //Intialize idval property with the second
        arg gathered from the  ses.load(-,-) method  (101)

      ... .... ....
    }

  public  String getPname(){
     //logic  to record from DB table based on given id value
    PreparedStatement ps=con.prepareStatement("SELECT PID,PNAME,PRICE,QTY FROM PRODUCT
                                                                                    WHERE PID=?");
      //set arg value
    ps.setInt(1,idVal);
      //execute the Query
     ResultSet rs=ps.executeQuery();

     if(rs.next()){
          //real object creation
        real=(Product) Product.class.newinstance();
          //Maping RS record  real obj
          real.setPid(idVal);
          real.setPname(rs.getString(2));
          real.setPrice(rs.getFloat(3));
          real.setQty(rs.getFloat(4));
      }
     else
       throw new ObjectNotFoundException( idVal+" object is not found");

        return real.getPname();
    }

  public   String  getQty(){
     // get the record from DB based on id value and put into real obj or throw exception if
          record is not available
        ...
       ...
       return  real.getQty();

    }

  public  Float getPrice(){
     public   String  getQty(){
     // get the record from DB based on id value and put into real obj or throw exception if
          record is not available
        ...
       ...
     return real.getPrice();
    }

  public   Integer getPid(){
       return idVal;
    }

   ....
   ....
   ....
}  //class
```

=>This InMemory Proxy class is generated at run time --compiled at run time --executed at runtime in the JVM Memory of RAM

What is the difference b/w ses.get(-,-) and ses.load(-,-) methods in Hibernate ?

ses.get(-,-)	ses.load(-,-)
a) Always perform eager load of record	a) can perform either eager loading or lazy loading of record
b) Uses only one object (real obj) of Entity class	b) Uses total two objects of Entity class (1 proxy object and 1 real obj)
c) Does not use any Design pattern internally	c) Uses PRoxy DesingPattern Internally (InMemory Proxy class generation)
d) Taking Entity class as final class does not create problem for this method	d) Taking Entity class as final class makes ses.load(-,-) method to perform eager loading or record becoz that kills the posibility of creating InMemory proxy class as sub class of entity class.
e) returns null if record is not available to load otherwise returns entity object having record	e) returns proxy object irrespective of wheather record is avaiable or not to load
f) This method can be used to check wheather is record is avaiable not (By check for null)	f) This method can not be to check wheather record is avaiable or not becoz it throws exception (ObjectNotFoundException) when non-indentifier methods are called proxy object if record is not avaiable.
g) taking lazy="true" or lazy="false" in <class> does not change the eager loading behaviour of ses.get(-,-) method	g) taking lazy="true" performs lazy loading and taking lazy="false" performs eager loading while working with ses.load(-,-) method.
h) No involvement of proxy Interface while working with this method	h) we can invole proxy interface to make ses.load(-,-) to generate the in InMemory class as impl class of given Proxy Interface
i) Useful in single layer apps .. where is record fetching and utialization takes place in the same layer (eager loading)	i) Useful in multiple layer app like MVC Apps where record will be fetched in one layer and will be utilized in another layer (lazy loading of record)

Upading the object /record

=>It is nothing but updating record the record db table by using the given Entity object.

It can be in two ways

a) Update object directly using ses.update(-) method (Useful for full object updation)

b) Load and update object using ses.load(-,-) and ses.update(-,-) methods (Useful for partial object updation)

a) Full object updation using ses.update(-) method directly

update(-) method signature

```
public void ses.update(Object obj)
```

```
public class FullObjectUpdateTest {

    public static void main(String[] args) {
        //Configutration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction tx=null;
        try(factory;ses){ //try with resource
            // begin Tx
            tx=ses.beginTransaction();
            //prepare object for Entity class
            Product p=new Product();
            p.setPid(1211); // must be existing id
            p.setPname("launge1"); //new value
            p.setPrice(4567.0f); //new value
            p.setQty(45.4f); //new vlaue
            //Instructio to update object
            ses.update(p);
        //Commit the Tx
            tx.commit();
            System.out.println("Full object is objected");
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not updated");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

Full object modification

makes hibernate to generate update query to update while whole record with given new values based on the given Id value.

```
Hibernate:
    update
        PRODUCT
    set
        PNAME=?,
        PRICE=?,
        QTY=?
    where
        PID=?
```

Limitations of this approach

=> can perform only full object updattion (complete record modification based on give id value)
i.e not suitable partial object modification

=> Can not check wheather updated or not becoz the the return of ses.udpdate(-) is void

=> if pass wrong id value for which record is not there in db table then we get exception

usecase for ses.update(-) method

update product screen

pid:: 1001
pname: table
price: 5000
qty : 30
update product

all these values must be entered by enduser to perform full object modification.

Paritial Object update using ses.load(-) and ses.update(-) method

```
package com.nt.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.nt.entity.Product;

public class PartialObjectUpdateTest {

    public static void main(String[] args) {
        //Configutration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction tx=null;
        try(factory;ses){ //try with resource
            //load object
            Product p=ses.get(Product.class, 1201);
            if(p==null) {
                System.out.println("record not found");
                return;
            }
            else {
            // begin Tx
            tx=ses.beginTransaction();
            //modify the object partially
            p.setPrice(12456.0f);
            ses.update(p);  //gives update obj insturction
        //Commit the Tx
            tx.commit();
            System.out.println("Parital object is updated");
            }//else
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not updated");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

generates select query + update queyr to perform load and update object operation

=>Here we can check weather record is availble or not before updaing ..So its very good partial update object update

=> This code basically generate update query like this though we are updating partially

```
update Product
    set pname=?,
        price=?,
        qty=?
    where pid=?
```

(Here ses.update(-) is using pre-generated update query)

=>To make hibernate f/w using dynamic update query by involving only required properties we need to place dynamic-update="true" in <class> tag.

```
<class name="com.nt.entity.Product" table="PRODUCT" dynamic-update="true">
```

Now the generate update SQL qyery is

```
update Product
    set , price=?
    where pid=?
```

usecase for load and update operation

products report page

101	table	10000	4	edit
102	softa	12000	3	edit
103	chair	11000	4	edit

product update page

pid 101 (editing is disbled)
pname table
price 10000 12945
qty : 4
update product

(partial modification)

=>Here load and update operation is good to use

=>For full object updation use ses.update(-) method ..
where as for partial object updation use ses.load(-,-) and ses.update(-) method

=> If we modify the obejct after loading it in a Transaction env.. the modifications will reflect to
underlying db table record when the transaction is committed. (i.e there is no need of placing ses.update(-) method)

PartialObjectUpdateTest1.java

```
package com.nt.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.nt.entity.Product;

public class PartialObjectUpdateTest1 {

    public static void main(String[] args) {
        //Configuratrion obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction tx=null;
        try(factory;ses){ //try with resource
            //load object
            Product prses.get(Product.class, 1001);
            if(p==null) {
                System.out.println("record not found");
                return;
            }
            else {
            // begin Tx
            tx=ses.beginTransaction();
              //modify the object partially
              p.setPrice(1566.0f);
    //Commit the Tx
                tx.commit();
                System.out.println("Parital object is  updated");
            }//else
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not updated");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

Here ses.update(-) is not called ... but any modification done to loaded in Transaction
will reflect db table row accordingly.. This is nothing but synchronization object to db table
row i.e the modifications done in object will reflect to db table row.

Q) How to perform synchronization b/w object to db table row?

Ans) =>For object to db table row synchorization Load the object and modify the object in a Transaction env..
(For example refer the above code)

=>For db table row to object synchroization call load object and call ses.refresh() to reload the
changes done in the db table row to object.
(For example refer the below code)

```
package com.nt.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.nt.entity.Product;

public class ObjectReloadTest {

    public static void main(String[] args) {
        //Configuratrion obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        try(factory;ses){  //try with resource
            //load object
            Product prses.get(Product.class, 1001);     //generates the select SQL Query
            if(p==null) {
                System.out.println("record not found");
                return;
            }
            else {
                //display record
                System.out.println(p);
                System.out.println("App going into sleep state.....");
                Thread.sleep(30000);  //sleep the app for 30 ms
                 // During this sleep time modify the db table record
                ses.refresh(p);  //reloads the object data from db table record  //regenerates the select SQL Query
                System.out.println(p);
            }//else
        }//try
        catch(Exception e) {
            e.printStackTrace();
        }//catch

    }//main
}//class
```

Deleting the object

=>It is all about deleting the record represented by the object based on the id value given the object.
=> In delete object operation , the object will not deleted .. rather record represented by the object will be
deleted i.e after deleting the record the object can be used for other persistence operations..

Delete object operation can be done in two ways
a) Directly by calling ses.delete(-) method
b) By loading and deleting object using ses.get(-,-) and ses.delete(-,-) methods

Signature of ses.delete(-) method
==========

```
public  void  delete(Object obj)
```

Approach1
==========

a) Directly by calling ses.delete(-) method (not recomanded)

=>delete operation is always complete record deletion
i.e there is no partial record deletion concept.

=>update operation can be full record updation
or partial record updation

=>select operation can be full record selection
or partial record selection

=>insert operation can be full record insertion
or partial record insertion

DeleteObjectTest.java

```
package com.nt.test;

import java.util.Arrays;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class DeleteObjectTest {

    public static void main(String[] args) {
        //Configuratrion obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction  tx=null;
        try(factory;ses){ //try with resource
            // begin Tx
            tx=ses.beginTransaction();
            //prepare object for Entity class
                Product p=new Product();
            p.setPid(1001);  // must be existing id
                //only  id property value is sufficient ... no need of
                // placing other property values
                //instruct to delete the object
                ses.delete(p);
//Commit the Tx
                tx.commit();
                System.out.println(" object is deleted");
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not deleted");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

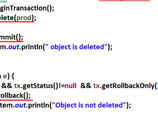
note:: This Approach always generate
just delete query to delete the record

Deletes the record
by generating Delete Query..

Limitions of this approach
==========
=> If record not there to delete .. it throws exception which can be caught and handle
=> we can not check wheather record deleted or not

To overcome these use Approach2.

usecase diagarm approach
==========



==========

Approach2 :: Deleting the record by loading the record (Recommanded approach)

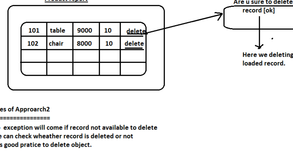
==========

```
package com.nt.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class LoadAnbDeleteObjectTest {

    public static void main(String[] args) {
        //Configuratrion obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction  tx=null;
        try(factory;ses){  //try with resource
            //load object
            Product  prod=ses.get(Product.class, 1002);
            if(prod==null){
                System.out.println("Record not found to delete");
                return;
            }
            else {
            // begin Tx
            tx=ses.beginTransaction();
                ses.delete(prod);
//Commit the Tx
                tx.commit();
                System.out.println(" object is deleted");
            }//else
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not deleted");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

Here delete and select query
will be generated

usecase of this approach

Product report



advantages of Approach2
==========
=> No exception will come if record not available to delete
=> We can check wheather record is deleted or not
=> It is good pratice to delete object.

note:: while working with single row persistence operation methods like ses.load(-,-)/ses.get(-,-) , ses.delete(-)
ses.update(-) and etc.. the persistence operations takes place by taking the id value as the criteria value.
if want to perform same operations using other criteria values . we need to for HQL , native SQL and other approaches..

saving or upding obejct using ses.saveOrUpdate(-)
==========

Ans) This method performs save object operation if record with given id value is not already availble in db table otherwise this method performs update object operation.

syntax : public void saveOrUpdate(Object obj)

This method can be used in two approaches

approach1) With out having generator , unsaved-value cfg
approach2) Having generator , un-saved value cfg

approach1) With out having generator , unsaved-value cfg (save or update obj operation)

=>Here generrates select +insert query while performing save object operation and generates select +update query while performing update object operation

=>If given object's id vlaue based record is avaiable in the db table then it performs update object operation (select + update query) otherwise it performs save obj operation (select + insert query)

```
// SaveOrUpdateObjectTest.java
package com.nt.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class SaveOrUpdateObjectTest {

    public static void main(String[] args) {
        //Configutration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction tx=null;
        try(factory;ses){ //try with resource
            // begin Tx
            tx=ses.beginTransaction();
              //prepare object for Entity class
              Product p=new Product();
              p.setPid(2001);
              p.setPname("softa");
              p.setPrice(19000.0f);
              p.setQty(25.0f);

              //save or update the obejct
              ses.saveOrUpdate(p);

    //Commit the Tx
              tx.commit();
              System.out.println(" object saved or updated");
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not saved or updated");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

usecase :: creating PF account it is not already there otherwise it will add update PF account with new company details.

Mapping file (Product.hbm.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

    <hibernate-mapping>
     <class name="com.nt.entity.Product" table="PRODUCT"
                                    dynamic-update="true"  dynamic-insert="true">
       <id name="pid" column="PID" />
       <property name="pname" column="PNAME"/>
       <property name="price" column="PRICE"/>
       <property name="qty" column="QTY"/>
     </class>
    </hibernate-mapping>
```

note:: The ses.saveOrUpdate(-) method supports both dyamic- insert and dyanmic-update query generations.

Q) Why dynamic -select , dynamic-delete Queries generation is not possible? (or) Why dynamic -select , dynamic-delete attributes are not in the <class> tag of mapping? given

Ans) Delete SQL Query deletes the complete record , not the partial record .. So dynamic-delete is not possible.
=> Load object operation takes id value as input value and creates the Entity object dynamically after select the record from db table ..Since it is not taking object as the input .. So we can say dynamic -select is not possible ..

=> In Load object operation the entity object will be created dynamically at runtime after selecting the record from Db table.(So dyanamic select query is not possible)

approach2) Having generator , un-saved value cfg (Recomanded) (save or update object operation)

case1:: id generator is cfg , unsaved-value is not mathching with given entity objects id property value then it perform update operation by directly generation update SQL Query having entity obj's id value as the criteria value.

mapping file
==========

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

    <hibernate-mapping>
     <class name="com.nt.entity.Product" table="PRODUCT"
                                    dynamic-update="true"  dynamic-insert="true" >
       <id name="pid" column="PID" unsaved-value="300" >
         <generator class= "increment"/>
       </id>
       <property name="pname" column="PNAME"/>
       <property name="price" column="PRICE"/>
       <property name="qty" column="QTY"/>
     </class>
    </hibernate-mapping>
```

Clent App
==========

```
// SaveOrUpdateObjectTest.java
package com.nt.test;


import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class SaveOrUpdateObjectTest1 {

    public static void main(String[] args) {
        //Configutration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        Transaction tx=null;
        try(factory;ses){ //try with resource
            // begin Tx
            tx=ses.beginTransaction();
              //prepare object for Entity class
              Product p=new Product();
              p.setPid(3001);
              p.setPname("chair");
              p.setPrice(13000.0f);
              p.setQty(15.0f);

              //save or update the obejct
              ses.saveOrUpdate(p);

    //Commit the Tx
              tx.commit();
              System.out.println(" object saved or updated");
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not saved or updated");
            }
            e.printStackTrace();
        }//catch

    }//main
}//class
```

case2:: id Generator is cfg , unsaved-value is matching with given entity obj's id property value then save object operation takes place by directly generating insert SQL Query ..This process the generator generated value will be taken as the id value.

case3: id generator is cfg , but unsaved-value is not cfg .. then ses.saveOrUpdate(-) method performs update object operation by directly generating update SQL Query taking entity obj's id value as the criteria value.

case4: id generator not cfg , but unsaved-value is cfg then ses.saveOrUpdate(-) method perform insert object operation if unsaved-value is matching id property value |if id property value is already there in db table then throws exception ,if inserts record) otherwise updates the record by taking id property value as the criteria value.

Why unsaved-value concept is given incase saveOrUpdate(-) method?

Ans) with out un-saved value , the ses.saveOrUpdate(-) performs select Query + insert/update Query generation.. here select query is purely extra query to check wheather record is avaialble not with given id vlaue of given object.
To make ses.saveOrUpdate(-) method directly generating update or inserting with out select query the unsaved-value concept is given.

usecase for Approach2 based ses.saveOrUpdate(-) method (same as approach1)

=>The Java class that contains all its methods as static methods is called utility class
=> Generally utility contains methods that are useful in multiple parts of the Project.
example of utility classes :: StringUtility , System , DriverManager, IOUtils, Collections and etc..

=> The block that executes automatically when JVM loads the class is called static block
=>In static block , we can accesss and use only static data... (variables) ..

=> The logic that creates Configuration object, Session Factory object, Session object is called Hibernate BootStrapping logic .. Instead of placing this logic in every Client App or in every DAO class it is recomanded to place in seperate "HibernateUtil.java" (Utility class)

=>static block is class level one time executing block ..where as constructor is object level one time executiing block

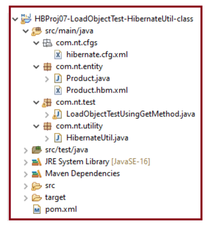
HibernateUtil.java

```
package com.nt.utility;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
        private static SessionFactory factory;
        static {
                System.out.println("HibernateUtil.static block");
                //create Configuration object
                Configuration cfg=new Configuration();
                //Load and read HB configuration file
                cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
                //build SessionFactory
                factory=cfg.buildSessionFactory();
        }

        //static factory method for SessionFactory object
        public static SessionFactory getSessionFactory() {
                        return factory;
        }

        //static factory method for session obj
        public static Session getSession() {
                        Session ses=null;
                        if(factory!=null)
                         ses=factory.openSession();
                        return ses;
        }

}
```

Client App

```
public class LoadObjectTestUsingGetMethod {

        public static void main(String[] args) {
                //get SessionFactory object
                SessionFactory factory=HibernateUtil.getSessionFactory();
                //get SEssion object
                Session ses=HibernateUtil.getSession();
                try(factory;ses){ //try with resource
                        //Load object using get(-,-) method
                        Product prod=ses.get(Product.class, 14334);
                                if(prod==null)
                                 System.out.println("Product not found");
                                else
                                 System.out.println(prod);
                }//try
                catch(HibernateException he) {
                        he.printStackTrace();
                }

        }//main
}//class
```

Using seperate HibernateUtil class for Bootstrapping.



==========
working with ses.merge(-) method (Merging object)
==========

=> this method can be used in two angels
a) Approach1:: As alternate for ses.saveOrUpdate(-) method
b) Approach2:: To perform Merge obejct operation

```
public Object merge(Object object)      throws HibernateException
```

a) Approach1:: As alternate for ses.saveOrUpdate(-) method

(Not recomanded use ses.saveOrUpdate(-) method directly)

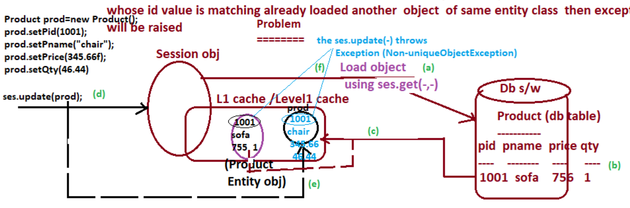
In Client App
==========

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class Apporach1MergeObjectTest {

        public static void main(String[] args) {
                //get SessionFactory object
                SessionFactory factory=HibernateUtil.getSessionFactory();
                //get SEssion object
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(factory;ses){ //try with resource
                        //being Transaction
                        tx=ses.beginTransaction();
                         Product prod=new Product();
                         prod.setPid(1001);
                         prod.setPname("table5");
                         prod.setPrice(456.0f);
                         prod.setQty(56.0f);
                         Product prod1=(Product) ses.merge(prod);
                         tx.commit();
                         System.out.println("Given obejct data "+prod+" hashCode::"+prod.hashCode());
                         System.out.println("Recived obejct data "+prod1+" hashCode::"+prod1.hashCode());
                         System.out.println("Object is saved or updated");
                }//try
                catch(HibernateException he) {
                        if(tx!=null || tx.getStatus()!=null || tx.getRollbackOnly()) {
                                tx.rollback();
                                System.out.println("Object is not saved or not updated");
                        }
                        he.printStackTrace();
                }

        }//main
}//class
```

Internally behaves likes ses.saveOrUpdate(-) method

b) Approach2:: To perform Merge obejct operation

=>Even Session object maintains one built-in cache/buffer memory called Level1/L1 cache to hold the given or loaded object of Entity class.
=> This L1 cache /Level1 cache can not maintain two objects of same entity class having same identity value .. if we force L1 cache to hold an of entity class object with certain id value whose id value is matching already loaded another object of same entity class then exception will be raised



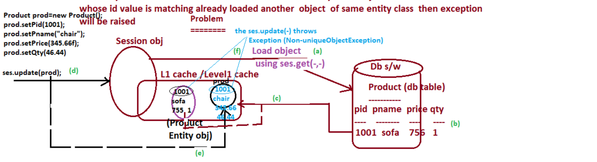
=> before tx.commit() method and after calling ses.save(-) or ses.update(-) or ses.delete(-) methods the given object of entity class will be stored in L1 cache/ Level1 cache..
=> The object for Entity class when ses.get(-,-) is called will also reside in L1 cache /Level1 cache..

=> if we try perform different peristnece operations using diffrent object of entity class with same identity value that will force the session obj two maintain two objects of same entity class with same id values.. That process throws Non-UniqueObject Exception.

b) Approach2:: To perform Merge obejct operation

=>Even Session object maintains one built-in cache/buffer memory called Level1/L1 cache to hold the given or loaded object of Entity class.

=> This L1 cache /Level1 cache can not maintain two objects of same entity class having same identity value .. if we force L1 cache to hold an of entity class object with certain id value whose id value is matching already loaded another object of same entity class then exception will be raised



=> before tx.commit() method and after calling ses.save(-) or ses.update(-) or ses.delete(-) methods the given object of entity class will be stored in L1 cache/ Level1 cache..

=> The object for Entity class when ses.get(-,-) is called will also reside in L1 cache /Level1 cache..

=> if we try perform different peristnece operations using diffrent object of entity class with same identity value that will force the session obj two maintain two objects of same entity class with same id values.. That process throws Non-UniqueObject Exception.

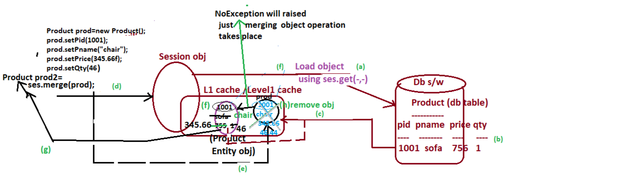
=>This is problem with ses.ses.update(-) , ses.saveOrUpdate(-) ,ses.save(-) ,ses.delete(-) i.e when these are called having entity object whose id value mataching already loaded entity object id value at L1 cache.. then the NonUniqueObjectException will be raised.

//Problem Code

```
public class Apporach2MergeObjectTestProblem {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get SEssion object
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        boolean flag=false;
        try{ //try with resource
            //being Transaction
            tx=ses.beginTransaction();

            //Load object Object
          Product prod=ses.get(Product.class, 3001);  //Loads Product class object(3001) and puts in L1 cache
           if(prod!=null)
            System.out.println(prod);

              //update object
              Product prod1=new Product();
              prod1.setPid(3001);
              prod1.setPname("goldspoon4");
              prod1.setPrice(4561.0f);
              prod1.setQty(56.0f);
              ses.save(prod1);
              //  ses.update(prod1); // tries to puts another object(3001) same Product class in L1 cache ,
                                     //So exception will be raised (NonUniqueObjectException)
              System.out.println("Object is  updated");
              flag=true;
        }//try
        catch(HibernateException he) {
            flag=false;
            he.printStackTrace();
        }//catch

        finally {
            if(flag)
                tx.commit();
            else
                tx.rollback();
          //close objs
            ses.close();
            factory.close();
        }
    }//main
}//class
```

Solution using ses.merge(-) method

In the above problem context if we call ses.merge(-) method having new object with already loaded object's id value then the changed data from new object will gathered, will be merged with already loaded object of L1/Level1 cache returns that update object. back to Client App. (with new/gives object data the already loaded old object's will be replaced/merged)



==========
ses.merge(prod) method performs
==========

a) Takes the given Entity object from ses.merge(-) argument
b) tries put that object into L1 cache by checking wheather already another object is there or not in L1 cahe for entity class with same id value

c) if avaiable
==========
Takes new object's data merges with extsting object data i.e replaces existing object's data with new object's data and returns existing object back to Application with chaged data.

if not avaialbe
==========
keeps the given object in L1 cache and performs insert object or update operation based given id value based record avaiabilty when the the tx.commit() is called.

lastly returns object represeting inserted/updated record.

Solution code
==========

```
public class Apporach2MergeObjectTestSolution {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get SEssion object
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        boolean flag=false;
        try{ //try with resource
            //being Transaction
            tx=ses.beginTransaction();

            //Load object Object
          Product prod=ses.get(Product.class, 3001);  //Loads Product class object(3001) and puts in L1 cache
           if(prod!=null)
            System.out.println(prod);

              //update object
              Product prod1=new Product();
              prod1.setPid(3002);
              prod1.setPname("SS poon1");
              prod1.setPrice(4561.0f);
              prod1.setQty(100.0f);

              Product prod2=(Product) ses.merge(prod1);
              System.out.println(prod2); displays load object with merged/replaced data

              System.out.println("Object is  merged");
              flag=true;
        }//try
        catch(HibernateException he) {
            flag=false;
            he.printStackTrace();
        }//catch

        finally {
            if(flag)
                tx.commit();
            else
                tx.rollback();
          //close objs
            ses.close();
            factory.close();
        }
    }//main
}//class
```

=>ses.merge(-) approach1 is alternate ses.saveOrUpdate(-) method .. not so useful where as ses.mege(-) apprache2 is very popular and industry standard (merging object operation).

ses.merge(-) approach1 and ses.saveOrUpdate(-) usecase :: creating new membership if it not already there .. if it is already avaible updates the existing membership detials

ses.merge(-) approach2(merging object) usecase :: In a Multithread env.. customer purchase(load object) , vendor product pricing updation(update object) activities are happening simulataenously then ses.merge(-) takes the new pricing details and merges already loaded obj.

b/w
What is the difference ses.update(-) and ses.merge(-) method?

ses.update(-) ==========	ses.merge(-) method ==========
a) just performs update object operation by generating update SQL Query	a) performs merge object operation (Approach2) using update SQL query or insert operation/update object operation(Approach2) by using SELECT SQL Query + insert/update SQL Query.
b) If the ses.update(-) method with Entity object whose id value is matching with already Loaded object's id value of same Entity class then we get NonUniqueObjectException	b) In the same context, no exception will be raised more over new obejct's data will be merged with existing obj's data.
c) Does not any thing becoz the return type type void public void update(Object obj)	d) Returns Entity object representing merged data (appraoch2) or representing saved object or upodated object becoz the return type java.lang.Object public Object merge(Object obj)

We can have two types of configurations in Hibernate programming

a) Xml driven cfgs (from hibernate 1.x onwords)
[The HB cfgs, mapping cfgs will be done using xml files]

b) Annotation driven cfgs (from hibernate 3.2 onwords)
[The HB cfgs are using xml files but mapping cfgs will be done using annotations]

b) Annotation driven cfgs (from hibernate 3.2 onwords)

=> Here HB mapping file (xml file) will be replaced with mapping annotations in Entity class itself.
=> As of now (Hibernate 5.6.x) there are no annotations replacing HB cfgfile ..
i.e we still need to use xml file for HB cfgs .. It is given like data to change Jdbc properties of Db s/w directly with flexibility.
=> Every Entity class with annotations must be mapped with Hibernate cfg file (hibernate.cfg.xml) using <mapping class="_____" /> attribute

<mapping resource="_____" /> for specifying HB mapping file
<mapping class="_____" /> for specifying Entity class with annotations

=>Working with mapping annotations in Entity class is very much standard..
=>If we provide both mapping annotations in entity class and HB mapping file (xml file) then the settings done in mapping file will override the settings done mapping annotations.

=> It is recomaded to use mapping annotations in Entity class in the following priority order
(a) JPA Annotations
(These annotations are portable across the multiple Entity classes of multiple ORM softwares)
(b) Hibernate specific annotations
(c) JSE , JEE annotations
(d) Third party annotations

=> While working with xml driven cfgs the comps are
a) HB cfg file (xml file) b) HB mapping file (xml file) c) Entity class(java) d) DAO class/Client App(java) — HibernateUtil class (helper class to Client App/DAO)
=> While working with annotation driven cfgs the comps are
a) HB cfg file (xml file) c) Entity class with mapping annotations(java) d) DAO class/Client App(java) — HibernateUtil class (helper class to Client App/DAO)

Basic Mapping annotations
==========

JPA annotations

@Entity --- To mark java class/Java bean as the entity class/Persistnece class (alternate to <class> tag of mapping file)
@Table --- To map entity class with db table (alternate to <class> tag of mapping file)
@Id --- To map the Entity class property as singular Identity property (alternate to <id> tag)
@Column -- To map the Entity class property with db table column (alternate to <property> tag)

note:: if the db table name is matching with entity class name then specifying table name @Table is optional.
note:: if the Entity class property name is matching with db table col name then spcifying col name in @Id , @Column annotations is optional

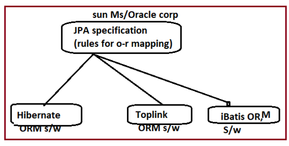
Example App
==========

=> create Maven Project using "maven-archetype-quickstart" as the archetype (change java version 10+ as needed through pom.xml)

=> add hibernate-core , lobmok, ojdbc8 depedency in pom.xml file

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.6.5.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.22</version>
  <scope>provided</scope>
</dependency>
```

```
<!-- https://mvnrepository.com/artifact/com.oracle.ojdbc/ojdbc8 -->
<dependency>
  <groupId>com.oracle.ojdbc</groupId>
  <artifactId>ojdbc8</artifactId>
  <version>19.3.0.0</version>
</dependency>
```

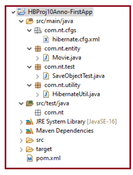


=> add the pkgs in src/main/java folder

=> develop Entity class/domain class/Model class/Persistence class with mapping annotations

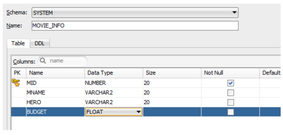
Movie.java

```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name="MOVIE_INFO")
@Data //lombok api
public class Movie {
    @Id //singular Id filed
    @Column(name="MID" ) //mapping with column
    private Integer mid;

    @Column(name="MNAME",length = 20) //mapping with column
    private String mname;

    @Column(name="HERO",length=20) //mapping with column
    private String hero;

    @Column(name="BUDGET") //mapping witu column
    private Float budget;
}
```



note:: we can apply mapping annotation either on fields or on getter methods i.e can not apply these mapping annotations on the top of setter methods.
note: Since lombok api does not allow u put getter methods explicitly .So we prefer placing the mapping annotation the fileds (properties/Member variables)

=> make sure MOVIE_INFO db table is avaiable in oracle Db s/w



=> Develop Hibernate cfg file (same as previous App .. but specify Entity class name is <mapping class="_____"/>)

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Connection properties -->
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">manager</property>

    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

    <!-- mapping file name -->
    <mapping class="com.nt.entity.Movie"/>
  </session-factory>
```

=> Develop the Client App having logic to save object (persistence logics devleopment in client Apps is same)

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Movie;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session object
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(ses;factory){
            //begin Tx
            tx=ses.beginTransaction();
            //prepare object
            Movie movie=new Movie(1101,"RRR","NTR-charan",400.0f);
            //save object
            ses.save(movie);
            tx.commit();
            System.out.println("Object is saved");
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not saved");
            }
            e.printStackTrace();
        }

    }//main
}//class
```

=> Develop HibernateUtil.java as Utility class (same as previous Apps)

HibernateUtil.java

```
package com.nt.utility;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static SessionFactory factory;
    static {
        System.out.println("HibernateUtil.static block");
        //create Configuration object
        Configuration cfg=new Configuration();
        //Load and read HB configuration file
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SessionFactory
        factory=cfg.buildSessionFactory();
    }

    //static factory method for SessionFactory object
    public static SessionFactory getSessionFactory() {
        return factory;
    }

    //static factory method for session obj
    public static Session getSession() {
        Session ses=null;
        if(factory!=null)
         ses=factory.openSession();
        return ses;
    }

}
```

More Annotations to use in annotation driven hibernate programming
==========

@Transient :: makes certain property not participating in any kind of Persistence operations

```
@Entity
@Table(name="MOVIE_INFO")
@Data //lombok api
@AllArgsConstructor
@NoArgsConstructor
public class Movie {
    @Id //singular Id filed
    @Column(name="MID" ) //mapping with column
    private Integer mid;

    @Column(name="MNAME",length = 20) //mapping with column
    private String mname;

    @Column(name="HERO",length=20) //mapping with column
    private String hero;

    @Column(name="BUDGET")
    @Transient
    private Float budget;
}
```

In LoadObjectTest.java

```
try(ses;factory){
        Movie movie=ses.get(Movie.class, 1101);
        if(movie==null)
            System.out.println("Movie not found");
        else
            System.out.println(movie);
    }//try
```

```
select
    movie0_.MID as mid1_0_0_,
    movie0_.HERO as hero2_0_0_,
    movie0_.MNAME as mname3_0_0_
  from
    MOVIE_INFO movie0_
  where
    movie0_.MID=?
Movie(mid=1101, mname=RRR, hero=NTR-charan, budget=null)
```

(The @Transient property budeget not participated in persistence operation)

SaveObjectTest.java

```
Transaction tx=null;
    try(ses;factory){
        //begin Tx
        tx=ses.beginTransaction();
        //prepare object
        Movie movie=new Movie(1102,"RadheShyam","prabhas",200.0f);
        //save object
        ses.save(movie);
        tx.commit();
        System.out.println("Object is saved");
    }//try
```

Generated INSERT SQL Query :: INSERT INTO MOVIE_INFO (MID,MNAME,HERO) VALUES (?,?,?)
(The @Transient property budeget not participated in persistence operation)

To make only non-null properties related col involving in the insert, update Query Generation we need to use @DynamicInsert, @DynamicUpdate at entity class level as shown below.

```
@Entity
@Table(name="MOVIE_INFO")
@Data //lombok api
@AllArgsConstructor
@NoArgsConstructor
@DynamicInsert(value = true)
public class Movie {
    @Id //singular Id filed
    @Column(name="MID" ) //mapping with column
    private Integer mid;

    @Column(name="MNAME",length = 20) //mapping with column
    private String mname;

    @Column(name="HERO",length=20) //mapping with column
    private String hero;

    @Column(name="BUDGET")
    private Float budget;
}
```

SaveObjectTest.java

```
Transaction tx=null;
    try(ses;factory){
        //begin Tx
        tx=ses.beginTransaction();
        //prepare object
        Movie movie=new Movie();
        movie.setMid(1201);
        movie.setMname("Pupsa");
        //save object
        ses.save(movie);
        tx.commit();
        System.out.println("Object is saved");
    }//try
```

Assignment :: write example for dynamic update generation

Generated insert SQL query ::

```
Hibernate:
  insert
  into
    MOVIE_INFO
    (MNAME, MID)
  values
    (?, ?)
Object is saved
```

What is the differnce b/w @Transient and @DyanamicInsert Annotation?

Ans) @Transient is field level annotation and makes the property of Entity class obj not participating in any persistence operation irrespective of wheather u have kept data for it or not.

Ans) @DynamicInsert is class level annotation and makes hibernate F/w to generate insert SQL Query dynamically only by using properties of entity class object for which non-null data is kept.

note:: DyanamicUpdate is similar to @DynamicInsert towords dynamic update SQL Query generation for update object operation.

Q) How enable or disable lazy loading for ses.load(-,-) method while working annotation driven cfgs ?

Place @Proxy(lazy=false) to disable lazy loading for ses.load(-,-) method i.e indirectly we are looking to enable eager loading of object.

```
@Entity
@Table(name="MOVIE_INFO")
@Data //lombok api
@AllArgsConstructor
@NoArgsConstructor
@DynamicInsert(value = true)
@Proxy(lazy = false)
public class Movie {
    @Id //singular Id filed
    @Column(name="MID" ) //mapping with column
    private Integer mid;

    @Column(name="MNAME",length = 20) //mapping with column
    private String mname;

    @Column(name="HERO",length=20) //mapping with column
    private String hero;

    @Column(name="BUDGET")
    private Float budget;
}
```

enables the eager loading

LoadObjectTest.java

```
try(ses;factory){
    Movie movie=ses.load(Movie.class, 1101);
    //System.out.println(movie);
    }//try
    catch(Exception e) {
        e.printStackTrace();
    }
```

It hits the Db s/w irrespective of wheather we are using loaded object or not

How to perform lazy loading ses.load(-,-) method though we have entity class as final class?

Ans) Possible by making HB f/w to generate the InMemory Proxy class as the impl class for the given Proxy Interface we need to specify this ProxyInterface with the support of @Proxy annotation.

step1) Design Proxy Interface having getter methods based on the Entity class properties

```
package com.nt.proxy;

public interface IMovie {
    public Integer getMid();
    public String getMname();
    public String getHero();
    public Float getBudget();
}
```

step2) make the Entity class implementing Proxy Interface *and also specify the Proxy Interface name*

Movie.java (Entity class)
==========

```
@Entity
@Table(name="MOVIE_INFO")
@Data //lombok api
@AllArgsConstructor
@Proxy(lazy = true,proxyClass = com.nt.proxy.IMovie.class)
public final class Movie implements IMovie {
    @Id //singular Id filed
    @Column(name="MID" ) //mapping with column
    private Integer mid;

    @Column(name="MNAME",length = 20) //mapping with column
    private String mname;

    @Column(name="HERO",length=20) //mapping with column
    private String hero;

    @Column(name="BUDGET")
    private Float budget;

    public Movie() {
        System.out.println("Movie:: 0-param constructor"+this.getClass());
    }

}
```

step3) Develop the Client App as shown below

```
package com.nt.test;

import java.util.Arrays;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Movie;
import com.nt.proxy.IMovie;
import com.nt.utility.HibernateUtil;

public class LoadObjectTestUsingLoadMethod {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session object
        Session ses=HibernateUtil.getSession();
        try(ses;factory){
            IMovie movie=ses.load(IMovie.class, 1101);
            System.out.println(movie.getClass()+" <-------> "+movie.getClass().getSuperclass()+" <---> "+Arrays.toString(movie.getClass().getInterfaces()));
                //System.out.println(movie);
        }//try
        catch(Exception e) {
            e.printStackTrace();
        }

    }//main
}//class
```

gives java.lang.Object (c)

implements given PRoxy Interface

Conclusion::

@Transient :: makes the proeprty as transient property (Non persistable property)
@DynamicInsert :: To generate insert SQL queery dynamically
@DynamicUpdate :: To generate update SQL queery dynamically
@Proxy :: To enable or disable lazy loading ses.load(-,-) and also to generate Proxy class of Lazy Loading by implementing given Proxy Interface.

==========
hibernate.hbm2ddl.auto property
==========

(DyamicShema Geration)

=> This is HB cfg file property which gives instruction to HB f/w to generate schema (db tables, sequences and etc..) based on the given mapping files information..

=> This property gives shema generation ,updation instructions to hibernate f/w ..

=>The possible values for this property are
- validate (default)
- create
- update (recomanded value)
- create-drop

hbm2ddl :: HB mapping to Data definittion Language
(means performing DDL Queries generation and execution based on the mapping file instructions)

======
create
======

=> always create db tables , sequences and etc. based on mapping file configurations.. if they are already avaiable then they will be deleted.

=> this value for "hbm2ddl.auto" property makes the App to drop the existing db tables and others like sequences as per mapping file cfgs.. and creates them for every exeution..

example
=========
In hibenrate cfg file *(hibernate.cfg.xml)*

```
<property name="hbm2ddl.auto">create</property>
```

In mapping file (Product.hbm.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Product" table="PRODUCT">
    <!-- o-r mapping -->
    <id name="pid" column="PID" length="5">   <!-- singular Id Field cfg -->
      <generator class="increment"/>
    </id>
    <property name="pname" column="PNAME" length="15" not-null="true" unique="true"/>
    <property name="price" column="PRICE"/>
    <property name="qty" column="QTY"/>
  </class>
</hibernate-mapping>
```

=>Only for String type columns we can control the length using "length" attribute.. for Non-String cols we can not controll length.. they will be taking the dfault length.

=> not-null="true" , unique="true" can be applied on both numeric and non-numeric properties/columns

=>In Annotation driven setup we can specify these attributes usng length, nullable , unique params of @Column annotation.

```
@Column(name="PNAME",length = 20 ,unique = true, nullable = false)
private Stirng pname;
```

Client App
==========

```
public class SaveObjectTest {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        //specify the hibernate cfg file name and location
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
  //build SessionFactory having all services specified in cfg file and mapping file
        SessionFactory factory=cfg.buildSessionFactory();
        // create Session obj
        Session ses=factory.openSession();
        Transaction tx=null;
        try {
            // begin Transaction
              tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
                //prepare entity object
                Product prod=new Product();
                prod.setPid(1234);
                prod.setPname("spoon"); prod.setPrice(35678.5f);
                prod.setQty(2.0f);
        //save object
                Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
                System.out.println("The Generated id value ::"+idVal);
      tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
       System.out.println("Object is saved(Record is inserted)");
        }
        catch(HibernateException he) {
            he.printStackTrace();
            tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
            System.out.println("Object is not saved(Record is not inserted)");
        }

        //close Session obj
        ses.close();
        //close SesssionFactory obj
        factory.close();
    }//main
}//class
```

========
update *(Mostly used value for hbm2ddl.auto property)*
========

=> It is multiple pupose value
- It creates new db tables if they are not already avaiable
- it alters the existing db tables if they are already avaible but not matching with mapping files cfg (only adding new cols to existing db tables is allowed)
- It uses the existing db tables if they are alrady avaiable and matching with db mapping files cfg

In hibernate.cfg.xml

```
<property name="hbm2ddl.auto">update</property>
```

To make this "udpate" value adding new columns to db table by altering db table we need to perform following operations

(a) add extra property in the Entity class and generate setter and getter methods for it.

In Product.java

```
private String status;

public void setStatus(String status){
  this.status=status;
}
public String getStatus(){
  return status;
}
```

*** altering table is possiblity for "update" value only towards adding the columns , not towards deleting the cols or ,updating the cols.*

(b) cfg the "status" property in the mapping file

```
<property name="status" column="STATUS" length="20"/>
```

(c) Run the Client app

========
validate
========

=> verifies/ validates the db table and entity class as per hb mapping file configurations.. if not there exception will be thrown .. otherwise nothing takes place.. the activity will go on...

eg: In hiberate.cfg.xml

```
<property name="hbm2ddl.auto">validate</property>
```

==========
create-drop
==========

=>creates the db tables during the application startup i.e when the SEssionFactory obj is created .. and drop the same db tables during the Application shutdown i.e when the SEssionFactory object is closed.

=>This value is very useful in the testing phase ,UAT phase , Demo phase of the project.. where db tables will be created for the Demo , will be used during the demo and will be destroyed at the end of the demo.

UAT ---> User Acceptence Test.

eg: In hibernate.cfg.xml

```
<property name="hbm2ddl.auto">create-drop</property>
```

client App
========

```
public class SaveObjectTest {

    public static void main(String[] args) {
        //Bootstarp / Activate the hibernate
        Configuration cfg=new Configuration();
        //specify the hibernate cfg file name and location
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
  //build SessionFactory having all services specified in cfg file and mapping file
        SessionFactory factory=cfg.buildSessionFactory();
        System.out.println(" Session factory obj is ready ...");
        // create Session obj
        Session ses=factory.openSession();
        Transaction tx=null;
        try {
            // begin Transaction
              tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
                //prepare entity object
                Product prod=new Product();
                prod.setPid(1234);
                prod.setPname("spoo34"); prod.setPrice(35678.5f);
                prod.setQty(2.0f);

                Thread.sleep(30000);
        //save object
                Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
                System.out.println("The Generated id value ::"+idVal);
      tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
       System.out.println("Object is saved(Record is inserted)");
       Thread.sleep(30000);
        }
        catch(HibernateException he) {
            he.printStackTrace();
            tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
            System.out.println("Object is not saved(Record is not inserted)");
        }
        catch(Exception e) {
            e.printStackTrace();
        }

        //close Session obj
        ses.close();
        //close SesssionFactory obj
        factory.close();
        System.out.println(" Session factory obj is closed ...");
    }//main
}//class
```

creates the db tables

uses the db tables

uses the db tables

drops the db tables

==========
Versoning in hibenrate
==========

=>This feature keeps track of how many times the object/record is modified through Hibernate Persistence logic.

=>Using this feature we can keep track of how many times the record is been modified and this tracking helps in certain use-cases

a) To charge money caller tune change

b) To charge money to change phone number inm bank account for than 3 times

c) To keep track of how many times ATM service is used in a month and etc..

note:: This Versinong also useful towards Locking Object concept (future topic)

=> For versoning implementation in hibernate

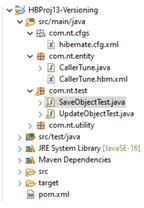
a) we need to take the property of numberic type like int , long, short or wrapper types in Entity class

b) Configure that property in using <version> tag in mappin file (xml cfgs) or using @Version (using annotation cfgs)

c) Write two Client Apps

i) To save object

ii) To load and modify object



Entity class

```
package com.nt.entity;

import lombok.Data;

@Data
public class CallerTune {
	private Integer tuneId;
	private String  tuneName;
	private  String movieName;
	private  Integer updationCount;  //for Versioning

}
```

HB cfg file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Connection properties -->
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">manager</property>

    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>


    <!-- mapping file name -->
    <mapping resource="com/nt/entity/CallerTune.hbm.xml"/>
  </session-factory>

</hibernate-configuration>
```

HB mapping file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

	<hibernate-mapping>
	 <class name="com.nt.entity.CallerTune"  table="CALLERTUNE">
	   <id name="tuneId" column="TUNEID" >
	     <generator class="increment"/>
	   </id>

	   <version name="updationCount" column="UPDATIONCOUNT" />

	   <property name="tuneName" column="TUNENAME"  length="20"/>
	   <property name="movieName" column="MOVIENAME" length="20"/>
	 </class>
	</hibernate-mapping>
```

must be placed after <id> tag

SaveObjectTest.java

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.CallerTune;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

	public static void main(String[] args) {
		//get SessionFactory object
		SessionFactory factory=HibernateUtil.getSessionFactory();
		//get Session object
		Session ses=factory.openSession();
		Transaction tx=null;
		try(factory;ses){
			//begin Tx
			tx=ses.beginTransaction();
			  //prepare object
			  CallerTune tune=new CallerTune();
			  tune.setTuneName("srivalli");
			  tune.setMovieName("pushpa- The raise");
			  //save object
			  ses.save(tune);
			 tx.commit();
			 System.out.println("Object is saved");
		}//try
		catch(Exception e) {
			if(tx!=null || tx.getStatus()!=null || tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Object is not saved");
			}//if
		}//catch
```

UpdateObjectTest.java

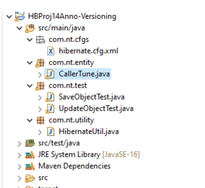
```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.CallerTune;
import com.nt.utility.HibernateUtil;

public class UpdateObjectTest {

	public static void main(String[] args) {
		//get SessionFactory object
		SessionFactory factory=HibernateUtil.getSessionFactory();
		//get Session object
		Session ses=factory.openSession();
		Transaction tx=null;
		try(factory;ses){
			//Load object
			 CallerTune tune=ses.get(CallerTune.class, 1);
			 if(tune==null) {
				System.out.println("record not found");
				return;
			 }
			 else {
			//begin Tx
			tx=ses.beginTransaction();
			  //update object
			    tune.setTuneName("oo antava ..");
			   ses.update(tune);
			 tx.commit();
			 System.out.println("Object is updated for   ..."+tune.getUpdationCount()+".. times");
			 }//else
		}//try
		catch(Exception e) {
			if(tx!=null || tx.getStatus()!=null || tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Object is not updated");
			}//if
		}//catch
	}//main
}//class
```

CALLERTUNE

TUNEID	UPDATIONCOUNT	TUNENAME	MOVIENAME
1	1	0 ~~oo antava ..~~	pushpa- The raise
		1 ~~sami sami~~	
		2 srivalli	

Versioning using Annotation driven cfgs (<version> ---- @Version)
==========

```
import org.hibernate.annotations.GenericGenerator;

import lombok.Data;

@Data
@Entity
@Table(name="CALLERTUNE_ANNO")
public class CallerTune {
	@Id
	@GeneratedValue
	private Integer tuneId;
	@Column(length = 20)
	private String  tuneName;
	@Column(length = 20)
	private  String movieName;
	@Version
	private  Integer updationCount;  //for Versioning

}
```



CALLERTUNE_ANNO

TUNEID	MOVIENAME	TUNENAME	UPDATIONCOUNT
1	21 pushpa- The raise ~~sami sami~~ ..	srivalli	0 1

==========
Object time stamping feature of Hibernarate
==========

=>This feature keeps track of when the object/record is saved/inserted intially later it also keeps track of when the record/object is lastly updated/modified.
=>Object Versioning feeture keeps track how many times the obejct/record is modified using Hibernate Persistence logic where as Object timestamping feature keeps track when exactly the Object/record is inserted and lastly updated.
=> In xml driven cfg we can not apply both fetures at time in single HB mapping file , where as in annotation driven cfgs it is quite possible.

usecases for timestamping :: keeping track of when the password is lastly changed
keeping track of when stock market share value is lastly updated/modified
keeping track of when the last Tx happend in the Bank Account.
and etc..

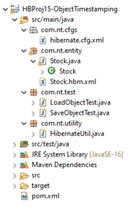
=> Applying Timestamping in xml driven cfgs needs

a) Take java.sql.Timestamp or java.util.Date type property in the Entity class
In entity class
java.sql.Timestamp lastUpdated;

b) cfg the the above property in mapping file using <timestamp> feature.
In mapping file
<timestamp name="lastUpdated" /> (place this after <id> tag)

c) Develop one client App to save object and another Client App to load and update the object.

Example App (maven project with 3 dependencies hibernate-core ,project lombok , ojdbc8)
==========
versioning.timestamping
=> In xml driven cfg we can not apply both fetures at time in single HB mapping file becoz the XSD/DTD rules does not allow to place both <version>, <timestamp> tags at a time under <class> tag.



Entity class
==========

```
package com.nt.entity;

import java.sql.Timestamp;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
public class Stock {
	private Integer stockId;
	@NonNull
	private String stockName;
	@NonNull
	private double price;
	@NonNull
	private String exchange;
	private Timestamp lastUpdated;  // for Object Timestamping

}
```

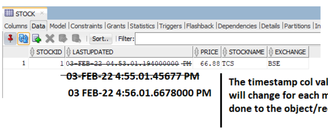
Stock.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

	<hibernate-mapping>
	 <class name="com.nt.entity.Stock" >
	   <id name="stockId">
	     <generator class="increment"/>
	   </id>
	   <timestamp name="lastUpdated" />
	   <property name="price" />
	   <property name="stockName"  length="20"/>
	   <property name="exchange" length="20"/>
	 </class>
	</hibernate-mapping>
```

hibernate.cfg.xml
---
same as previous

```
//SaveObjecTest.java
=====================
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Stock;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

	public static void main(String[] args) {
		//get SessionFactory
		SessionFactory factory=HibernateUtil.getSessionFactory();
		//get Session
		Session ses=factory.openSession();
		Transaction tx=null;
		try(factory;ses){
			//begin Tx
			tx=ses.beginTransaction();
		  //prepare object
		    Stock stock=new Stock("TCS",56.77,"BSE");
		    int idVal=(Integer)ses.save(stock);
	    tx.commit();
	    System.out.println("Object/record is saved with the id value::"+idVal);
	    System.out.println("Object/record is saved at::"+stock.getLastUpdated());
		}//try
		catch(Exception e) {
			if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Object is not saved");
			}

		}//catch
	}//main
}//class
```

```
//LoadObjectTest.java

package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Stock;
import com.nt.utility.HibernateUtil;

public class LoadObjectTest {

	public static void main(String[] args) {
		//get SessionFactory
		SessionFactory factory=HibernateUtil.getSessionFactory();
		//get Session
		Session ses=factory.openSession();
		Transaction tx=null;
		try(factory;ses){
			//Load object
			Stock stock=ses.get(Stock.class, 1);
			if(stock==null) {
				System.out.println("Stock not found");
				return ;
			}
			else {
				//begin Tx
				tx=ses.beginTransaction();
	// modify the loaded object
				 stock.setPrice(66.88);
				 ses.update(stock);
				tx.commit();
				System.out.println("Object/record is lastly updated at:: "+stock.getLastUpdated());
			}
		}/try
		catch(Exception e) {
			if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Object is not updated");
			}

		}//catch
	}//main
}//class
```



~~03-FEB-22 4:55:01:45677 PM~~
03 FEB-22 4:56.01.6678000 PM

The timestamp col value will change for each modification done to the object/record.

=> In annotation driven HB programming
->we can enable timestamp, version features at a time on single entity class
-> In timestamp fetaure we can we can take seperate col/property holding the object /record insertion date and time (@CrationTimestamp)
and another seperate property holding the object/record lastupation date and time. (@UpdationTimestamp)

==========
Example App having versoning , timestamping
==========

```
package com.nt.entity;

import java.sql.Timestamp;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Version;

import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.UpdateTimestamp;

import lombok.Data;

@Data
@Entity
@Table(name="CALLERTUNE_ANNO_VER_TS")
public class CallerTune {
	@Id
	@GeneratedValue
	private Integer tuneId;
	@Column(length = 20)
	private String tuneName;
	@Column(length = 20)
	private String movieName;
	@Version
	private Integer updationCount;  //for Versioning
	@CreationTimestamp
	private Timestamp launchTime;  // holds record insertion time
	@UpdateTimestamp
	private Timestamp lastUpdationTime;  //holds record lastupdation time

}
```

(changes time to time) (fixed) (changes)

Composite ID field Configuration

=> If pk constraint is applied on single col of db table is called singular pk constraint
=>If pk constraint is applied on the multiple cols of db tables together then it is called composite pk constraint.

=>If all cols of db table are not capable maintaining unique values .. then we need to see either the combination multiple cols together can maintain duplicate values or not .. If possible with multiple cols then go for composite pk constraint.

Projects_Programers_info (sometimes 1 programmers neeeds to work in multiple projects)

prgmrId	prgmrName	salary	projId	projName	budget
101	raja	90000	1001	openFx	2000000
101	raja	90000	1002	Amex	3000000
102	suresh	80000	1001	openTx	2000000
102	suresh	80000	1003	DigiServ	4000000

composite pk on multiple cols togather

=>If one property of entity class is cfg as id property then it is called singular Id .. For this we need to use <id> tag.
=>If multiple properties of entity class togather is cfg as id property then it is called composite Id .. For this we need to use <composite-id> tag.

conclusion:: singular
If the underlying db table is having pk constraint then take pk col related property of entity class and make it as singular id property using <id> tag.

composite
If the underlying db table is having pk constraint applied on the multiple cols togather then take those multiple col related multiple properties in entity class and make them as composite id property using <composite-id> tag.

=> We can cfg id generator like increment only while working singulr id filed cfg..i.e not possible while working with composite id field cfg becoz we make multiple properties same type of different types togather as composite id filed properties.

=> Composite id filed (multiple property values) must be supplied/assigned manually.

=>The various methods of hibernate persistence logic that use on Session object expects id value as Serializable obj and gives the generated id value as Serializable obj

save(-) method signature :
public Serializable save(Object obj) | Gives the generated id value as the Serializable object

get(-,-) method signature :
public <T> T get(Class<T> clazz, Serizlable id) | Expects the id value as Serializable object

id value :: 100 ----{-autoboxing}-----> wrapper Integer obj(100)
(note:: All wrapper class objs are Serializable objs by default)

eg1:: int idVal=(Integer)ses.save(prod);
-->returns id property (Integer pid) value as the id value in the form of Integer Wrapper class obj ..which will be converted into simple int later using autounboxing concept.

eg2: Product prod=ses.get(Product.class, 101);
simple int converts wrapper Integer using AutoBoxing concept which is indirectly Serializable obj.

While working with composite Id filed cfg --> we still need to pass that composite id value as Serializable obj and we get that same id value as Serilizable obj.

=> While designing entity class having composite id filed cfg .. we need to take seperate class having multiple properties of composite id and make that class implementing java.io.Serializable(I) as shown below

Projects_Programers_info (sometimes 1 programmers neeeds to work in multiple projects)

prgmrId	prgmrName	salary	projId	projName	budget
101	raja	90000	1001	openFx	2000000
101	raja	90000	1002	Amex	3000000
102	suresh	80000	1001	openTx	2000000
102	suresh	80000	1003	DigiServ	4000000

composite pk on multiple cols togather

Composite Id class /Id class

```
@Data
public class PrgmrProjId implements java.io.Serializable{
  private Integer prgmrId;
  private Integer projId;
}
```

we can place any type and any no.of properties in the Id class.

Entity class

```
@Data
public class ProgrammerProjectInfo {
   //HAS-A property for Id class
  private PrgmrProjId id;
  private String prgmrName;
  private String projName;
  private double salary;
  private double budget;
}
```

we need to cfg composite id filed (id) in mapping file using special <composite-id> tag as shown below.

In Mapping file

```
<class name="pkg.ProgrammerProjectInfo" table="PROGRAMMER_PROJECTS_INFO">
  <composite-id name="id" class="pkg.PrgmrProjId">
    <key-property name="prgmrId" column="PRGMRID"/>
    <key-property name="projId" column="PROJID"/>
  </composite-id>
  <property name="prgmrName" column="PRGMRNAME"/>
  <property name="projName" column="ProjNAME"/>
  ....
  ....
</class>
```

:composite id field cfg

Example App

hibernate.cfg.xml

-->similar to old projects

//Id class

```
@Data
@AllArgsConstructor
@NoArgsConstructor
public class PrgmrProjId implements Serializable{
    private Integer prgmrId;
    private Integer projId;

}
```

// Entity class

```
package com.nt.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class ProgrammerProjectInfo {
    //HAS -A property of Id class
    private PrgmrProjId id;
    private String prgmrName;
    private String projName;
    private double salary;
    private double budget;

}
```

mapping file

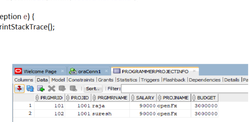
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.ProgrammerProjectInfo" >
    <composite-id name="id" class="com.nt.entity.PrgmrProjId">
      <key-property name="prgmrId"/>
      <key-property name="projId"/>
    </composite-id>

    <property name="prgmrName" length="20" />
    <property name="salary" />
    <property name="projName" length="20"/>
    <property name="budget" />
  </class>
</hibernate-mapping>
```


saveObjectTest.java

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.PrgmrProjId;
import com.nt.entity.ProgrammerProjectInfo;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {
   public static void main(String[] args) {
        //get SessionFactory
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=factory.openSession();
        Transaction tx=null;
        try{factory;ses){
            //begin Tx
            tx=ses.beginTransaction();
              //prepare Id class object
              PrgmrProjId id=new PrgmrProjId(102,1001);
              //prepare Entity class object
              ProgrammerProjectInfo info=new
                        ProgrammerProjectInfo(id, "suresh", "openFx",90000, 3000000);
            //save object
            PrgmrProjId idVal=(PrgmrProjId) ses.save(info);
    tx.commit();
    System.out.println("Object/record is saved with the id value::"+idVal);

        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not saved");
            }

        }//catch
   }//main

}//class
```

//LoadObjectTest.java

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.PrgmrProjId;
import com.nt.entity.ProgrammerProjectInfo;
import com.nt.utility.HibernateUtil;

public class LoadObjectTest {

    public static void main(String[] args) {
        //get SessionFactory
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=factory.openSession();
        try{factory;ses){
            //prepare id class obj
            PrgmrProjId id=new PrgmrProjId(101,1001);
            //Load object
            ProgrammerProjectInfo info=ses.get(ProgrammerProjectInfo.class, id);
            if(info!=null)
             System.out.println(info);
            else
                System.out.println("record nof found");
        }//try
        catch(Exception e) {
            e.printStackTrace();
        }

    }//main
}//class
```



Annotation driven composite id field cfg

=>use @Id for singular Id filed cfg

=> use @ EmbeddedId for composite id filed cfg

note: While working composite id field cfg ... we need to take @Embeddable on top of Id class indicating that class not an entity class , that is embeddable/attachable helper class to Entity class.

=>The HAS -A property of Entity class of type Id class must be annotated with @EmbeddedId annotation indicating that property is internally holding multiple other properties representing composite Id field.

Example App
==========

```
// Id class

package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
@Embeddable
public class PrgmrProjId implements Serializable{
	@Column(name="PRGMRID")
	private Integer prgmrId;
	@Column(name="PROJID")
	private Integer projId;

}
```

Entity class

```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.Table;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name="PROGRAMMERPROJECTINFO")
public class ProgrammerProjectInfo {
	//HAS -A property of Id class
	@EmbeddedId
	private PrgmrProjId id;
	@Column(length = 20,name = "PRGMRNAME")
	private String prgmrName;
	@Column(length = 20,name="PROJNAME")
	private String projName;
	@Column(name="SALARY")
	private double salary;
	@Column(name="BUDGET")
	private double budget;
}
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Connection properties -->
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">manager</property>

    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>


    <!-- mapping file name -->
    <mapping class="com.nt.entity.ProgrammerProjectInfo"/>
  </session-factory>

</hibernate-configuration>
```

ClientApp1

//SaveObjectTest.java

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.PrgmrProjId;
import com.nt.entity.ProgrammerProjectInfo;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

	public static void main(String[] args) {
		//get SessionFactory
		SessionFactory factory=HibernateUtil.getSessionFactory();
		//get Session
		Session ses=factory.openSession();
		Transaction tx=null;
		try(factory;ses){
			//begin Tx
			tx=ses.beginTransaction();
			  //prepare Id class object
			  PrgmrProjId  id=new PrgmrProjId(101,1002);
			 //prepare Entity class object
			  ProgrammerProjectInfo info=new ProgrammerProjectInfo(id, "mahesh", "openFx",80000, 4000000);
			  //save object
			  PrgmrProjId idVal=(PrgmrProjId) ses.save(info);
		tx.commit();
		System.out.println("Object/record is saved with the id value::"+idVal);

		}//try
		catch(Exception e) {
			if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Object is not saved");
			}

		}//catch
	}//main
}//class
```

LoadObjectTest.java

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.PrgmrProjId;
import com.nt.entity.ProgrammerProjectInfo;
import com.nt.utility.HibernateUtil;

public class LoadObjectTest {

	public static void main(String[] args) {
		//get SessionFactory
		SessionFactory factory=HibernateUtil.getSessionFactory();
		//get Session
		Session ses=factory.openSession();
		try(factory;ses){
			//prepare id class obj
			PrgmrProjId id=new PrgmrProjId(101,1002);
			//Load object
			ProgrammerProjectInfo info=ses.get(ProgrammerProjectInfo.class, id);
			if(info!=null)
			 System.out.println(info);
			else
				System.out.println("record nof found");
		}//try
		catch(Exception e) {
			 e.printStackTrace();
			}
	}//main
}//class
```

How to choose column for Primary key constraint

Candidate key column

=>The column using which we can identity the record and access the record is technically called as candidate key column ..

=> Generally the column that holds unique values and may choose as pk column in future is called candidate key column

Person_info (db table)

```
|-->pid (n)
|-->pname (vc2)
|-->paddrs (vc2)
|-->mobileNo (n)
|-->voterId
|--->passportNo
|--->panNo
|--->age
|--->qualification
```

Candidate key colums

Natural key column

=>The candidate key column whose values are having business meaning and whose will be changed becoz of the outside world business polocies is called natural key column

```
|-->mobileNo (n)
|-->voterId
|--->passportNo
|--->panNo
```

These column value are having business meaning in outside world and these value will change according to the outside world business polocies.

=> adding 11 digit in mobile numbers
=> adding IND word before adhar number
=> making bank account number as 16 digit number and etc..

note:: Natual key column values will be given by enduser ..like giving aadharNo , voterId , panNo and etc..

Surrogate key column

=> it is the candiate key column who value is generated by the underlying App or Db s/w dynamically i.e this column column will not be collected from enduser... does not any business meaning and will not be changed for any outside business polocies.

eg:: sid, empno, pid generated by underlying college App, office App or society app
eg:: indentity values generated by the Generators of Hibernate framework

Limitations of taking natural key column as PK column

=> values are very lengthy ,so they need more memory to allocate

=> Values are expected from enduser . if enduser fails to give those values then the record insertion will be failed.

=> Becoz of the outside world business polocies .. the values in pk column will be changed .. this values change effect the dependent tables FK columns data and their dependent db tables data and so on.. In this process the respective java classes data will also be changed .. Overall this change becomes very costly..

FK --->Foreign key column
PK ----> Primary key column

note:: It is recomanded , not to take Natural key column as PK column.

Advantages of taking Surrogate key column as PK column

a) These are dynamically generated values from underlying Db s/w or from underlying Application not having any business meaning .. So no possiblity of changing becoz of the outside world business polocies

b) Values are very short values .. So they will not allocate much memory

c)These Values are not expected from enduser.. So if enduser fails to supply certain values then record insertion will not be failed.

=>candidate key column , natural key column, surrogate key column and etc.. are logical columns i.e they will be applied on already available other physical cols of the db table.

Generators in Hibenrate

=>These generators generate values to singular Id propery using one or another algorithm i.e they are generating values to PK columns dynamically.. which indicates pk col is surrogate key column.

Generators in xml driven cfgs

=>increment
=>identity
=>sequence
=>hilo (removed in 5.x)
=>seqhilo
=>native
=>uuid
=>guid
=>foreign
=>select
=>assigned (default)
and etc.

JPA Generators in annpotation driven cfgs

AUTO (default)
IDENTITY
SEQUENCE
TABLE

note:: Custom generators can be devleped

Generators in xml driven cfgs of hibernate

=>the above listed gnerators are the short names lengty gnerator classes who extend from org.hibernate.id.IndentifierGenerator(c)

short name	class
increment	org.hibernate.id.IncrementGenerator
identity	org.hibernate.id.IdentityGenerator
assigned	org.hibernate.id.Assigned

and etc..

All these classes indirectly impleme org.hbernate.id.IdentifierGenerator(I)

note:: Generators are applicable only on singular id filed cfg .. not on composite id field cfg

syntax for generators

```
<id name="....."    column="......">
    <generator class="...............">
        <param name="......" value="....."/>
        <param name="......" value="....."/>
        <param name="......" value="....."/>
    </generator>
</id>
```

generator short name or fully qualifed class name

input values to the generators.

```
eg:: <id name="pid" column="PID">
        <generator class="increment"/>
                (or)
        <generator class="  org.hibernate.id.IncrementGenerator"/>

     </id>
```

=>while cfg the generator , we need to choose the choose by considering the following two criterias

a) The underlying Db s/w should support the generator
b) The value generated by generator must be compitable to store in the id property.

note:: The generators in hibernate makes programmer to cfg identity property or relavant pk colum as Surrogate key column.

HB Generators in xml driven cfgs
==========

=>increment
=>identity
=>sequence
=>hilo {removed in 5.x}
=>seqhilo
=>native
=>uuid
=>guid
=>foreign
=>select
=>assigned (default)
and etc.

assigned
=======

=>Makes the application to assign the value to id property before calling ses.save(-) method.
=> if no generators is cfg , default generator that will be applied is "assigned".
=> "assigned" , we need to assign the value to the id property manually i.e nothing will be generated dynamically.
=> It works for all DB s/ws ..
=> It allows to us to take any type of property as the id property.

eg: In mapping file

```
<id name="Id" column="PID">
    <generator class="assigned"/>
            (or)
    <generator class="org.hibernate.Id.Assigned"/>
</id>
```

note:: It is not industry standard generator becoz it makes the programmer to assign the value to id property manually.. in that process we may give duplicate id values voilating PK constraint.

usecase:: As part vechile registration .. if we take regNo as the pk column /id property then u want to select fancy number as vehicle registration number then this "assigned" generator is good.

TSO7 EN 4779
|---> 4+7+7+9 =27
|---> 2+7 =9

this kind of number should assigned as vehicle registration number through manual process.

SaveObjectTest.java
==========

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

    public static void main(String[] args) {

        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=HibernateUtil.getSession();

        Transaction tx=null;
        try{factory;ses){
            // begin Transaction
              tx=ses.beginTransaction();  //Internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
                //prepare entity object
                Product prod=new Product();
                prod.setPid(144);
                prod.setPname("table1"); prod.setPrice(1368.5f);
                prod.setQty(12.0f);
      //save object
                Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
                System.out.println("The Generated id value ::"+idVal);
    tx.commit();  // Internally calls con.commit() method to make insertion executin result perminent
    System.out.println("Object is saved(Record is Inserted)");
        }
        catch(HibernateException he) {
        he.printStackTrace();
        if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
            tx.rollback();  //internally calls con.rollback() method to rollback the results of  query execution.
            System.out.println("Object is not saved(Record is not Inserted)");
        }//if
        }//catch

    }//main
}//class
```

---

increment
==========

=>Generates the id value of type int , long , short , byte .
=>uses <max val>+1 to generate intial id value of execution by hitting the Db s/w..
The next id values of same execution will be generated using "generated value +1" formulae.
=> Suitable for multithreaded env..
=> Not suitable for Clustered env.. (making multiple copies same app running in different JVMs of same machine or different machines)
=> For any Generator the generator class will be instantiated during the application bootstrap (when Session,SessionFactory , Configuration objects are ready) and that instatiated object will be used for multiple times.

=> This Generators (increment) works in all Db s/ws..

case1:: for one id value generation , hits the Db s/w and gets the max value and performs max val+1 operation to generated id vlaue

In mapping file

```
<id name="pid" column="PID">
    <generator class="increment"/>
</id>
```

In client app
==========

```
Transaction tx=null;
    try{factory;ses){
        // begin Transaction
          tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
            //prepare entity object
            Product prod=new Product();
            prod.setPid(144); // This value will be overridden with the generator
            prod.setPname("table1"); prod.setPrice(1368.5f);    generated value.
            prod.setQty(12.0f);
  //save object
            Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
            System.out.println("The Generated id value ::"+idVal);
 tx.commit();  // Internally calls con.commit() method to make insertion executin result perminent
 System.out.println("Object is saved(Record is Inserted)");

    }
    catch(HibernateException he) {
    he.printStackTrace();
    if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
        tx.rollback();  //internally calls con.rollback() method to rollback the results of  query execution.
        System.out.println("Object is not saved(Record is not Inserted)");
    }//if
    }//catch
```

case2: if one Client App calls ses.save(-) method for multiple times .. only first the max value will come Db table to gernate first id value using max val1+ formulae.. the remaining id values will generated using "already generated val + 1" formulae.

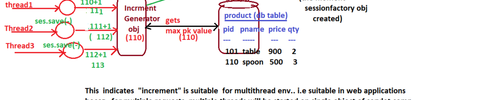
```
Transaction tx=null;
    try{factory;ses){
        for(int i=1;i<=5;++i) {
        // begin Transaction
          tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
            //prepare entity object
             Product prod=new Product();
             prod.setPid(144);
             prod.setPname("table1"); prod.setPrice(1368.5f);
             prod.setQty(12.0f);
  //save object
            Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
            System.out.println("The Generated id value ::"+idVal);
 tx.commit();  // Internally calls con.commit() method to make insertion executin result perminent
 System.out.println("Object is saved(Record is Inserted)");
            }

    }
    catch(HibernateException he) {
    he.printStackTrace();
    if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
        tx.rollback();  //internally calls con.rollback() method to rollback the results of  query execution.
        System.out.println("Object is not saved(Record is not Inserted)");
    }//if
    }//catch
```

Generated Queries

```
Hibernate:
    select    max(PID)    from    PRODUCT
The Generated id value ::147
Hibernate:
    insert
    into
        PRODUCT
        (PNAME, PRICE, QTY, PID)
    values
        (?, ?, ?, ?)
Object is saved(Record is Inserted)
The Generated id value ::148
Hibernate:
    insert
    into
        PRODUCT
        (PNAME, PRICE, QTY, PID)
    values
        (?, ?, ?, ?)
Object is saved(Record is Inserted)
The Generated id value ::149
Hibernate:
    insert
    into
        PRODUCT
        (PNAME, PRICE, QTY, PID)
    values
        (?, ?, ?, ?)
Object is saved(Record is Inserted)
The Generated id value ::150
Hibernate:
    insert
    into
        PRODUCT
        (PNAME, PRICE, QTY, PID)
    values
        (?, ?, ?, ?)
Object is saved(Record is Inserted)
The Generated id value ::151
Hibernate:
    insert
    into
        PRODUCT
        (PNAME, PRICE, QTY, PID)
    values
        (?, ?, ?, ?)
Object is saved(Record is Inserted)
```

---

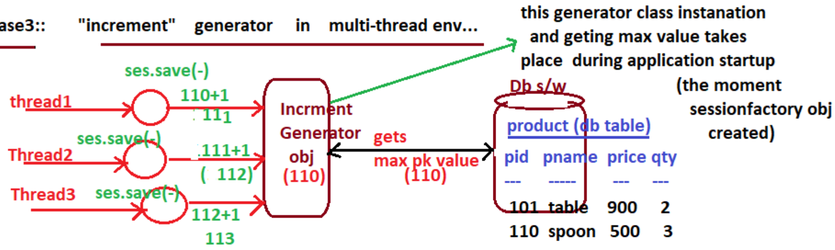
case3:: "increment" generator in multi-thread env...



This indicates "increment" is suitable for multithread env.. i.e suitable in web applications becoz for multiple requests multiple threads will be started on single object of servlet comp class.

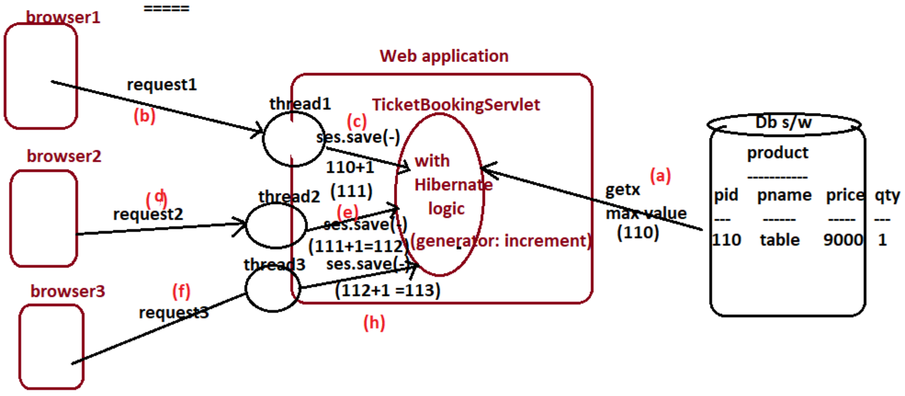
Generators in Xml driven cfgs (increment generator)

case3:: "increment" generator in multi-thread env...



=>Generally we get multi-threaded env.. web applications.. like online ticket booking Apps, e-commerce app and etc..

usecase::
=====



Logic to create Thread
==========

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class TicketBookingOperation implements Runnable {

        @Override
        public void run() {
                //get Session
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(ses){
                        //begin Tx
                        tx=ses.beginTransaction();
                        Product prod=new Product();
                        prod.setPname("table"); prod.setPrice(90000.0f);
                        prod.setQty(67.88f);
                         int idVal=(Integer)ses.save(prod);
                         System.out.println("Generated id value ::"+idVal);
                        tx.commit();
                        System.out.println("Object is saved");
                }
                catch(Exception e) {
                        if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                                tx.rollback();
                                System.out.println("Object is not saved");
                        }//if

                }//catch
        }
}
```

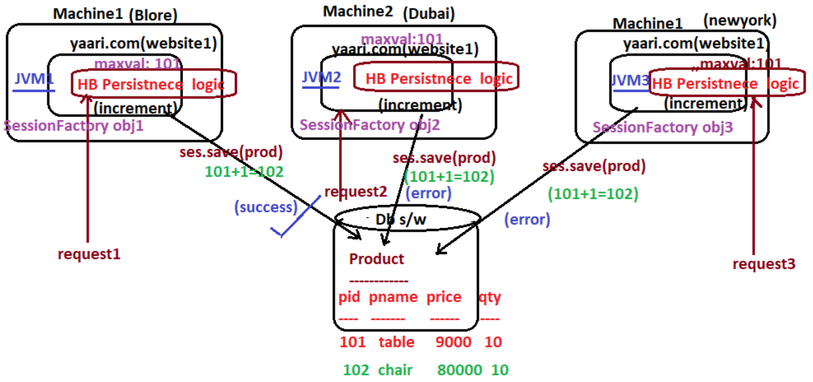
SaveObjectInMultithreded.java

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class SaveObjectTestInMultiThreaded {

        public static void main(String[] args) {

                TicketBookingOperation operation=new TicketBookingOperation();
                Thread th1=new Thread(operation);
                th1.start();

                Thread th2=new Thread(operation);
                th2.start();

                Thread th3=new Thread(operation);
                th3.start();

        }//main
}//class
```

Generators in Xml driven cfgs (Increment generator)
==========

=> Increment generator is not suitable in clustred env..

=> A cluster env.. that runs same app on multiple machines of network
for load balance and transparent failover activity.

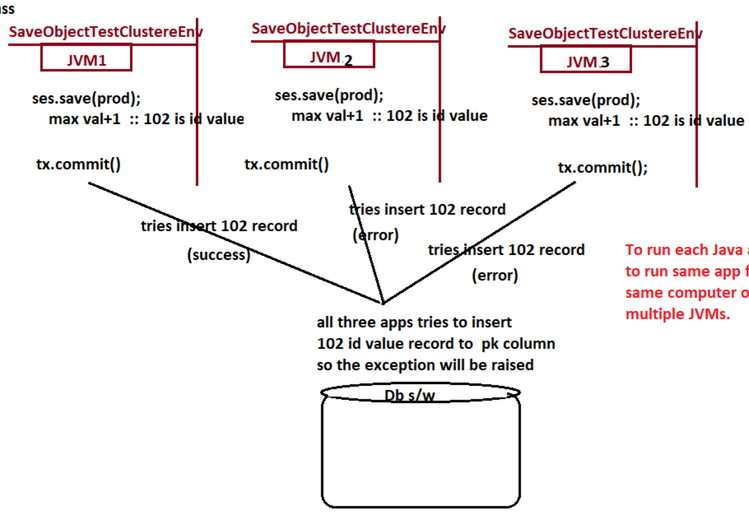


=>if all the 3 apps running on the different jvms of 3 different computers simultaenously gets
max value from db table by taling to Db s/w through IncrementGenerator instantiation then
there is possiblity of getting unique constraint voilation.

Example code
==========

```
ackage com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class SaveObjectTestForClusteredEnv {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=HibernateUtil.getSession();


        Transaction tx=null;
        try(factory;ses){
            // begin Transaction
              tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
                //prepare entity object
                 Product prod=new Product();
                 prod.setPid(144);
                 prod.setPname("table1"); prod.setPrice(1368.5f);
                 prod.setQty(12.0f);


       //save object
              Integer idVal=  (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
              System.out.println("The Generated id value ::"+idVal);

              try {
                    Thread.sleep(50000);  //during this sleep period run same app for multiple times
                 }
                 catch(Exception e) {
                       e.printStackTrace();
                 }

       tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
        System.out.println("Object is saved(Record is inserted)");

        }
        catch(HibernateException he) {
        he.printStackTrace();
         if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
              tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
               System.out.println("Object is not saved(Record is not inserted)");
         }//if
        }//catch

    }//main
}//class
```

In this sleep period run this app multiple other times.. see the exception in their execution



To run each Java app we need one JVM ..So to run same app for multiple times either in same computer or differnet computers we need multiple JVMs.

Generally the industry takes, the multiple copies of the same app to keep in multiple SErvers /machine
i.e these multiple Apps executing with multiple different jvms ..So that leads to multiuple sEssionFactory objects
and creating multiple generator class objs creations..

Identity Generator
==========

=> Generates the identity value of type long,short,int by using the identity column value of underlying Db s/w.

=> Upto oraclce 11g there are no Identity columns ... So this does not in oracle upto 11g
From oracle 12c .. "autoincrement" constraint is given who value will be generated automatically when insert values to other columns using (previous val+1) formulae.

=> Mysql Db s/w supports this generator becoz it is having "autoincrement" constraint support..
=>While working with this generator the initial value comes as max value +1 and remaining values will comes as previous val+1 formulae.
=> It looks llke "increment" generator but it is different.. In case of "increment" generator the DB generates the id value and Inserts to pk column of db table where as "indentity" generator makes the underlying Db s/w to generate id values and brings that value to assign to id property.

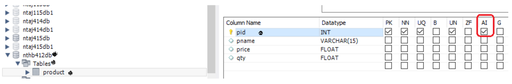
increment Gernrator ::: HB generated value to pk column of db table
identity generatr :: DB table identity col(pk column with autoincrement constraint) comes to id property.

=> This generator is suitable in multi-threaded env..
=>This generator is suitable in clustered env..

if column /pkcolum is having "autoincrement" constraint , then the value to that col will be inserted dymaically based on the values set to other cols

==========
case1:: identity generator generating the id value
==========

step0) *In mysql Db s/w make the pk column (pid) of db table (product) as indentity column by applying "autoincrement" constraint*



step1)Activate entries of mysql Db s/w in hibenrate.cfg.xml file

In hibernate.cfg.xml

```
<property name="connection.url">jdbc:mysql:///nthb412db</property>
<property name="connection.username">root</property>
<property name="connection.password">root</property>
```

step2) Configure "idenity" generator

In mapping file

```
<id name="pid" column="PID">
    <generator class="identity"/>
</id>
```

step3) Run the Client App that performs save object operation

SaveObjectTest.java

```
Transaction tx=null;
    try(factory;ses){
        // begin Transaction
          tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
           //prepare entity object
            Product prod=new Product();
            prod.setPname("table1"); prod.setPrice(1368.5f);
            prod.setQty(12.0f);
  //save object
            Integer idVal=  (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
            System.out.println("The Generated id value ::"+idVal);
  tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
   System.out.println("Object is saved(Record is inserted)");
          }//try
     catch(HibernateException he) {
     he.printStackTrace();
      if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
            tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
            System.out.println("Object is not saved(Record is not inserted)");
      }//if
     }//catch
```

Generated SQL Query :: insert into Product(pname,price,qty) values(?,?,?)

Case2: using "indentity" generator in batch of records insertion

SaveObjetTestjava

```
//SaveObjectTest.java
=======================

package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class SaveObjectTestInBatch {

    public static void main(String[] args) {

        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=HibernateUtil.getSession();

        Transaction tx=null;
        try(factory;ses){

            for(int i=1;i<=5;++i) {
            // begin Transaction
              tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on
Db s/w
                //prepare entity object
                 Product prod=new Product();
                 prod.setPname("table1"); prod.setPrice(1368.5f);
                 prod.setQty(12.0f);
      //save object
                 Integer idVal=  (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object
(insert object data as the record)
                 System.out.println("The Generated id value ::"+idVal);
     tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
      System.out.println("Object is saved(Record is inserted)");
            }//for
            }//try
      catch(HibernateException he) {
      he.printStackTrace();
       if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
            tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
            System.out.println("Object is not saved(Record is not inserted)");
       }//if
      }//catch

    }//main
}//class
```

=>identity generator can be used though table is empty .. if code is executing first time then we get <max value +1> otherwise it gets the id value using (prevuous val+1) formulae/.

What is differece b/w inrecrement and indeity generrator?

increment ==========	indentity ==========
a) Works in all Db s/ws	a) works in only that Db s/w which supports indentity cols (eg:: mysql)
b) HB Generator generates id value to the property and later will be inserted to the db able pk col	b) The underlying Db s/w generates the value to pk column and that column value comes our App as id property value
c) The PK column in db table need does not any other special constriant	c) The pk column in db table needs special constraint like auto-_increment
d) The pk column partcicipates int the insert SQL Query of record insertion	d) The pk column does not partcipate in record insertion related insert SQL query
e) Not suitable for clustering env..	e) suitable for clustered in evn...
f) if the existing max value of pk column is deleted.. the next execution considers suing that value as PK colun value	f) In this case the next vaue is generated using previous value +1 becoz of the values inserted to the other cols.. the autoincrement constraint will take care inserting value to pk column.

note:: Most regulary used Id generator in real projects is "indentity" or "sequence" ..not the increment becoz it is not suitble for clusterd env.. generally all the large scale projects run in clustered env..

---

sequence (HB Id generator)
========

=>Uses the sequence of underlying Db s/w to generate the id value of type int or short or long
=> Oracle Db s/w supports the db "sequences" , So the "sequence" generators works in oracle .
Mysql Db s/w does not support the db "sequences" , So the "sequence" generators does not work in mysql DB s/w
=> Can use the default sequence (hibernate_sequence) to generate the id value or it can use the configured sequence to generate the id value or it can create new sequence besed on the givne related input value.
sequence

sequence (HB Id generator)
==========

=>Uses the sequence of underlying Db s/w to generate the id value of type int or short or long

=> Oracle Db s/w supports the db "sequences" , So the "sequence" generators works in oracle .

Mysql Db s/w does not support the db "sequences" , So the "sequence" generators does not work in mysql DB s/w

=> Can use the default sequence (hibernate_sequence) to generate the id value or it can use the configured sequence to generate the id value or it can create new sequence besed on the givne sequence related input values.

case1:: making hibernate creating default sequence (hibenate_sequenece) having logic start with 1 and increment by 1

step1) make sure that u r application pointing to oracle Db s/w

In hibernate.cfg.xml

```
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">system</property>
<property name="connection.password">manager</property>
```

step2) configure "sequence" generator in mapping file

In mapping file (Product.hbm.xml)

```
<id name="pid" column="PID">
    <generator class="sequence"/>
  </id>
```

Creates the hibernate_sequence having logic start with 1 increment by 1

step3) Write code in Client App to save object the object

In SaveObjectTest.java

```
Transaction tx=null;
    try(factory2ses){
      // begin Transaction
        tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
          //prepare entity object
            Product prod=new Product();
            prod.setPname("table1"); prod.setPrice(1368.5f);
            prod.setQty(12.0f);
  //save object
        Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
        System.out.println("The Generated id value ::"+idVal);
  tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
  System.out.println("Object is saved(Record is inserted)");
        }///try
    catch(HibernateException he) {
    he.printStackTrace();
    if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
        tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
        System.out.println("Object is not saved(Record is not inserted)");
    }//If
    }//catch
```

note:: It uses hibernate_sequence.nextval to generate the id value.

Case2: Making the hibenrate App to use the programmer configured user-defined sequence to generate the id value

step1) keep the above setup same

step2) create user-defined sequence in oracle Db s/w using SQL developer tool

SQLdeveloper ----> sequences ---> right click ---> create new sequence --->

```
CREATE SEQUENCE "SYSTEM"."PRODID_SEQ" MINVALUE 100 MAXVALUE 10000 INCREMENT BY 5 START WITH 100 CACHE 20
NOORDER NOCYCLE ;
```

step3) cfg sequence generator in mapping specifying our sequence name as the input value.

In mapping file

```
<id name="pid" column="PID">
    <generator class="sequence">
      <param name="sequence_name">PRODID_SEQ</param>
    </generator>
  </id>
```

Generated values
100
105
110
115
..
..

step4) Run The client App that is saving the object

```
Transaction tx=null;
    try(factory2ses){
      // begin Transaction
        tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
          //prepare entity object
            Product prod=new Product();
            prod.setPname("table1"); prod.setPrice(1368.5f);
            prod.setQty(12.0f);
  //save object
        Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
        System.out.println("The Generated id value ::"+idVal);
  tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
  System.out.println("Object is saved(Record is inserted)");
        }///try
    catch(HibernateException he) {
    he.printStackTrace();
    if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
        tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
        System.out.println("Object is not saved(Record is not inserted)");
    }//If
    }//catch
```

Case3:: Making hibernate App to generate user-defined sequence dynamically with the given name and param values.

step1) keep the above setup ready (application taking to oracle Db s/w)

step2) cfg the "sequence" generator with the given input values.

```
<id name="pid" column="PID">
    <generator class="sequence">
      <param name="sequence_name">HB_PID_SEQ</param>
      <param name="initial_value">1000</param>
      <param name="increment_size">10</param>
    </generator>
  </id>
```

(for sequence name)
(for start with)
(for increment by)

The generated values
1000
1011
1021
1031
.. ..

Generated sequence is
create sequenxce hb_pid_seq start with 1000 increment by 10 ;

step3) Run the client App

```
Transaction tx=null;
    try(factory2ses){
      // begin Transaction
        tx=ses.beginTransaction();  //internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
          //prepare entity object
            Product prod=new Product();
            prod.setPname("table1"); prod.setPrice(1368.5f);
            prod.setQty(12.0f);
  //save object
        Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
        System.out.println("The Generated id value ::"+idVal);
  tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
  System.out.println("Object is saved(Record is inserted)");
        }///try
    catch(HibernateException he) {
    he.printStackTrace();
    if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
        tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
        System.out.println("Object is not saved(Record is not inserted)");
    }//If
    }//catch
```

=>"sequence" generator is suitable in multi-threaded env..

=>"sequence" generator is suitable in clustered env..

=> mysql Db s/w's "autoincrement" constraint is similar to sequence of oracle Db s/w

What is the difference b/w sequence generator and indentity generator?

sequence ==========	identity ==========
a)uses underlying Db s/w managed "sequence" to generate the id vlaue	a)uses underlying Db s/w managed "identity" column to generate the id vlaue. (like autoincrement column in mysql DB s/w)
b) Works in those Db s/ws that supports sequences (eg: oracle)	b) Works in those Db s/ws which supports identity col (column with auto increment constraint)
c) The sequence generated value will be inserted to PK colum and later will be assigned to id property of Entity class obj	c) The autoincrement constraint based pk col acting as identity colum gets its value dynamically based up on values to insertion to other columns using max val+1 (first value) formulae or previous val+1 formulae(next values) and later that value will be assigned id property of Entity class obj
d) we have control on start value , increment value , end value and etc.. while this generator	d) No such control .. first value is max val+1 formulae and next values are previous val+1 formulae.
e) suitable in multi-thread env.. and clustered env..	e) same here.

=>From Mysql 8.x Version onwords there is indirect support for "sequence" generator with given sequence name temporary db table will be generated having next_val as column holding the next value to generate

=>For example if we confugure "sequence" gneerator in mapping file like this *pointing to mysql DB s/w*

```
<id name="pid" column="PID">
    <generator class="sequence">
      <param name="sequence_name">HB_PID_SEQ</param>
      <param name="initial_value">1000</param>
      <param name="increment_size">10</param>
    </generator>
  </id>
```

Then the mysql Db s/w creates db table HB_PID_SEQ having NEXT_VAL as the column name with the intial value 1000 and chaging value by 10 always

HB_PID_SEQ (db table)
---
next_value

~~1000~~
~~1011~~
~~1021~~
~~1031~~
1041

---

hilo generator
==========

=> Uses hi/lo algorithm to generate the id values of type int or short or long

=> Uses given db table colum value as source of high value and "max_lo" param value as the source of low value to generate the id value

=> The supported parameters are "table" , "column" and max_lo

"table" , "column" are given to specify table ,columm name to get high vlaue
"max_lo" is given to get low value.

=> This "hilo" generated is removed from hibenrate 5.x in support to "seqhilo" generator , So use hibernate 4.x setup to work with this generator

=> This generator works in all Db s/w...

=> The "hilo" generator formulae to generate the id value is
<high value> * <low value +1>

=>This generator is suitable for multi-threaded and env.. and clustered env..

hilo generator
==========

=> Uses hi/lo algorithm to generate the id values of type int or short or long
=> Uses given db table colum value as source of high value and "max_lo" param value as the source of low value to generate the id value
=> The supported parameters are "table" , "colum" and max_lo

"table", "column" are given to specify table ,columm name to get high vlaue
"max_lo" is given to get low value.

=> This "hilo" generator is removed from hibenrate 5.x in support to "seqhilo" generator , So use hibernate 4.x setup to work with this generator
=> This generator works in all Db s/w...

=> The "hilo" generator formulae to generate the id value is
<high value> * <low value +1> (or) (<high value> * <low value>)+ <high value>

=>This generator is suitable for multi-threaded and env.. and clustered env..

Example1
=========

=> change hibenrate version 4.x in pom.xml file by changing dependency in pom.xml file

In pom.xml

```
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-core</artifactId>
   <version>4.3.9.Final</version>
</dependency>
```

=> create helper table in Db s/w with column having initial value in any Db s/w
(This column value becomes source of high value)

HI_TABLE (db table in oracle)

HI_VAL (column in db table)
---
20 (Source of high value)

=> configure hilo generator in mapping file

In product.hbm.xml

```
<id name="pid" column="PID">
    <generator class="hilo">
        <param name="table">HI_TABLE</param>
        <param name="column">HI_VAL</param>
        <param name="max_lo">5</param>
    </generator>
</id>
```

HI_TABLE

HI_VAL

~~10~~
~~11~~
~~12~~
13

For record insertion this vlaue will be incremented by 1

<high value>	*	(<low value>+1)
10		(5+1) = 60
11	*	(5+1) = 66
12	*	(5+1) = 72
13	*	(5+1) = 78

=> write code in Client App to save the object

```
Transaction tx=null;
try{
    // begin Transaction
      tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
        //prepare entity object
          Product prod=new Product();
          prod.setPname("table1"); prod.setPrice(1368.5f);
          prod.setQty(12.0f);
  //save object
        Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
        System.out.println("The Generated id value ::"+idVal);
 tx.commit();  // Internally calls con.commit() method to make insertion executin result perminent
 System.out.println("Object is saved(Record is inserted)");
        }//try
  catch(HibernateException he) {
  he.printStackTrace();
        tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
         System.out.println("Object is not saved(Record is not inserted)");
  }//catch
  finally {
        ses.close();
        factory.close();
  }
```

Example2
=======

=>Every thing same as above but cfg the hilo generator as shown below

In mapping file
==========

```
<id name="pid"  column="PID">
    <generator class="hilo">
      <param name="max_lo">10</param>
    </generator>
  </id>
```

This time the source of high value will be colleted from by creating db table hibernate_unique_key having column "next_hi" with default value 0

<hi value>	*	(<max_lo> value + 1)
0	*	(10 +1) = 0 but it takes 1
1	*	(10+1) = 11
2	*	(10+1) = 22
3	*	(10+1) = 33

hilibernate_unique_key
---
next_hi

~~0~~
~~1~~
2

Example3
========

=> every thing same as example1 but write following the code in mapping file

default table name is :: hibernate_unique_key
default high column value is :: next_hi with 0
default value for max_lo parm is : 32767

<high value>	*	(max_lo vlaue +1)
0	*	32767 +1 = 0 but it takes 1
1	*	32767 +1 = 32768
2	*	32767 +1 = 65535

hilibernate_unique_key
---
next_hi

~~0~~
~~1~~
2

Example4
========

=>Generally the hilo generator will be used for Batch record Insertion where max_lo param value indicates batch size i.e using formulate it generates id value and increments that id value by 1 to insert batch size records, after that again uses the formulae to generate id value.

In mapping file
==========

```
<id name="pid"  column="PID">
    <generator class="hilo">
      <param name="table">HI_TABLE</param>
      <param name="column">HI_VAL</param>
      <param name="max_lo">5</param>
    </generator>
  </id>
```

source of high value

source of low value (batch size)

Code in Client App
==========

```
Transaction tx=null;
    try{
 for(int i=1;i<=10;++i) {
        // begin Transaction
          tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
            //prepare entity object
              Product prod=new Product();
              prod.setPname("table1"); prod.setPrice(1368.5f);
              prod.setQty(12.0f);
      //save object
            Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
            System.out.println("The Generated id value ::"+idVal);
    tx.commit();  // Internally calls con.commit() method to make insertion executin result perminent
     System.out.println("Object is saved(Record is inserted)");
 }
        }///try
    catch(HibernateException he) {
    he.printStackTrace();
        tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
         System.out.println("Object is not saved(Record is not inserted)");
    }//catch
    finally {
        ses.close();
        factory.close();
    }
```

HI_TABLE

HI_VAL

~~10~~
11

first id vlaue :: high value * low value +1
10 * 5 +1 = 60

61
62
63
64
65

next 5 values using increment by 1
becoz max_lo param value is 5 (batch size is 5)

next id value :: high value * low value +1
11 * 5 +1 = 66

67
68
69

next value s are generated using increment by 1

since source of high value is comming db table with programmer generated value or default values.. so hilo generator is removed from hibernate 5.x and recomanding to use seqhilo as alternate.

seqhilo
=======

=>Works in all versions of hibernate ..especially in hibernate 5.x.
=> Works only in those Db s/ws that supports sequences
=> uses the given/default sequence generaed value as the source of high value and max_lo param value as the source of low value.
=> this generator is created by inspiring from "hilo" generator
=> formulae is ::
<seq generted high value>* (max_lo param value+1)
(or)
(<seq generted high value>* (max_lo param value)) + the sequence generated high value

=> This generator works in oracle Db s/w becoz it suppors the sequences but does not work in mysql Db s/w becoz mysql do not support sequences , if u attempt to use in mysql Db s/w we get "org.hibernate.MappingException: org.hibernate.dialect.MySQL8Dialect does not support sequences"

Example1 (Working with programmer created sequence to generate the high value)
=========

step1) make the app pointing to oracle Db s/w

```
SQL>CREATE SEQUENCE "SYSTEM"."PRODID_SEQ" MINVALUE
100 MAXVALUE 10000 INCREMENT BY 1 START WITH 120
CACHE 20 NOORDER NOCYCLE ;
```

step2) cfg seqhilo generator in hibernate mapping file

```
<id name="pid" column="PID">
    <generator class="seqhilo">
      <param name="sequence">prodid_seq</param>
      <param name="max_lo">5</param>
    </generator>
```

(seq generated hi value)	*	(max_lo param val+1)
100	*	5+1 = 600
101	*	5+1 = 606
102	*	5+1 = 612
103	*	5+1 = 618

step3) Run the Client App having logic to save the object

```
Transaction tx=null;
    try(factory;ses){
        // begin Transaction
        tx=ses.beginTransaction();  //internally calls  con.setAutoCommit(false) to disable autoCommit mode on Db s/w
          //prepare entity object
            Product prod=new Product();
            prod.setPname("table1"); prod.setPrice(1368.5f);
            prod.setQty(12.0f);
    //save object
        Integer idVal= (Integer) ses.save(prod); // Gives persistence instruction to hibernate to save object (insert object data as the record)
        System.out.println("The Generated id value ::"+idVal);
  tx.commit();  // internally calls con.commit() method to make insertion executin result perminent
  System.out.println("Object is saved(Record is inserted)");
        }//try
    catch(HibernateException he) {
    he.printStackTrace();
     if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
        tx.rollback();   //internally calls con.rollback() method to rollback the results of  query execution.
        System.out.println("Object is not saved(Record is not inserted)");
     }//if
    }//catch
```

Example2 Using hibernate generated default sequence "hibernate_sequence"
=========

=> all the setup is same as example
=> cfg the following "seqhilo" generator in mapping file

In Mapping file

```
<id name="pid" column="PID">
    <generator class="seqhilo">
      <param name="max_lo">5</param>
    </generator>
  </id>
```

It uses HB created default sequence hibernate_sequence generated value as source of high vlaue and max_lo param value as the source of low value.

hibernate_sequence code

```
CREATE SEQUENCE "SYSTEM"."HIBERNATE_SEQUENCE" MINVALUE 1 MAXVALUE
9999999999999999999999999999 INCREMENT BY 1 START WITH 21 CACHE 20 NOORDER
NOCYCLE ;
```

<seq generated value as hi value>	*	<max_lo param value>+1
1	*	5+1 = 6
2	*	5+1 =12
3	*	5+1 =18
4	*	5+1 =24

example3 (working with default sequence and default max_lo param value)
=========

=> all the setup is same as example
=> cfg the following "seqhilo" generator in mapping file

```
<id name="pid" column="PID">
    <generator class="seqhilo"/>
  </id>
```

=>The default sequence is hibernate_sequence that is start 1 increment by 1
=> the default max_lo param value is 9

<sequence generated hi value>	*	*<max_lo param val +1>*
1	*	*9 +1 = 10*
2	*	*9+ 1 = 20*
3	*	*9 +1 = 30*
4	*	*9+1 = 40*

example4 if the specifined sequence is not available in Db s/w then new sequence
======== will be created with that name having logic start with 1 increment by 1

=> all the setup is same as example
=> cfg the following "seqhilo" generator in mapping file

In mapping file

```
<id name="pid" column="PID">
    <generator class="seqhilo">
      <param name="sequence">HB_PRODID_SEQ2</param>
      <param name="max_lo">5</param>
    </generator>
  </id>
```

Creates HB_PRODID_SEQ2 with the logic start with 1 increment by 1 if that sequence is not already available

<sequence generated hi value>	*	*(<max_lo> param val +1)*
1	*	*5+1 = 6*
2	*	*5+1 = 12*
3	*	*5+1 = 18*

=>hilo , seqhilo generators are generally used form batch record insertion which required in group ticket reservation activity where max_lo param value acts as the batch size

==========
native generator
==========

=>This generator does not have its own behaviour ... It will pick up "sequence" or "identity" or "hilo" generator based on the capabilities of underlying Db s/w

=>if the underlying the Db s/w is oracle Db s/w then it uses "sequence" as the generator
=>if the underlying the Db s/w is mysql Db s/w then it uses "indentity" as the generator
=>if the underlying the Db s/w does not support "sequences" and "indentity" cols then it uses "hilo" as the generator.
=> if the underlying Db s/w supports both sequences, indentity cols then it uses "sequence" as the generator (eg: SQL Server)

example1 :: In oracle Db s/w using *default* "sequence" *hibernate_sequence*

```
<id name="pid" column="PID">
    <generator class="native"/>
  </id>
```

since no sequence name is specified it will create and/or use the default sequence hibernate_sequence.

example2::

In oracle Db s/w using the programmer specified user-defined sequence

```
<id name="pid" column="PID">
    <generator class="native">
      <param name="sequence_name">HB_PRODID_SEQ2</param>
    </generator>
  </id>
```

if this sequence is not availble in oracle Db s/w ..then it creates new sequence with that name having logic start with 1 increment by 1

example3 :: In oracle Db s/w using programmer specified sequence with additional params

In mapping file

```
<id name="pid" column="PID">
    <generator class="native">
      <param name="sequence_name">HB1_PRODID_SEQ3</param>
      <param name="initial_value">100</param>
      <param name="increment_size">2</param>
    </generator>
  </id>
```

if this sequence is not available in Db s/w then it creates new sequence having logic start with 100 and increment by 2

example4:: In mysql Db s/w using indentity generator

=> make sure that pid column (pk) column is having auto increment constraint (making the pk column as the idenity column)

Column Name	Datatype	PK	NN	UQ	B	UN	ZF	AI	G
pid	INT	☑	☑	☑	☐	☑	☐	☑	☐

=> cfg native generator by making the app taking to mysql Db s/w

In mapping file

```
<id name="pid" column="PID">
    <generator class="native"/>
  </id>
```

native, seqhio, hilo generators support multi-threaded env. and clustered env..

UUID (Universal Unique Id)
====
=> *Generates 32 digit hexa decimal number [0-9 & A to F] as the univesal unique id*
=>*Very useful to generate id value in Distributed Apps.*
=> *Generated id value having alphabets , so the identity property must be taken as String property*
=> *This is Hibernate generated id value and given to Db table while inserting record..*
=> *This UUID generator uses following thing to generate id value*
a) *Current machine IP Address*
b) *Current date and time*
c)*Current process id*
and etc..

binary :: 0 and 1
decimal :: 0 to 9
octal :: 0 to 7
hexa decimal :: 0 to 9 and a to f

=> *This is suitable for multi-threaded env.. and clustred env..*
=> *suitable to generate Transaction ids in bank Apps..*
=>this generator works in all DB s/ws.
step1) Keep any HB Project ready

step2) Change the id property to String type.

```
@Data
public class Product {
    // bean propertes
    private String pid;
    private String pname;
    private Double price;
    private double qty;

}
```

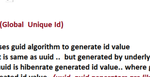
step3) cfg UUID generator in mapping file

```
<id name="pid" column="PID">
    <generator class="uuid"/>
</id>
```

step4) Write save object code in Client app

```
public class SaveObjectTest {

    public static void main(String[] args) {
        try {
        //get SessionFactory
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
            tx=ses.beginTransaction();
        Product p=new Product();
        p.setPname("sofa5");
        p.setPrice(5455.0);p.setQty(55.77);

            String idVal=(String)ses.save(p);
            System.out.println("generated id value ::"+idVal);
            tx.commit();
        }//try
        catch(HibernateException he) {
      he.printStackTrace();
        }
    }
```

In mysql Db s/w



In oracle Db s/w

GUID (Global Unique Id)
====
=>uses guid algorithm to generate id value
=> it is same as uuid .. but generated by underlying Db s/w..
=> uuid is hibernate generated id value.. where guid is the underlying Db s/w
geneated id value. *(uuid ,guid generators are like increment, identity generators)*
=>This generator works in all Db s/ws..
=>This generator uses uuid() function in mysql Db s/w to geneate the id value
=>This generator uses rawtohex(sys_guid()) function in oracle Db s/w to geneate the id value.

example (keep same setup of uuid generator)
======= In mapping file

```
==============
<id name="pid" column="PID">
    <generator class="guid"/>
</id>
```

Select (Geneator)
======
DB s/w's
=>Uses the underlying Db s/w generated value as the id value.
=>Trigger is a component in PL/SQL programming the can perform event hadling on the data of db tables.. like stroing value to pk col before insert , after insert and etc..
=>This generator Works only those Db s/ws that supports triggers..
(like oracle Db s/w)

example

step1) make sure that Application pointing to oracle Db s/w

step2) create BeforeInsert trigger in oracle DB s/w using SQL developer..

SQL Developer --->triggers ---->new Trigger -->

name : ID_GEN_TRIGGER
type :: Before
timing : insert
object :: Product , shema: system

```
CREATE OR REPLACE TRIGGER ID_GEN_TRIGGER
BEFORE INSERT ON PRODUCT FOR EACH ROW
DECLARE
 prod_no number(4);
BEGIN
  SELECT PROD_ID_SEQ.NEXTVAL INTO prod_no FROM DUAL;
  :new.PID :=prod_no;
END;
```

Trigger makes the sequence gerated value as the PID column before inserting every record

step3) cfg "SELECT" generator in mapping file..

```
<generator class="select">
    <param name="key">pname</param>
</generator>
```

Retter to take this column as unique key column

Candidate key column in Db table using which as the criteria column the select operation takes place on PK col to get the trigger inserted value back to HB app and uses it as the id value of the Entity object

step4) Write code in Client app to save the object..

```
public class SaveOjectTest {

    public static void main(String[] args) {
        try {
        //get SessionFactory
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
            tx=ses.beginTransaction();
        Product p=new Product();
        p.setPname("sofa7");
        p.setPrice(5455.0);p.setQty(55.77);

            int idVal=(Integer)ses.save(p);
            System.out.println("generated id value ::"+idVal);
            tx.commit();
        }//try
        catch(HibernateException he) {
      he.printStackTrace();
        }
    }
```

Flow of execution..

=>ses.save(prod) generate insert SQL query having pname,price,qty,PID column values--> before executing this insert query in DB table the Trigger placed in Db s/w gathers value from PROD_ID_SEQ and inserts to PID pk column.. Hibernte now use the the sepcified candidate key column (pname) value as the criteria value to get PID column value back App and makes it as Entity object's id value.

```
Hibernate:
    insert
    into
      PRODUCT
      (PNAME, PRICE, QTY )
    values
      (?, ?, ?, ?)
Hibernate:
    select
      PID
    from
      PRODUCT
    where
      PNAME =?
  generated id value ::121
```

Before this query execution the triliger inserts vlaue pid pk column

foreign
======
=> This generator generates id value to child object by collecting id value from the associated parent object.
=>This generator very useful in one to one association mapping (So will be discussed in future)

Custom Id Generator
==========
step1) All pre-defined generators are implementing org.hibernate.id. IdentifierGenerator(I)
So make u r Generator class also implementing same interface.

RandomIdGenerator.java

```
public class RandomIdGenerator implements IdentifierGenerator {

    @Override
    public Serializable generate(SharedSessionContractImplementor session, Object object) throws HibernateException {

        int value=new Random().nextInt(10000);
        return value;
    }

}
```

step2) Cfg Id Generator in Mapping file

In mapping file

```
<id name="pid" column="PID">
    <generator class="com.nt.generator.RandomIdGenerator"/>
</id>
```

step3) Execute the Client App

SaveObjectTest.java

```
public class SaveObjectTest {

    public static void main(String[] args) {

        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session
        Session ses=HibernateUtil.getSession();

        Transaction tx=null;
        try(Factory;ses){
            // begin Transaction
                tx=ses.beginTransaction();  //Internally calls con.setAutoCommit(false) to disable autoCommit mode on Db s/w
                //prepare entity object
                Product prod=new Product();
                prod.setPname("sofa3"); prod.setPrice(1368.5f);
                prod.setQty(12.0f);
        //save object
                Integer idVal= (Integer) ses.save(prod);
                System.out.println("The Generated id value ::"+idVal);
            tx.commit();  // Internally calls con.commit() method to make insertion executin result perminent
          System.out.println("Object is saved(Record is inserted)");
                }//try
        catch(HibernateException he) {
        he.printStackTrace();
        if(tx!=null && tx.getStatus()!=null  && tx.getRollbackOnly()) {
            tx.rollback();   //Internally calls con.rollback() method to rollback the results of query execution.
            System.out.println("Object is not saved(Record is not inserted)");
        }//if
        }//catch

    }//main
}//class
```

Assingmnet:: Develop custom ID generator which generates the id values like this

```
NIT001
NIT002
NIT003
...
..
...
NIT009
NIT010
```

Hibernate Generators in Annotation driven cfgs

==========

Two types of Generators in annotation driven cfgs

a) Hibernate Generators cfg using annotations (specific to hibernate)
b) JPA Generators cfg using annotations (common for all ORM s/ws) (Best)

a) Hibernate Generators cfg using annotations (specific to hibernate)

=> what ever HB generators we have used in xml driven cfgs.. the same generators can also be cfg using annotation drivens , For that we need to use @GeneratedValue , @GenericGenerator , @Parameter annotations which are required to use along with @Id annotation.

@GenericGenerator :: To cfg certain HB generator having details and providing one logical name
@Parameter :: To supply input values to HB generators cfg
@GeneratedValue :: To cfg Generator in annotation driven env.
(This is required for both HB generators cfg and JPA Generators cfg)

HB Generators list :: assigned, increment, identity , hilo , seqhilo, select, uuid,guid , native and etc..

b) JPA Generators cfg using annotations (common for all ORM s/ws) (Best)

We can cfg the following 4 JPA generators
a) IDENTITY
b) SEQUENCE
c) TABLE
d) AUTO

=>For this we need to use @GeneratedValue , @SequenceGenerator, @TableGenerator annotations along with @Id annotation.

Hibernate Generators cfg using annotations (specific to hibernate)

cfg of HB Generator "increment" using annotations

step1) keep any annotation driven cfg App ready

Movie.java

```
@Entity
@Table(name="MOVIE_INFO")
@Data //lombok api
@AllArgsConstructor
public class Movie {
    @Id //singular Id filed
    @Column(name="MID" ) //mapping with column
    @GenericGenerator(name = "gen1",strategy = "increment")
    @GeneratedValue(generator ="gen1")
    private Integer mid;

    @Column(name="MNAME",length = 20) //mapping with column
    private String mname;

    @Column(name="HERO",length=20) //mapping with column
    private String hero;

    @Column(name="BUDGET")
    private Float budget;

    public Movie() {
        System.out.println("Movie:: 0-param constructor"+this.getClass());
    }

}
```

step2) Develope the Client App to save the object

```
public class SaveObjectTest {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session object
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(ses;factory){
            //begin Tx
            tx=ses.beginTransaction();
            //prepare object
            Movie movie=new Movie();
            movie.setMid(1201);
            movie.setMname("Pupsa");
            movie.setHero("allu-arjun");
            movie.setBudget(1000000000.0f);
            //save object
            int idValue=(Integer)ses.save(movie);
            tx.commit();
            System.out.println("Object is saved with id value::"+idValue);
        }//try
        catch(Exception e) {
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Object is not saved");a
            }
            e.printStackTrace();
        }

    }//main
}//class
```

cfg of HB Generator Identity using annotations

step1) make sure that u r application is pointing to mysql Db s/w

step2) add the following cfg on the id property of entity class

In Movie.java

```
@Id //singular Id filed
@GenericGenerator(name = "gen1",strategy = "identity")
@GeneratedValue(generator ="gen1")
private Integer mid;
```

step3) Develop the client App having logic to save the object.
same as above

cfg of HB Generator sequence using annotations

eg1::

step1) make sure that u r application is pointing to oracle Db s/w

step2) cfg @Id property as shown below

```
@Id
 @GenericGenerator(name="gen1",strategy = "sequence")
 @GeneratedValue(generator = "gen1")
 private Integer mid;
```

note:: creates the new sequence "gen1" if it is not already avalable having logic start with 1 increment by 1.

step3) Develop client App having logic to save the obejct

same as above

eg2::

```
@Id
@GenericGenerator(name="gen1",strategy = "sequence",
                  parameters = {@Parameter(name = "sequence_name", value = "HB_PRODID_SEQ1") }
                )
@GeneratedValue(generator = "gen1")
private Integer mid;
```

=>Uses the sequence HB_PROID_SEQ to generate the id value for mid proeprty

eg3:

```
@Id
@GenericGenerator(name="gen1",strategy = "sequence",
                  parameters = {@Parameter(name = "sequence_name", value = "NIT_PRODID_SEQ1"),
                                @Parameter(name = "initial_value", value = "100"),
                                @Parameter(name="increment_size",value="1")
                                }
                )
@GeneratedValue(generator = "gen1")
private Integer mid;
```

array type annotation attribute expecting @Parameter as the element value

cfg of HB Generator seqhilo using annotations

```
@Id
@GenericGenerator(name="gen1",strategy = "seqhilo",
                  parameters = {@Parameter(name = "sequence_name", value = "NIT_PRODID_SEQ1"),
                                @Parameter(name = "max_lo", value = "10")
                                }
                )
@GeneratedValue(generator = "gen1")

private Integer mid;
```

Does not work as expected

note:: seqhilo is not working in through annotaiton cfgs... Infact some other HB generators are also not working in annotations cfgs .. So prefer using JPA Generators.

Conclusion on HB Generators :: Mostly used HB generators in realtime are
a) In oracle setup use "sequence" generator
b) in mysql setup use "identity" generator

JPA Generators
==========
=>Can be used in only annotation driven cfgs
=>These generators are portable across the multiple ORM f/ws
=>The JPA generators are
a) IDENTITY
b) SEQUENCE
c) TABLE
d) AUTO (default)

Generators in Hibernate

-> HB generators using xml driven cfgs
-> HB generators using annotation driven cfgs
-> JPA generators using annotation driven cfgs
-> Custom Generators

Priority towards choosing the generators
a) JPA Generators
b) Hibernate Generartors
c) Custom Generators

IDENTITY
=======
=> uses Identity col of underlying Db table to get the generated id value of type int or long or short
=>Works only in the Db s/w that supports identity cols (col with auto increment connstraint)
note: works in mysql Db s/w .. does not work in oracle Db s/w (upto 11g)
=>same as hibernate's "identity" generator

example
=======
step1) make sure that the app is pointing to mysql Db s/w (make sure that pk col of db table is enabled with AI (AutoIncrement) constraint
step2) cfg IDENTITY generator

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Id
private Integer mid;
```

step3) Run the Client App having logic to save the object

SEQUENCE
==========
=> uses the underlying DB s/w supported sequence to generate the id value of type int or short or long.
=> if no sequence name is specified .. then it uses/creates the default sequence called hibernate_sequence.having logic start with 1 increment by 1
=>Works in oracle Db s/w .. but does not work in mysql Db s/w.

example1 (with respect to oracle Db s/w)
==========

```
@Id
@GeneratedValue(strategy = GenerationType.SEQUENCE)
private Integer mid;
```

=>creates or uses the default sequence called hibernate_sequence.

example2
=========

```
@Id
@SequenceGenerator(name = "gen1",sequenceName = "JPA_MID_SEQ",
                   initialValue = 700,allocationSize = 5)
@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
private Integer mid;
```

=>creates and uses custom sequence "JPA_MID_SEQ" having logic start with 700 increment by 5
The values generated are 700, 706,711,716,721 , ...

example3
==========

```
@Id
@SequenceGenerator(name = "gen1",sequenceName = "JPA_MID_SEQ1")
@GeneratedValue(generator="gen1",strategy = GenerationType.SEQUENCE)
private Integer mid;
```

=>The default value for "initialValue" is 1
=>The default value for "allocationSize" is 50
=>creates and uses the sequence "JPA_MID_SEQ1" having logic of "start with 1 and increment by 50".
=>The generated values are 1 , 52, 102,152,202,.....

TABLE
=======
=>It is mix of HILO generator +sequence generators.. creates the dynamic db tables and uses them to generate the id values.
=>Works in all Db s/ws.

example1::

```
@Id
@TableGenerator(name="gen1",pkColumnName = "ID_COL",
                table = "ID_TAB",pkColumnValue = "PID",
                valueColumnName = "VAL_COL",initialValue = 400,allocationSize = 2)
@GeneratedValue(generator = "gen1",strategy = GenerationType.TABLE)
private Integer pid;
```

table

=>creates and uses the ID_TAB having columns ID_COL and VAL_COL and uses them to generate the id value.
=>The the vlaue column (VAL_COL) maintains next id value to be generated.

ID_TAB (db table)
======
ID_COL(pk) VAL_COL
--- ---
PID ~~400~~ ~~401~~ ~~404~~ *406, ... , ...*

The generated id vlaues :: 401, 404,406,408,.....

example2:

```
@Id
@GeneratedValue(strategy = GenerationType.TABLE)
private Integer mid;
```

=>It creates dynamic table with the name hibernate_sequences having pk col name "sequence_name" and value col name "next_val"

HIBERNATE_SEQUENCES	
SEQUENCE_NAME	NEXT_VAL
1 default	~~1~~ ~~2~~ ~~3~~ 4

The defualt value of sequence_name col is "default" (pk col value)
the defult initial value is : 1
the defualt allocation size : 1

genearated value are : 1, 2, 3 ...

AUTO (defult JPA Generator)
=====
=>Picks the up generator based on the capabilities of underlying Db s/w (like native generator is hibenrate)
=> In oracle Db s/w it uses "SEQUENCE" generator and in mysql Db s/w it uses " TABLE ".

exampl1:: (for oracle)

```
//AUTO for oracle
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer mid;
```

uses hibernate_sequeence to generate id value

example2 :: (for oracle)

```
@Id
@SequenceGenerator(name = "gen1",sequenceName = "JPA_PID_SEQ2",initialValue = 400,allocationSize = 10)
@GeneratedValue(generator = "gen1", strategy= GenerationType.AUTO)
private Integer mid;
```

creates and uses sequence with the name "JPA_PID_SEQ2" having logic "start with 400 increment by 10".

example3 :: (for mysql)

```
@Id
@GeneratedValue(strategy = GenerationType.AUTO)
private Integer mid;
```

=>it internally uses TABLE generator to generate id values.. creating db table having name "hiberate_sequence" with col next_val

hibernate_sequence
next_val
1/2/3/4/....

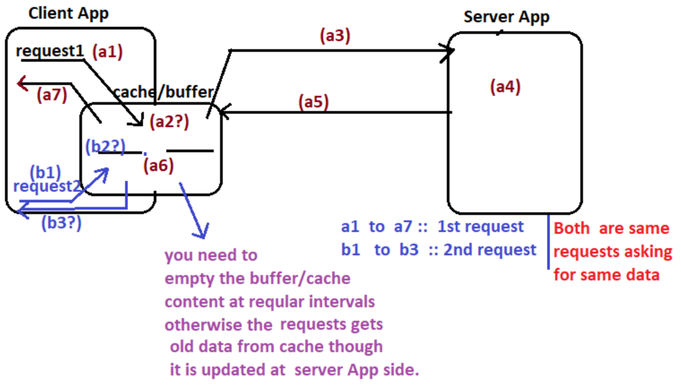
The generated id vlaues :: 1/2/3./4/5/....

=>mostly used JPA generator in the industry is AUTO, IDENTITY

Caching in hibernate
==========

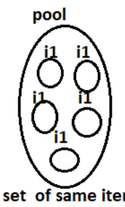
cache/buffer

=> It is a temporary memory that holds data for temporary period
=> In client -Server App , cache/buffer holds data supplid by the server App and uses that data across the multiple same requests having reusability

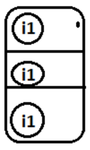


what is the difference b/w pool and cache?

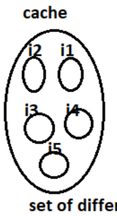
Ans) pool is set of same items having reusability .. cache is set of different items giving reusability.



set of same items
[To implement pool takethe support of List Collection]



eg:: jdbc con pool



set of different items
[To implement cache take the support of Map collection)

keys	values
key1	i1
key2	i2
key3	i3

eg:: Hibernate's Level1 and Level2 cahes
Hibernate's query cache

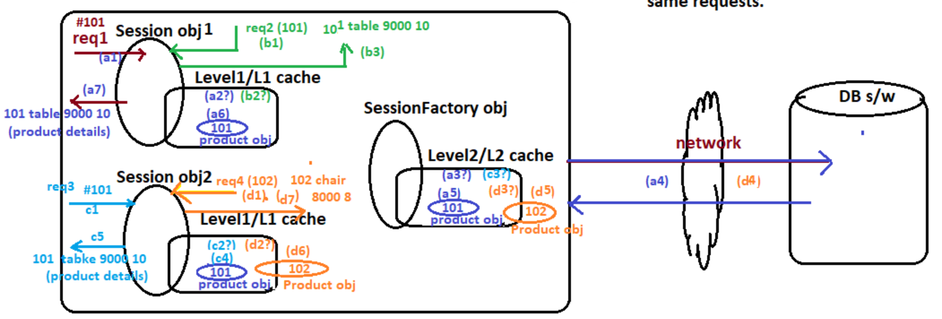
=>Hibernate supports two levels of caching to hold the objects of entity classes representing the records across the multiple same requets b/w hibernate App (java app with hibernate code) and Db s/w.

1) Level1 /L1 cache/ Session cache /first Level cache
[One per each session object]

2) Level2/L2 cache/ SessionFactory cache/Global cache /second level cache
[one per each SessionFactory obj i.e one per each DB s/w]

Both these caches holds records in the form of Objects of entity classes and those objects will be reused acorss the multiple same requests.

Hibernate App (Java App with Hibernate Persistnece logic)



a1 to a7 :: 1st request
b1 to b3 :: 2nd request
c1 to c5 :: 3rd request

all these 3 requests are same requests asking for same data.

d1 to d7 :: 4th request

First Level cache/Level1 /L1 cache/Session Cache
==========

=>It is built-in cache in hibernate i.e it comes automatically for every HB Session object creation
=> It is 1 per Session object
=> This cache can not be disabled becoz it is part of hibernate echo system i.e all CURD Operations in hibernate takes place by involving L1/Level1 cache directly or indirectly.

case1 of L1 cache
==========

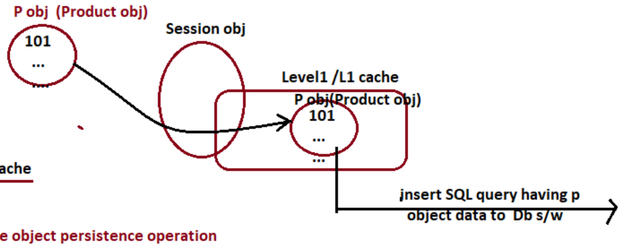
=> When we call ses.save(-) or ses.update(-) or ses.delete(-) or ses.saveOrUpdate(-) or ses.merge(-) having the object entity class .. first given object goes to L1 cache of current Session obj and will be there untill tx.commit() is called and certain persistence operation is completed by generating the SQL Query.

```
Transaction tx=null;
try(factory;ses){
  //Begin Tx
  tx=ses.beginTransaction();
   //create Product object
    Product p=new Product();
     p.setPname("table"); ....
       ....
     int idVal=(int)ses.save(p);
    S.o.println("The generated id value :"+idVal);
    tx.commit();
    S.o.p("object is saved");
  }
catch(Exception e){
  tx.rollback();
  }
```

in the heap memory

p obj moves to L1/Leve1 cache

p obj data is collected from L1 cache and will be used insert /save object persistence operation

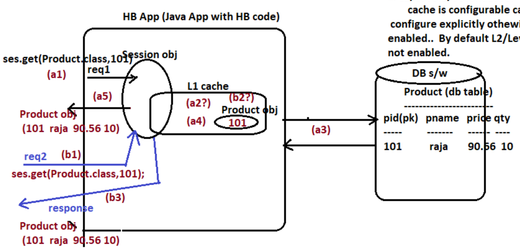


case2::
======

case2::
======

if we try to load same record/object for multiple times by passing same id value using ses.get(-,-) method only for 1st time the record/object will come from Db table ..and that object will be placed L1 cache to use across the multiple same requests .. This gives reusability of object and reduces the network round trips b/w Java app(HB code) and Db s/w.

=>L2/Level2/SecondLevel cache is configurable cache .. we need to configure explicitly othewise will not be enabled.. By default L2/Level2 cache is not enabled.



a1 to a5 :: 1st request
b1 to b3 :: 2nd request

(Both are same requests)

Sample code
==========

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.nt.entity.Movie;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

    public static void main(String[] args) {
        //get SessionFactory object
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session object
        Session ses=HibernateUtil.getSession();
        try(factory;ses){
        // Load object                                                    in
        Movie movie=(Movie)ses.get(Movie.class, 24);  //Collects from DB and keeps  L1 cache
        System.out.println(movie);
        System.out.println("----------------------");
        Movie movie1=(Movie)ses.get(Movie.class, 24);  //Collects from  L1 cache
        System.out.println(movie1);
        System.out.println("----------------------");                    in
        Movie movie2=(Movie)ses.get(Movie.class, 25);  //Collects from DB and keeps  L1 cache
        System.out.println(movie2);
        }//try
        catch(Exception e) {
            e.printStackTrace();
        }


    }//main
}//class
```

output messages
==========

```
Hibernate:
    select
        movie0_.mid as mid1_0_0_,
        movie0_.BUDGET as budget2_0_0_,
        movie0_.HERO as hero3_0_0_,
        movie0_.MNAME as mname4_0_0_
    from
        MOVIE_INFO movie0_
    where
        movie0_.mid=?
Movie:: 0-param constructorclass com.nt.entity.Movie
Movie(mid=24, mname=Pupsa, hero=allu-arjun, budget=1.0E9)
----------------------
Movie(mid=24, mname=Pupsa, hero=allu-arjun, budget=1.0E9)
----------------------
Hibernate:
    select
        movie0_.mid as mid1_0_0_,
        movie0_.BUDGET as budget2_0_0_,
        movie0_.HERO as hero3_0_0_,
        movie0_.MNAME as mname4_0_0_
    from
        MOVIE_INFO movie0_
    where
        movie0_.mid=?
Movie:: 0-param constructorclass com.nt.entity.Movie
Movie(mid=25, mname=Pupsa, hero=allu-arjun, budget=1.0E9)
```

1st
for ses.get(Movie .class,24)
(record collected Db and kept in L1 cache)

for 2nd ses.get(Movie .class,24)
(record collected directly from L1 cache)

for ses.get(Movie.class,25);
(record collected Db and kept in L1 cache)

To control L1 cache
==========

=>To remove specific object from L1/Level1 cache
ses.evict(movie); --> takes given object's id value and removes id value based Movie class object from L1 cache

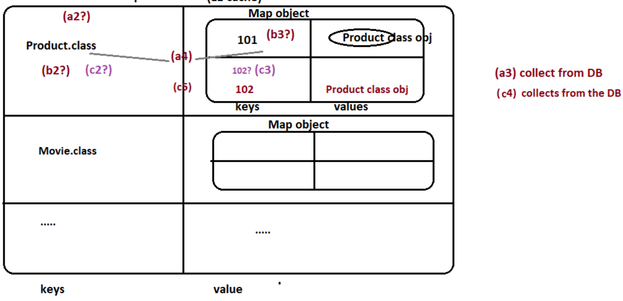
=> To empty L1 cache totally
ses.clear(); -> all objects of L1 cache will be removed

=> ses.close() ;
closes the session with Db s/w . In this process the L1 cache associated with Session obj will be emptied.

```
// Load object                                                    (24)
Movie movie=(Movie)ses.get(Movie.class, 24);  //Collects from DB and keeps  L1 cache
System.out.println(movie);
System.out.println("----------------------");                    (24)
Movie movie1=(Movie)ses.get(Movie.class, 24);  //Collects from  L1 cache
System.out.println(movie1);
System.out.println("----------------------");                    (25)                in
Movie movie2=(Movie)ses.get(Movie.class, 25);  //Collects from DB and keeps  L1 cache
System.out.println(movie2);

ses.evict(movie);  //removes Movie obj with 24 id value from L1 cache
Movie movie3=(Movie)ses.get(Movie.class, 24);  //Collects from DB  and keeps  in  L1 cache
System.out.println(movie3);                                         (24)

//To empty the L1 cache
ses.clear(); // Removes all the objects from L1 cache
Movie movie4=(Movie)ses.get(Movie.class, 24);  //Collects from DB  and keeps  in  L1 cache
System.out.println(movie4);                                         (24)
```

To get best effect from cahing empty cache by calling ses.evict(-) or ses.clear() at regular internvals through Schduling process.

=>The objects in L1 cache /Level1 cache are identified using Entity class name + id value
=> In pratical implementation the L1 and L2 caches are not just Map Collection. they Map of Map collection,,
(Map<Class, Map<Serilizable,Object>>)




```
(a5)                (a1)
Product prod=ses.get(Product.class,101);   //collects from DB and keeps  L1 cache
Product prod1=ses.get(Product.class,101);   //collects from  L1 cache
    (b4)            (b1)
```

a1 to a5 :: 1st request
b1 to b4 :: 2nd request

```
                                    (c1)
Product prod2=ses.get(Product.class,102);
 (c6)
```

=>Hibernate f/w keeps Entity object in 3 states in the execution of application i.e Hiberrnate f/w is managing the life cycle of Entity object by keepin the object: in 3 states.

1) Transient state (Not Associated with Session)

2) Persistent state (Associated with the Session)

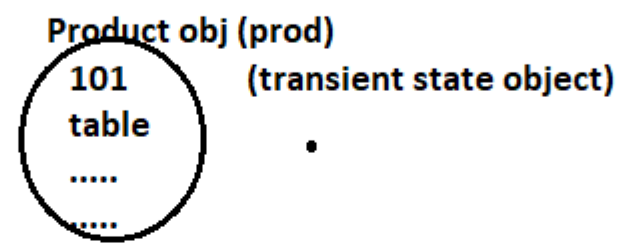
3) Detached state (Previously Persistent)

Transient state
==========
=>This is just created object for an Entity class having data
=> This object is not associated/linked with the Session
=> this object does not contain id value (identity value)

(Student enuirying about course but still no joined)

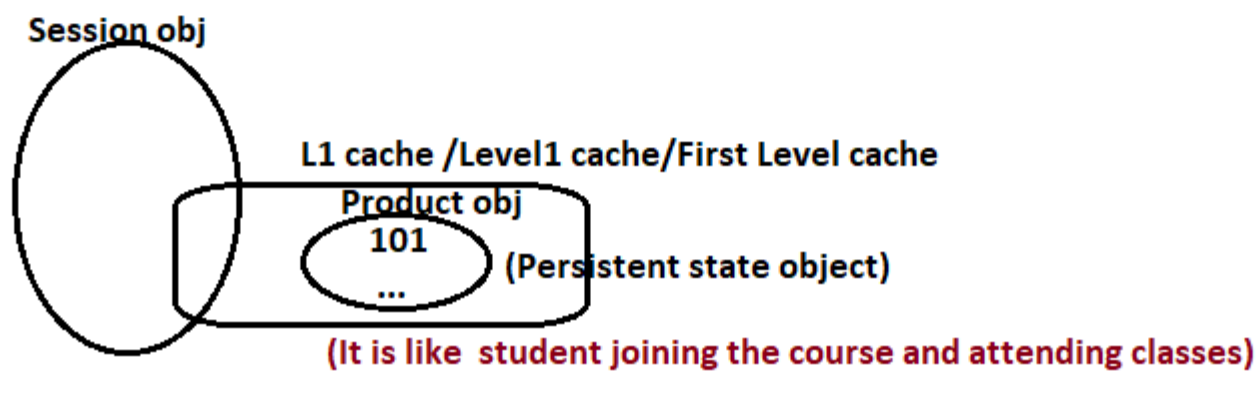
```
Product prod=new Product();
prod.setPid(1001);
....
....
```

transient state object.

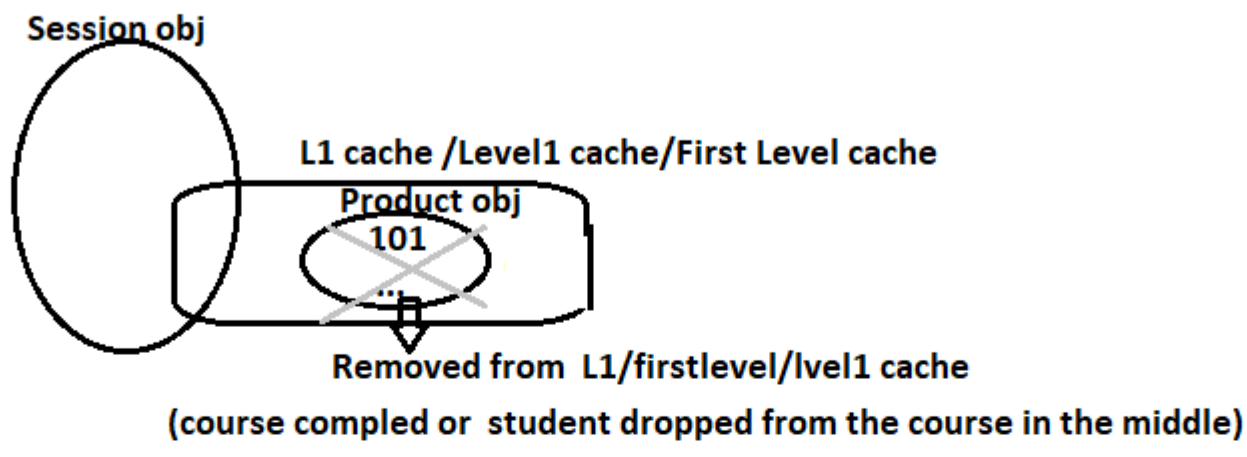


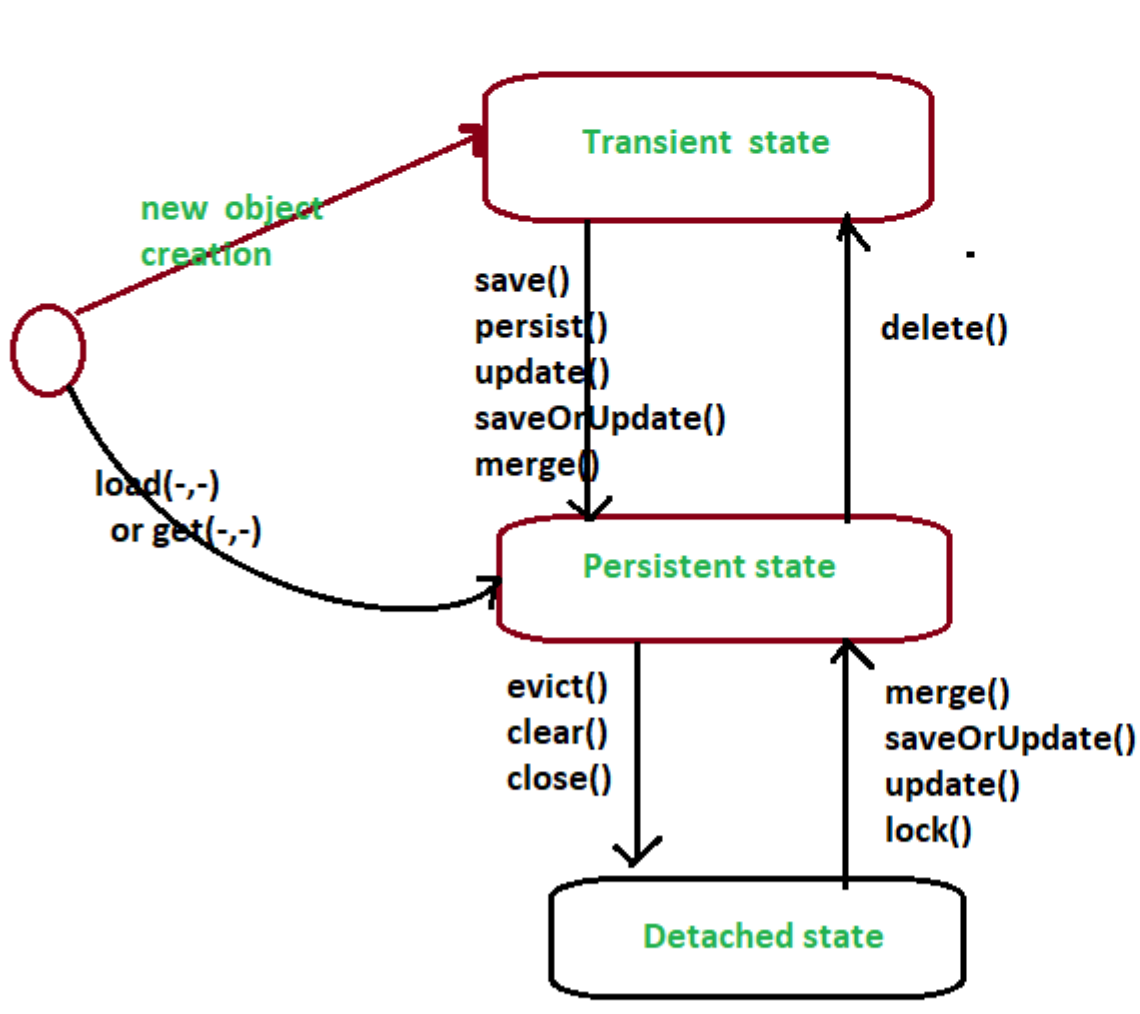
Persistent state
==========
=> Here the entity object is associated with session (stays in L1 cache of the Session)
=> This state object contains id value .. represents the db table record , So the modifications done in the object will reflect to db table record i.e synchroization with with db table record will be maintained.
=> All the persistence operations in the app takes place using this persistent state objs

```
ses.save(prod); // moves the object L1 cache until tx.commit() is called
                //i.e object is is assoicated with the session .. so we can
                // say that object is in persistent mode.

Product prod1=ses.get(Product.class,101); // this method dynamically creates the object for entity
                                          class in Persistent state itself.
```



Detached state
==========
=> This state indicates that the entity obj is previosly persistent object.
=> When entity obj is removed from the cache , when leve1/first -level1/ session cache is closed then the entity objects of that cache becomes detached state object.
=> Contains id value .. but does not represent the db table record .. So no synchronization with Db s/w





transient state :: object is created ---> but no record representation (student enquirying about the course)
persistent state :: object is given to HB session ---> represents the record (student joining in the course and attending the course)
Detached state :: Object is removed from the L1 cache of HB session obj. (Student dropping from course or course compled)

==========

Interacting with two db s/w using hibernate persistence logic

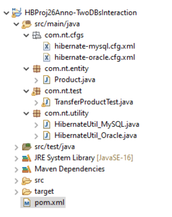
==========

=> we can make our hibernate persistence logic talking with any no.of Db s/ws ..
=>For that we need to use multiple SessionFactory objects 1 per Db s/w basis
similarly we need to take hibernate cfg files also on 1 per Db s/w basis..

Usecases1 :: transfering money from 1 bank account (DB1) to another bank account (DB2)
usecase2:: Transfering proudct from 1 inventory (DB1) to another inventory (DB2)
usecase3 :: if bank1 acquires bank2 .. then the account details of bank1(db1) should copied to bank2(db2)

Resources of the Application (Transfering Product details from oracle DB table (product) to
========== mysql db table (Product))

=> hibernate-oracle.cfg.xml
=> hibernate-mysql.cfg.xml
=> Product.java (Entity class with annotations)
=> HibernateUtil-Oracle.java
=> HibernateUtil-mysql.java
=> TransferProductTest.java (Client App)



Product.java

```
----------------
package com.nt.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import lombok.Data;

@Data
@Entity
public class Product {
        @Id
        @GeneratedValue
        private Integer pid;
        private String pname;
        private Double price;
        private Double qty;


}
```

hibernate-oracle.cfg.xml

```
-----------------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Connection properties -->
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">manager</property>

    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

    <!-- mapping file name -->
    <mapping class="com.nt.entity.Product"/>
  </session-factory>

</hibernate-configuration>
```

hibernate_mysql.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Connection properties -->
    <property name="connection.url">jdbc:mysql:///nthb412db</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>

    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>

    <!-- mapping file name -->
    <mapping class="com.nt.entity.Product"/>
  </session-factory>

</hibernate-configuration>
```

Utility class for mysql

```
public class HibernateUtil_MySQL {
        private static SessionFactory factory;
        static {
                //create Configuration object
                Configuration cfg=new Configuration();
                //Load and read HB configuration file
                cfg.configure("com/nt/cfgs/hibernate-mysql.cfg.xml");
                //build SessionFactory
                factory=cfg.buildSessionFactory();
        }

        //static factory method for SessionFactory object
        public static SessionFactory getSessionFactory() {
                        return factory;
        }

        //static factory method for session obj
        public static Session getSession() {
                        Session ses=null;
                        if(factory!=null)
                         ses=factory.openSession();
                        return ses;
        }
```

Utility class for oracle

```
public class HibernateUtil_Oracle {
        private static SessionFactory factory;
        static {
                //create Configuration object
                Configuration cfg=new Configuration();
                //Load and read HB configuration file
                cfg.configure("com/nt/cfgs/hibernate-oracle.cfg.xml");
                //build SessionFactory
                factory=cfg.buildSessionFactory();
        }

        //static factory method for SessionFactory object
        public static SessionFactory getSessionFactory() {
                        return factory;
        }

        //static factory method for session obj
        public static Session getSession() {
                        Session ses=null;
                        if(factory!=null)
                         ses=factory.openSession();
                        return ses;
        }
}
```

```
}
```

Client app

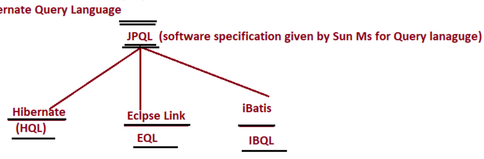
```
=========

public class TransferProductTest {

        public static void main(String[] args) {
                SessionFactory oracleFactory=HibernateUtil_Oracle.getSessionFactory();
                SessionFactory mysqlFactory=HibernateUtil_MySQL.getSessionFactory();
                Session oracleSes=HibernateUtil_Oracle.getSession();
                Session mysqlSes=HibernateUtil_MySQL.getSession();
                Transaction mysqlTx=null;
                try(oracleFactory;mysqlFactory;oracleSes;mysqlSes){
                        //get Object /record from oracle Db table
                        Product prod=oracleSes.get(Product.class,28);
                        //save the object/record into mysql db table
                        mysqlTx=mysqlSes.beginTransaction();
                          int idVal=(int) mysqlSes.save(prod);
        mysqlTx.commit();
        System.out.println("Record/Object is transffered from oracle to mysql Db table");
                }
                catch(Exception e) {

                        if(mysqlTx!=null && mysqlTx.getStatus()!=null && mysqlTx.getRollbackOnly()) {
                                mysqlTx.rollback();
        System.out.println("Problem in transffering record");
                        }//if
                         e.printStackTrace();
                }//catch
        }//main
}//class
```

Bulk Operations in hibernate

=> So far we worked with single row operations i.e we are manipulating one record at a time
=> if u want to manupulate 1 or more records at a time or to manipulate single record with our choice condition .. we definetly should go for these bulk operations

The options for bulk operations are
a) HQL/JPQL Queries (2)
b) Native SQL Queries (3)
c) JPA Criteria API (1)

JPQL --> Java Persistence Query language
HQL --> Hibernate Query Language



HQL :: Hibernate Query Lanaguge
EQL :: Eclipse Query Language
IBQL :: IBatis Query Language

JPQL ==> theory (planning/specification)
HQL ==> Hiberante's Impl/reality for JPQL

=>ORM /JPA => Theory /planning/specification
=>hibenrate/ibatis/toplink/eclipse link and etc.. are pratical implementations.

=>HQL/JPQL is Objects based Query lanaguage
=> we write SQL queries using db table and cols where as we write HQL/JPQL queries using entity class name and its properties/attributes.
=>HQL/JPQL queries Db s/w independent queries where SQL Queries are DB s/w dependent queries(30%)
=>HQL/JPQL Queries support both select and non-select operations
=>HQL/JPQL Queries support both ordinal positional params(?1,?2,?3) and named params (:<name>)
=>HQL/JPQL Queries allows different condition caluses
=>HQL/JPQL Queries allows joins
=>HQL/JPQL Queries allows sub queries
=>HQL/JPQL Queries support aggragate operations
=> HQL/JPQL Queries learning curve is very small i.e we can learn in less time
=>HQL/JPQL Query will be converted into SQL Query internally by using one or another translator

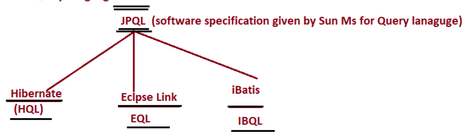
<= HQL /JPQL based DDL Operations(create table, drop table, alter table,..) are not possible
=> HQL /JPQL based INSERT Query is not possible
=> HQL /JPQL based PL/SQL programming is not possible.

Bulk Operations in hibernate

=> So far we worked with single row operations i.e we are manipulating one record at a time
=> if u want to manupulate 1 or more records at a time or to manipulate single record with our choice condition .. we definetly should go for these bulk operations

The options for bulk operations are
a) HQL/JPQL Queries (2)
b) Native SQL Queries (3)
c) JPA Criteria API (1)

JPQL --> Java Persistence Query language
HQL --> Hibernate Query Language



HQL :: Hibernate Query Lanaguge
EQL :: Eclipse Query Language
IBQL :: IBatis Query Language

JPQL ==> theory (planning/specification)
HQL ==> Hiberante's Impl/reality for JPQL

=>ORM /JPA => Theory /planning/specification
=>hibenrate/ibatis/toplink/eclipse link and etc.. are pratical implementations.

=>HQL/JPQL is Objects based Query lanaguage
=> we write SQL queries using db table and cols where as we write HQL/JPQL queries using entity class name and its properties/attributes.
=>HQL/JPQL queries Db s/w independent queries where SQL Queries are DB s/w dependent queries(30%)
=>HQL/JPQL Queries support both select and non-select operations
=>HQL/JPQL Queries support both ordinal positional params(?1,?2,?3) and named params (:<name>)
=>HQL/JPQL Queries allows different condition caluses
=>HQL/JPQL Queries allows joins
=>HQL/JPQL Queries allows sub queries
=>HQL/JPQL Queries support aggragate operations
=> HQL/JPQL Queries learning curve is very small i.e we can learn in less time
=>HQL/JPQL Query will be converted into SQL Query internally by using one or another translator

<= HQL /JPQL based DDL Operations(create table, drop table, alter table,..) are not possible
=> HQL /JPQL based INSERT Query is not possible
=> HQL /JPQL based PL/SQL programming is not possible.

SQL> SEELCT * FROM PRODUCT
Entity class name db table
HQL> from Product (or)
HQL> select p from Product as p
alias name

HQL> from Product p
HQL> from Product as p

if the HQL select Query is selecting all the cols of db table then placing "select" keyword in HQL select Query is optional

---

SQL> SELECT PID,PNAME FROM PRODUCT WHERE PRICE>=? AND PRICE<=?
col names db table col name col name
HQL> select pid,pname from Product where price>=? and price<=?
property names Entity class name property name property name
HQL> select p.pid , p.pname from Product as p where p.price>=? and p.price<=?
property names Entity class name alias name

if ur selecting specific one column or multiple cols from db table using HQL/JPQL select Query then specifying "SELECT" keyword in the HQL select query is mandatory

---

=>In SQL Queries, the sql keywords , table names, col names are not case-sensitive
=> In HQL/JPQL Queries , the HQL keywords are not case-sensitive ..but the entity class names and property names are case-sesintive.

SQL> UPDATE PRODUCT SET PRICE=PRICE+? WHERE PID=?
db table name col name col name
HQL> update Product set price=price+? where pid=?
Entity class name property name

=>Query object represents each HQL/JPQL Query and we need to use list() method or iterate() on Query object to execute the SELECT HQL Query and executeUpdate() method to execute the non-select HQL Query.

=> Query object , it is the object of a class that implements org.hibernate.query.Query(I).

Two types of HQL/JPQL Select Queries

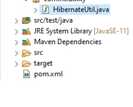
=>Selects all the col values of db table
eg: from Product

=> selects specific single or multiple col values of db table
=> select query performing aggregate operations also comes under scalar query
eg: select pid,pname from Product

=>Select HQL/JPQL queries can be executed as non-transctional queries .. but non-select HQL/JPQ Queries must be executed as transactional queries.

Example App
==========

dependencies in pom.xml :: lombok api , ojdbc8 , hibernate-core



hibernate.cfg.xml
---
same as first app

Product .hbm.xml
---
same as first App

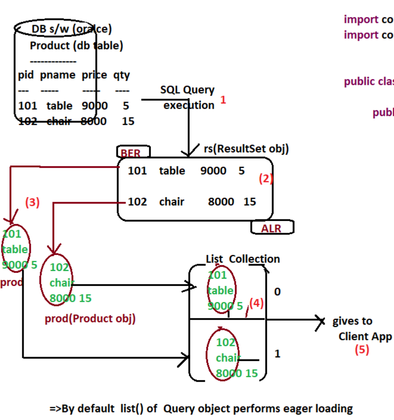
Product.java
---
same first app

HibernateUtil.java
---
same as previous apps

HQLSelectTest1.java

```
//HQLSelectTest1.java
package com.nt.test;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class HQLSelectTest1 {

    public static void main(String[] args) {
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();
        try(factory;ses){
          //prepare Query object having HQL/JPQL Query
          Query query=ses.createQuery("from Product");
          List<Product> list=query.list();  //execute the HQL Query
          //process the Result
          /*    list.forEach(prod->{          //java 8  forEach(-) method
                        System.out.println(prod);
                    });*/
          //list.forEach(System.out::println);   //java8  forEach(-) method
                                                                    + method reference
            for(Product p:list) {  // enhanced for loop
                System.out.println(p);
            }

        }//try
        catch(Exception e) {
            e.printStackTrace();
        }

    }

}
```

execution of list() method Query obj
==========



=>By default list() of Query object performs eager loading

=>The list() of Query object performs eager loading irrespective of the lazy="true" is taken in the "lazy" attrubute of <calss> tag.
=>For lazy loading while working HQL Queries , we ned to worki with Query.iterate() method.

=>Query.list() executes HQL select Query having eager loading where as Query.iterate() exeutes the same HQL Query having lazy loading.

example on Query.list() method [Works only in hibernate 4.x]

```
Query query=ses.createQuery("from Product");
        Iterator<Product> it=query.iterate();
        while(it.hasNext()) {
                Product prod=it.next();
                System.out.println(prod.getPid()+"  "+prod.getPname()+"  "+prod.getPrice()+"  "+prod.getQty());
        }
```

=>Generate first select Query getting only PK column values(id values) of all reords and generate multiple select queries sumbutting those id values to get other record on one query per each record basis as show below.

PRODUCT

Columns | Data | Model | Constraints | Grants | Statistic

Sort.. | Filter:

	PID	PNAME	PRICE	QTY
1	23	chair	1368.5	12
2	24	sofa	1368.5	12
3	25	keyboard	3456	666
4	26	dinning	434242	34

```
select   PID from  Product
   23
   24
   25
   26
                                  from Product
select  pid,pname,price,qty            where pid=23
   23 chair  1368.5  12
                                  from Product
select  pid,pname,price,qty            where pid=24
   24  sofa  1368.5  12
                                  from Product
select  pid,pname,price,qty            where pid=25
   25   keyboard  3456  12
                                  from Product
select  pid,pname,price,qty            where pid=26
   26  dining  434242  24
```

---

what is the difference b/w Query.list() and Query.iterate() method ?

Query.list()	Query.iterate()
(a) execute HQL select Query having eager loading	(a) execute HQL select Query having lazy loading
(b) Generates 1 SQL select query to fetch all the records	(b) Generates 1 + n Select SQL Queries to fetch all the records
(c) hits the DB s/w only for 1 time to get all the records	(c) Hits the Db s/w 1+n times to get all records (n --> indicates no.of records)
(d) Returns List<T> collection	(d) returns Iterator<T> object refering List collection
(e) works in all version of hibernate perfectly	(e) works upto hibernate 4.x

---

Parameters in HQL/JPQL queries
==========

Upto 4.x
========
=>jdbc style Positional parameters (?)
=> named parameters (:<name>)

From hibernate 5.x , there is no support for jdbc style positional parameters (?)

From 5.x
========
=>Jpa style ordinal Positional parameters (?1,?2,?3,...)
=> named parameters (:<name>)

JPA Style ordinal positional Parameters (?1,?2,?3,?4 and etc...)
==========

```
//prepare Query object having HQL/JPQL Query
        Query query=ses.createQuery("from  Product where pid>=?0 and pid<=?1");
        //set query param values
        query.setParameter(0, 20);
        query.setParameter(1, 25);
         //execute the HQL Query
        List<Product>  list=query.list();
        //proess the Result
        list.forEach(prod->{
                System.out.println(prod);
        });
```

Imp points on jpa style ordinal positional params ?<n>
==========
=> Ordinal positional params can be started with any number [0 or positve number]
=> There should not be any gap between the ordinal positional params
=> Negetive numbers are not allowed in the ordinal positional params

```
from  Product where pid>=?0 and pid<=?1   //valid
from  Product where pid>=?10 and pid<=?12   //invalid   (10  , 12 --> gap  is there)
from  Product where pid>=?1 and pid<=?2   //valid
from  Product where pid>=?0 and pid<=?2   //in valid     (gap between the ordinal params)
from  Product where pid>=?-1 and pid<=?0   //invalid     (negetive numbers are not allowed)
from  Product where pid>=?20 and pid<=?21   //valid
from  Product where pid>=? and pid<=?   //invalid     (jdbc style positioanl params(?) not allowed)
                                                                                    in 5.x
from  Product where pid>=?11 and pid<=?10 //valid
from  Product where pid>=?2 and pid<=?1 //valid
```

=>instead of giving index based identity to each positional param like ?1,?2 and etc.. it is better to provide name for each parameter by taking the support of named parameters

---

HQL /JPLQ Queries with Named Parameters
==========

:<name> ---> syntax of the named parameter

```
// HQL/JPL Query With Named Parameters
        Query query=ses.createQuery("from  Product where pname in(:prod1,:prod2,:prod3) order by  pname asc");
        //set values to named parameeters
         query.setParameter("prod1","sofa");
         query.setParameter("prod2","chair");
         query.setParameter("prod3","table1");
         //execute  the query
List<Product> list=query.list();
//process the Result
list.forEach(prod->{
        System.out.println(prod);
});
```

Q) Can we put both ordinal positiolnal params and named parameters togather in HQL Query?

Q) Can we put both ordinal positiolnal params and named parameters togather in HQL Query?

yes) we can take place name parameters and jpa style ordinal positional parameters togahter in a single HQL/JPQ Queriers

```
// HQL/JPL Query With Named and ordinal positional Parameters
Query query=ses.createQuery("from Product where pname in(?1,:prod2,?2) order by pname asc");
//set values to named parameeters
query.setParameter(1,"sofa");
query.setParameter("prod2","chair");
query.setParameter(2,"table1");
//execute the query
List<Product> list=query.list();
//process the Result
list.forEach(prod->{
        System.out.println(prod);
});
```

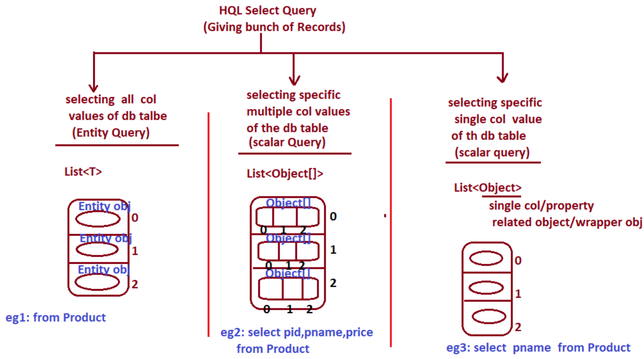
---

we can take parameters in HQL/JPQL Query only representing input value or condition values .. Not representing HQL keywords or entity class names or property names.

```
from Product where pname in(?1,?2,?3) order by pname asc   --- valid HQL Query
from :tab where pname in(?1,:prod2,?2) order by pname asc  ---- invalid HQL query
:p1 Product where pname in(?1,:prod2,?2) order by pname asc  ---invalid HQL query
?1 ?2 where ?3 in(?4,?5,?6) order by pname asc  ---> invalid HQL Query
```

=>if the HQL select query is selecting all the col values of db table then it called Enttiy HQL Sselect Query

=>if the HQL select query is selecting specific single col or multiple col values of db table then it called Scalar HQL Select Query.

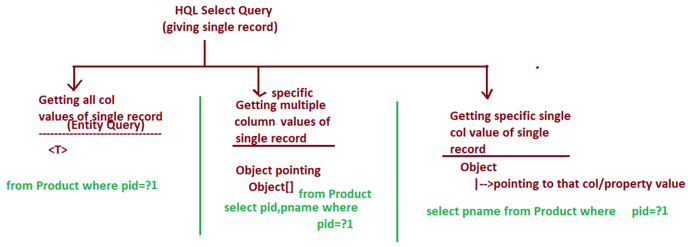



```
//executing HQL scalar select Query (retrieving specific multiple col values)
Query query=ses.createQuery("select pid,pname,price from Product where price>=:min and price<=:max");
query.setParameter("min", 1000.0f);
query.setParameter("max",30000.0f);
List<Object[]> list=query.list();
list.forEach(row->{
        for(Object val:row) {
                System.out.print(val+ " ");
        }
        System.out.println();
});
```

Generated SQL Queries and output

```
Hibernate:
  select
    product0_.PID as col_0_0_,
    product0_.PNAME as col_1_0_,
    product0_.PRICE as col_2_0_
  from
    PRODUCT product0_
  where
    product0_.PRICE>=?
    and product0_.PRICE<=?

23  chair  1368.5
24  sofa  1368.5
25  keyboard  3456.0
27  sofa1  1368.5
21  table1  1368.5
28  light  1368.5
29  sofa3  1368.5
```

```
System.out.println("----------------------------------------");
//executing HQL scalar select Query (retrieving specific single col value)
Query query1=ses.createQuery("select pname from Product where qty>=:min and qty<=:max");
query1.setParameter("min", 10.0f);
query1.setParameter("max", 20.0f);
List<Object> list1=query1.list();
list1.forEach(name->{
        System.out.println(name);
});
```




```
//executing HQL Entity select Query giving single record
Query query2=ses.createQuery(" from Product where pid=:id");
query2.setParameter("id", 25);
Product prod=(Product) query2.getSingleResult();
if(prod==null)
        System.out.println("Record not found");
else {
        System.out.println("Record found");
        System.out.println(prod);
}
```

```
//executing HQL scalar select Query giving single record multiple col values
Query query3=ses.createQuery("select pid,pname from Product where pid=:id");
query3.setParameter("id",25);
Object result=query3.getSingleResult();  //gives Object class obj pointing object[]
Object values[]=(Object[])result;  //type casting with Object[]
System.out.println(values[0]+"   "+values[1]);
//executing HQL scalar select Query giving single record single col value
Query query4=ses.createQuery("select pname from Product where pid=:id");
query4.setParameter("id",25);
String name=(String)query4.getSingleResult();
System.out.println("25 pid  pname ::"+name);
```

HQL Select Query having single aggregate function

```
//executing HQL scalar select Query giving single record single col value
Query query5=ses.createQuery("select count(*) from Product");
long count=(Long)query5.getSingleResult();
System.out.println("Records count::"+count);
```

HQL select Query having multiple aggragate functions

```
Query query6=ses.createQuery("select min(price),max(price),sum(price),avg(price) from Product");
Object results[]=(Object[])query6.getSingleResult();
System.out.println("min price ::"+results[0]);
System.out.println("max price ::"+results[1]);
System.out.println("sum of all prices ::"+results[2]);
System.out.println("avg of all prices ::"+results[3]);
```

HQL/JPQL sub query example
==========

```
Query query7=ses.createQuery("from Product  where price=(select max(price) from Product)");
List<Product> list=query7.getResultList();
list.forEach(prod->{
        System.out.println(prod);
});
```

Executing HQL/JPQL Non-Select Queries

=> These queries must be executed as Transactional statements (must be executed in a Tx (Transaction))

=> To execute each HQL/JPQL non-select Query we need to use executeUpdate(-) method. This method returns numeric value representing no.of records that are effected.

=>HQL/JPQL INSERT query is not given to insert the record ... It is given to insert the records in to one db table by selecting records from another table i.e HQL/JPQL insert can not be used to insert single record direcctly.
It is used to insert bulk records in to db table by selecting from another db table.

```
Transaction tx=null;
  try(factory;ses){
        //begin Tx
          tx=ses.beginTransaction();
           //prepare Query
          Query query=ses.createQuery("update Product set price=price-:discount where price>=:minPrice  and price<=:maxPrice");
           //set values to query params
          query.setParameter("discount", 300.0f);
          query.setParameter("minPrice", 1000.0f);
          query.setParameter("maxPrice", 10000.0f);
          //execute the Query
          int count=query.executeUpdate();
          tx.commit();
          System.out.println("No.of records that are effected ::"+count);
  }//try
  catch(HibernateException he) {
        he.printStackTrace();
        if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Problem in executing query");
        }//if
```

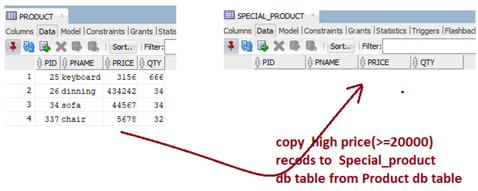
```
Transaction tx=null;
        try(factory;ses){
                //begin Tx
                  tx=ses.beginTransaction();
                   //prepare Query
                  Query query=ses.createQuery("delete from Product where qty<=:val");
                   //set values to query params
                  query.setParameter("val", 25.0f);
                  //execute the Query
                  int count=query.executeUpdate();
                  tx.commit();
                  System.out.println("No.of records that are effected ::"+count);
        }//try
        catch(HibernateException he) {
                he.printStackTrace();
                if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                        tx.rollback();
                        System.out.println("Problem in executing query");
                }//if
```

HQL /JPQL INSERT Query

=>INSERT INTO .... VALUES ... form of HQL INSERT Query is not there to insert single record to db table ... For this activity prefer using ses.save(-) or ses.persist(-) method.

=>INSERT INTO .... SELECT ... FROM .... form of HQL INSERT Query is avaiable to insert bulk records in to one db table ... by selecting from another db table.

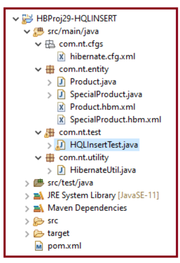
Example App
==========
INSERT special products from Product db table to Special_Product db table based on price range 10000 to 20000



copy High price(>=20000)
records to Special_product
db table from Product db table

Entity classes
=> Product (same as old apps)
|-->pid ,pname, price, qty
=>SpecialProdcut
|--->pid,pname, price ,qty

cfg file
========
hibernate.cfg.xml
|-->linked with two mapping files
(Product.hbm.xml , SpecialProduct.hbm.xml) --> mapping file
<mapping resource="com/nt/entity/Product.hbm.xml"/>
<mapping resource="com/nt/entity/SpecialProduct.hbm.xml"/>
=>HibenrateUtil.java (same as old)
=> HQLInsertTest.java (having HQL Insert Query execution logic)




```
//HQLInsertTest.java
package com.nt.test;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;


public class HQLInsertTest {

    public static void main(String[] args) {
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();

        Transaction tx=null;
        try(factory;ses){
            tx=ses.beginTransaction();   #Begin Tx
            Query query=ses.createQuery("INSERT INTO SpecialProduct(pid,pname,price,qty)
                        SELECT p.pid,p.pname,p.price,p.qty from Product as p WHERE p.price>=:min ");

            query.setParameter("min", 20000.0f);  # set query param value
            int count=query.executeUpdate();  # execute the query
            tx.commit();
            System.out.println("no.of records that are copied::"+ count);
        }
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Problem in HQL INSERT Query execution ");
            }
        }
    }//main
}//class
```

Named HQL /JPQL queries

=> To make HQL/JPQL Queries as global accessible queries across the multiple Session objects same or different client apps or DAO classes .. we need to place them in mapping file (in xml driven cfgs) or in entity class (in annotation drivencfgs) having logical name.

=>In xml driven cfgs , we need to use <query> tags to make the queries as the named queries.

=>In annotation driven cfgs , we need to use @NamedQuery annotation for the same.

Example in xml driven cfgs

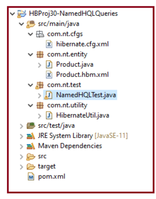
step1) Place HQL Queries in mapping file as the Named HQL Queries using multiple <query> tags

In Product.hbm.xml file (after </class> tag)

```
<query name="HQL_GET_PRODUCTS_BY_PRICE_RANGE">
    <![CDATA[ from Product where price>=:min and price<=:max ]]>
  </query>
  <query name="INCREASE_PRODUCT_PRICE_BY_PRICE_RANGE">
    <![CDATA[ update Product set price=price+:newValue where price>=:range ]]>
  </query>
```

step2) Access Named HQL Queries in Client Apps and execute them by setting the param values.

```
public class NamedHQLTest {

  public static void main(String[] args) {
      //get SessionFactory obj
      SessionFactory factory=HibernateUtil.getSessionFactory();
      //get Session obj
      Session ses=HibernateUtil.getSession();
      Transaction tx=null;
      try {
        //get Access Named HQL Select Query
            Query query=ses.getNamedQuery("HQL_GET_PRODUCTS_BY_PRICE_RANGE");
            //set values to query params
            query.setParameter("min", 1000.0f);
            query.setParameter("max",50000.0f);
            //execute the query
            List<Product> list=query.getResultList();
            //process the Result
            list.forEach(System.out::println);
            //begin Tx
             tx=ses.beginTransaction();
            //get Access Named HQL Non-Select Query
                Query query1=ses.getNamedQuery("INCREASE_PRODUCT_PRICE_BY_PRICE_RANGE");
                //set param values
                query1.setParameter("newValue", 2000.0f);
                query1.setParameter("range",40000.0f);
                //execute the query
               int count=query1.executeUpdate();
               System.out.println(" no.of record that are effected::"+count);
    //commit tx
                tx.commit();
      }//try
      catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("problem in executing non-select HQL query");
            }
      }//catch
  }//main
}//class
```



=>The HQL/JPQL Queries that are created using ses.createQuery(-) method is specific to one Session object i.e it can be used in multiple session objs of sinle or multiple client apps, where as the named HQL/JPQL Queries can be accessed and executed across the multiple Session objs of multiple client apps.

Named HQL/JPQL Queries in annotation driven env..

=>For this we need to use @NamedQuery annotation on the top of Entity class for multiple times

step1) Place Named HQL Queries in entity class

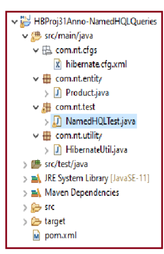
```
@Data
@Entity
@Table(name="PRODUCT")
@NamedQuery(name = "HQL_GET_PRODUCTS_BY_PRICE_RANGE",
            query="from Product where price>=:min and price<=:max")
@NamedQuery(name = "INCREASE_PRODUCT_PRICE_BY_PRICE_RANGE",
            query="update Product set price=price+:newValue where price>=:range")

public class Product {
    @Id
    @GeneratedValue
    private int pid;
    @Column(length=20)
    private String pname;
    private Float price;
    private Float qty;

    public Product() {
        System.out.println("Product:: 0-param constructor"+this.hashCode());
    }

}
```

@NamedQuery is Repeatable annotation i,e it can be placed at same level for multiple times

Repeatable annotations are introduced to java from java8 .. Since the Hibernate 4.x and 5.x are given based on java 8 .. so they support Repeatable annotations.

step2) get Access to Named HQL Queries in multiple client apps and execute them

```
public class NamedHQLTest {

  public static void main(String[] args) {
      //get SessionFactory obj
      SessionFactory factory=HibernateUtil.getSessionFactory();
      //get Session obj
      Session ses=HibernateUtil.getSession();
      Transaction tx=null;
      try {
        //get Access Named HQL Select Query
            Query query=ses.getNamedQuery("HQL_GET_PRODUCTS_BY_PRICE_RANGE");
            //set values to query params
            query.setParameter("min", 1000.0f);
            query.setParameter("max",50000.0f);
            //execute the query
            List<Product> list=query.getResultList();
            //process the Result
            list.forEach(System.out::println);
            //begin Tx
             tx=ses.beginTransaction();
            //get Access Named HQL Non-Select Query
                Query query1=ses.getNamedQuery("INCREASE_PRODUCT_PRICE_BY_PRICE_RANGE");
                //set param values
                query1.setParameter("newValue", 2000.0f);
                query1.setParameter("range",40000.0f);
                //execute the query
               int count=query1.executeUpdate();
               System.out.println(" no.of record that are effected::"+count);
    //commit tx
                tx.commit();
      }//try
      catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("problem in executing non-select HQL query");
            }
      }//catch

  }//main
}//class
```



Advantages of Named HQL Queries

(a) Make HQL/JPQL Query to execute across the multiple session objs of same or different client apps or DAO class

(b) Indirectly makes the named HQL/JPQL Query as Global HQL/JPQL Query having logical name

(c) Gives flexiblity of modification with out distrubing the source code of java apps.

Pagination usng HQL/JPQL

=>Process of displaying huge no.of records in multiple pages page by page is called pagination

eg:: gmail inbox , google search results , sales report , inventory report and etc...

=>Pagination concept is very useful in report generation..

=>Performing pagination using SQL Queries is very complex process .. we need to use sub queries with psuedo column "rownum" for this.

```
SELECT outer.*
  FROM (SELECT ROWNUM rn, inner.*
        FROM ( SELECT e.*
               FROM employee e
             ORDER BY job) inner) outer
  WHERE outer.rn >= 5 AND outer.rn <= 10
```

=>rownum or ROWNUM is a psuedo column(automaticaly generated invisible column) of oracle db table which holds record numbers

=> Hibernate simplifies pagingation by making the programmer to supply only start position and page size details in every page by calling two methods

1) query.setFirstResult(-) :: To specify start position of record in every page

2) query.setMaxResult(-) :: To specify page size in every page.

Pagination usng HQL/JPQL

=>Process of displaying huge no.of records in multiple pages page by page is called pagination

eg:: gmail inbox , google search results , sales report , inventory report and etc...

=>Pagination concept is very useful in report generation..

=>Performing pagination using SQL Queries is very complex process .. we need to use sub queries with psuedo column "rownum" for this.

```
SELECT outer.*
  FROM (SELECT ROWNUM rn, inner.*
        FROM (  SELECT e.*
                FROM employee e
            ORDER BY job) inner) outer
  WHERE outer.rn >= 5 AND outer.rn <= 10
```

=>rownum or ROWNUM is a psuedo column(automaticaly generated invisible column) of oracle db table which holds record numbers

=> Hibernate simplifies pagingation by making the programmer to supply only start position and page size details in every page by calling two methods

1) query.setFirstResult(-) :: To specify start position of record in every page
2) query.setMaxResult(-) :: To specify page size in every page.

To perform pagination in hibernate

query.setFirstResult(0); query.setMaxResults(5);	Gives 1 to 5 records	1 -->strart Pos 5 --> page Size	page1
query.setFirstResult(5); query.setMaxResult(5);	Gives 6 to 10 records	5 -->strart Pos 5 --> page Size	page2
query.setFirstResult(10); query.setMaxResult(5);	Gives 11 to 15 records	10 -> startPos 5 --> pageSize	page3

=>page size(no.of records per page) is fixed across the multiple pages but startPosition of cursor (FirstResult) will change time to time.

```
public class PaginationTest {

    public static void main(String[] args) {
        //get SessionFactory obj
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session obj
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try {
          //create  HQL Query
                Query query=ses.createQuery("from Product");
                query.setFirstResult(0);  //start Pos   //chage this value to 3,6,9,...
                query.setMaxResults(3);  //page Size
                //execute the query
                List<Product> list=query.getResultList();
                //process the Result
                list.forEach(System.out::println);
        }//try
        catch(HibernateException he) {
              he.printStackTrace();
        }//catch

    }//main
}//class
```

To get no.of page sizes::

```
int pagesCount= total records/pageSize;
    if(totalRecords%pageSize!=0)
        pagesCount++;
```

case1:: pageSize: 5
totalRecords : 20
pagesCount=20/5 = 4
since 20%5==0 the pagesCount remains 4 itself

case2: pageSize : 5
total records : 21
pagesCount= 21/5 = 4
since 21%5!=0 then pagesCount++ (4++ =5)

To get Start Position in every page

startPosition = (pageNO *pageSize) - pageSize

```
StartPosition for page1 :: (1 * 5) -  5    =0
StartPosition for page2 :: (2 * 5) -  5    = 5
StartPosition for page3 :: (3 * 5) -  5    = 10
StartPosition for page4 :: (4 * 5) -  5    = 15
                           ... ... ...  ... ....
```

PaginationTest1 .*java*

```
package com.nt.test;

import java.util.List;
import java.util.Scanner;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.query.Query;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class PaginationTest1 {

    public static void main(String[] args) {
        //read inputs  from enduser
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter PageSize::");
        int pageSize=sc.nextInt();

        //get SessionFactory obj
        SessionFactory factory=HibernateUtil.getSessionFactory();
        //get Session obj
        Session ses=HibernateUtil.getSession();
        try {
            //get  total recods in the db table
            Query query1=ses.createQuery("select count(*) from Product");
            long totalRecords=(long) query1.getSingleResult();
            long pagesCount=totalRecords/pageSize;
              if(totalRecords%pageSize!=0)
                  pagesCount++;

          //create  HQL Query for pagination
            Query query2=ses.createQuery("from Product");
             for(int pageNo=1;pageNo<=pagesCount;++pageNo) {
                  System.out.println(pageNo+"/"+pagesCount+" page records ");
                   //get startPosition for each page
                  int startPosition=(pageNo*pageSize)-pageSize;
                  // set pagination params
                  query2.setFirstResult(startPosition);
                  query2.setMaxResults(pageSize);
                  //execute the Query
                  List<Product> list=query2.getResultList();
        list.forEach(System.out::println);
        System.out.println("-----------------------");
                }//for
        }//try
        catch(HibernateException he) {
              he.printStackTrace();
        }//catch

    }//main
}//class
```

Pagintation as web based layered Application

```
                                (servlet/servlet filter)        (java class)          (java class)
view(html/jsp) --------------> controllerServlet --------------> service class -------------> DAO class --------> DB s/w
(presentation logic)            (Monitoring logic)              (b.logic)             (persistnece logic)      (Data)
```

=>Layered App says seperate different logics to different comps and make them taking to each other

=> A Layer is logical partiation of the Project/application having ability to seperate logics from other logics of other layers and makes the logics as flexible logics to modify.

=>application with out layers (i.e all logics are mixedup in single comp) is like Bachelor room

=> application with layers is like 3bhk/4bhk flat.

DAO class (Data Access object class)

=> java class that contains only persistnece logic (CURD operations logic)

=> It is generally taken on 1 per db table basis take

if project is having<100 db tables then we 1 DAO per 1 db table 4/5

if project is having >100 db tables then we take 1 DAO per related db tables

=> DAO class contains two parts

a) Query part (At the top of the DAO class as String constant variable values)

b) Code Part (Inside the methods having persistence logic)

Service class

=> contains b.logic representing calculations, sortings, orderings, comparisions, analyzationz filterings and etc..

=> It is taken on 1 per module basis

Contorller servlet

=> It is either servlet or servlet filter comp handlng http requests and responses

=> contains monitoring logics along with common services logics logics logging, auditing and etc..

=> It is generally taken 1 per Project basis

View comps

=> Contains presentation logic

=> these comps will be taken as needed..

Pagintation as web based layered Application

view(html/jsp) ----------> (servlet/servlet filter) controllerServlet ----------> (java class) service class ----------> (java class) DAO class --------> DB s/w
(presentation logic) (Monitoring logic) (b.logic) (persistnece logic) (Data)

=>Layered App says seperate different logics to different comps and make them taking to each other

=> A Layer is logical partiation of the Project/application having ability to seperate logics from other logics of other layers and makes the logics as flexible logics to modify.

=>application with out layers (i.e all logics are mixedup in single comp) is like Bachelor room
=> application with layers is like 3bhk/4bhk flat.

DAO class (Data Access object class)
==========

=> java class that contains only persistnece logic (CURD operations logic)
=> It is generally taken on 1 per db table basis take
if project is having<100 db tables then we 1 DAO per 1 db table 4/5
if project is having >100 db tables then we take 1 DAO per related db tables

=> DAO class contains two parts
a) Query part (At the top of the DAO class as String constant variable values)
b) Code Part (Inside the methods having persistence logic)

Service class
==========

=> contains b.logic representing calculations, sortings, orderings, comparisions, analyzationz filterings and etc..
=> It is taken on 1 per module basis

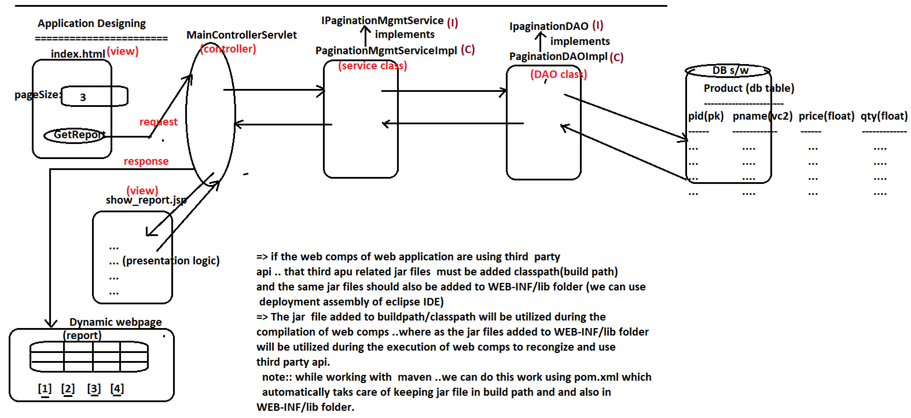
Contorller servlet
==========

=> It is either servlet or servlet filter comp handlng http requests and responses
=> contains monitoring logics along with common services logics logics logging, auditing and etc..
=> It is generally taken 1 per Project basis

View comps
==========

=> Contains presentation logic
=> these comps will be taken as needed..

Application Designing
==========



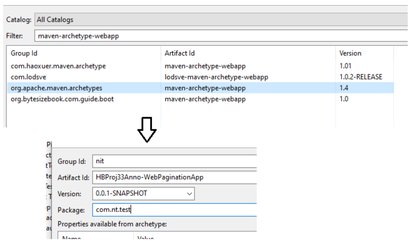
=> if the web comps of web application are using third party api .. that third apu related jar files must be added classpath(build path) and the same jar files should also be added to WEB-INF/lib folder (we can use deployment assembly of eclipse IDE)
=> The jar file added to buildpath/classpath will be utilized during the compilation of web comps ..where as the jar files added to WEB-INF/lib folder will be utilized during the execution of web comps to recongize and use third party api.
note:: while working with maven ..we can do this work using pom.xml which automatically taks care of keeping jar file in build path and and also in WEB-INF/lib folder.

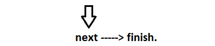
Procedure to develop web application in eclipse IDE using Maven
==========

maven archetype --> maven-archetype-quickstart --> for standalone Projects
maven archetype --> maven-archetype-webapp --> for web application Projects
(Project template giving readmade directory strucutre for different applications)

step1) create Maven Project by using the archetype "maven-archetype-webapp"

File menu ---> maven project ---->select archetype (maven-archetype-webapp) -->





next -----> finish.

step2) Perform the following operations on the Project

a) change the java version in pom.xml file
open pom.xml file ---> change java version 7 to 11

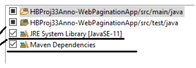
```
<maven.compiler.source>7</maven.compiler.source>
<maven.compiler.target>7</maven.compiler.target>
```

11 11

Right click on Project ----> maven --> update project.

b) enable few source directories

right click on project ---> build path -->
configure build path --->order and export tab --->select the empty two checkboxes

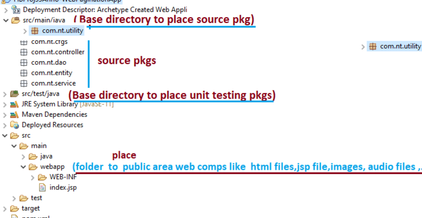


c) add following dependencies in pom.xml file

```
<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
<dependency>
  <groupId>javax.servlet</groupId>
  <artifactId>javax.servlet-api</artifactId>
  <version>4.0.1</version>
  <scope>provided</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.6.5.Final</version>
</dependency>

<!-- https://mvnrepository.com/artifact/com.oracle.ojdbc/ojdbc8 -->
<dependency>
  <groupId>com.oracle.ojdbc</groupId>
  <artifactId>ojdbc8</artifactId>
  <version>19.3.0.0</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.projectlombok/lombok -->
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <version>1.18.22</version>
</dependency>
```

collect these dependencies from mvnrepository.com

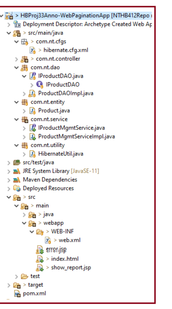
step3) Add the following directories to the Project



=> hibernate.cfg.xml (same as previous)
=> Product.java (Entity class)

ProductDAOImpl.java

```
package com.nt.dao;

import java.util.List;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.query.Query;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class ProductDAOImpl implements IProductDAO {

	@Override
	public long getTotalRecordsCount() {
		Session ses=HibernateUtil.getSession();
		try(ses){
			//prepare and execute HQL Select Query
			Query query=ses.createQuery("select count(*)  from Product");
			long count=(long)query.getSingleResult();
			return count;
		}
		catch(HibernateException he) {
			he.printStackTrace();
			throw he;
		}
	}//class

	@Override
	public List<Product> getPageData(int startPosition, int pageSize) {
		Session ses=HibernateUtil.getSession();
		try(ses){
				//prepare and execute HQL Select Query
				Query query=ses.createQuery(" from Product");
			//set pagination params
				query.setFirstResult(startPosition);
				query.setMaxResults(pageSize);
			//execute the query
				List<Product> list=query.getResultList();
				return list;
		}//try
		catch(HibernateException he) {
			he.printStackTrace();
			throw he;
		}
	}//method

}//class
```



ProductMgmtServiceImpl.java

```
package com.nt.service;

import java.util.List;

import com.nt.dao.IProductDAO;
import com.nt.dao.ProductDAOImpl;
import com.nt.entity.Product;

public class ProductMgmtServiceImpl implements IProductMgmtService {

	private IProductDAO  dao;
		public ProductMgmtServiceImpl() {
			dao=new ProductDAOImpl();  //DAO class object
		}

		@Override
		public long getPagesCount(int pageSize) {
			//get total  no.of records
			long recordsCount=dao.getTotalRecordsCount();
			//get PagesCount
			long pagesCount=recordsCount/pageSize;
			if(recordsCount%pageSize!=0)
				pagesCount++;
			return pagesCount;
		}

		public  List<Product>   getPageData(int pageNo,int pageSize){
			//find out startPosition for pagination
			 int startPosition=(pageNo*pageSize)-pageSize;
			 //use DAO to get the given page Data
			 List<Product> list=dao.getPageData(startPosition, pageSize);
			 return list;
		}

	}
```

ControllerServlet

```
package com.nt.controller;

import java.io.IOException;
import java.util.List;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import com.nt.service.IProductMgmtService;
import com.nt.service.ProductMgmtServiceImpl;
import com.nt.entity.Product;

@WebServlet("/controller")
public class MainControllerServlet  extends HttpServlet{
	private IProductMgmtService service;

	public void init() {
		service=new ProductMgmtServiceImpl();
	}

	@Override
	public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
			String pageSize=req.getParameter("pageSize");
			String pgNo=req.getParameter("pgNo");
			 HttpSession ses=req.getSession();
			 int pageNo=0;
			if(pageSize!=null) { //if submit button in clicked
				ses.setAttribute("pageSize", pageSize);
				pgNo="1";
				pageNo=Integer.parseInt(pgNo);
			}
			else { //if hyper link is cliked
				pageNo=Integer.parseInt(pgNo);
				 if(ses.getAttribute("pageSize")!=null)
				  pageSize=(String)ses.getAttribute("pageSize");
				 else
					pageSize="3"; //defaultPageSize is 3
			}


			//use service
			try {
				long pagesCount=service.getPagesCount(Integer.parseInt(pageSize));
				List<Product> list= service.getPageData(pageNo, Integer.parseInt(pageSize));
				//keep results in request scope
				req.setAttribute("pageData", list);
				req.setAttribute("pagesCount",pagesCount);
				//forward to show_report.jsp
				RequestDispatcher rd=req.getRequestDispatcher("/show_report.jsp");
					rd.forward(req, res);
			}
			catch(Exception e) {
				e.printStackTrace();
				req.setAttribute("errMsg", e.getMessage());
				RequestDispatcher rd=req.getRequestDispatcher("/error.jsp");
				rd.forward(req, res);
			}
	} //doGet(-,-)

	@Override
	public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
		doGet(req,res);
	}  //doPost(-,-)

}//class
```

show_report.jsp

```
<%@page  isELIgnored="false" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core"  prefix="c" %>

<c:choose>
  <c:when test="${!empty pageData}">
     <h1 style="color:blue;text-align:center"> The Report is </h1>
     <table border="1" bgcolor="cyan"  align="center">
        <tr>
          <th>PID</th> <th> PNAME </th> <th> PRICE </th> <th> QTY </th>
        </tr>
        <c:forEach  var="prod" items="${pageData}">
           <tr>
             <td>${prod.pid} </td>
             <td>${prod.pname} </td>
             <td>${prod.price} </td>
             <td>${prod.qty} </td>
           </tr>
        </c:forEach>
     </table>
     <br>
    <p align="center">
     <c:forEach var="i" begin="1"  end="${pagesCount}" step="1">
        <a href="controller?pgNo=${i}&s1=link">${i}</a>     
     </c:forEach>
     </p>
  </c:when>

  <c:otherwise>
     <h1 style="color:red;text-align:center">Records not found </h1>
  </c:otherwise>

</c:choose>
<br><br>
<a href="index.html">home</a>
```

error.jsp

```
<%@ page isELIgnored="false" %>
<h1 style="color:red;text-align:center">Problem in ReportGeneration:: ${errMsg}</h1>
<hr>
<a href="index.html">home</a>
```

index.html

```
<h1  style="color:red;text-align:center">  Pagination Example App</h1>

<form action="controller" method="POST">
  <table border="1" align="center" bgcolor="cyan">
     <tr>
      <td> Enter Page size :: </td>
      <td> <input type="text" name="pageSize"> </td>
     </tr>
     <tr>
       <td colspan="2"><input type="submit"  name="s1" value="GetReport"/> </td>
     </tr>
  </table>
</form>
```

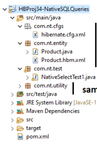
Native SQL Queries

=>These are DB s/w specific SQL Queries
=>if any operation is not possible with HQL/JPQL Queries then we need to use Native SQL Queries
=>These queries based persistence logic is DB s/w dependent persistence logic
=>Using these queries we can perform insert operation
=> Supports both positional and named parameters
=> we can work with two types of positional params
a) jdbc style positional params(?,?,?,...)
b) jpa style ordinal positional params(?1,?2,?3,....)
=> Using these Queries, we can call PL/SQL procedures and functions.
=> Each Native SQL Query is represented by one NativeQuery object(new) or SQLQuery obj(old)
=> To execute native SQL select Query use list() method ,similarly to execute native SQL non-select Query use executeUpdate() method

=> The Entity query(query selecting all col vlaues of the db table) results must be mapped with Entity class in order to get results in the form of Entity objects other wise same records will come in the form Object class object[]

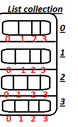
=> These queries goes and execute in Db s/w directly with minimal conversion becoz Db s/w understands SQL Queries i.e these queries give good performence.

=>These Queries will be written by using db tables and columns.



NativeSelect1.java
==========

```
//NativeSelectTest1.java
package com.nt.test;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.internal.build.AllowSysOut;
import org.hibernate.query.NativeQuery;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class NativeSelectTest1 {

    public static void main(String[] args) {
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();

        try(factory;ses){

            NativeQuery query=ses.createNativeQuery("SELECT * FROM PRODUCT");
            List<Object[]> list=query.getResultList();
            list.forEach(record->{
                for(Object val:record) {
                    System.out.print(val+" ");
                }//for
                System.out.println();
            });

        }//try
        catch(Exception e) {
            e.printStackTrace();
        }

    }//main
}//class
```



Since the results of entity query not mapped with Entity class So we get each as the Object class Object[]

```
NativeQuery query=ses.createNativeQuery("SELECT * FROM PRODUCT");
  query.addEntity(Product.class);
  List<Product> list=query.getResultList();
  list.forEach(prod->{
    System.out.println(prod.getPid()+" "+prod.getPname()+" "+prod.getPrice()+" "+prod.getQty());
  });
```

Since the Entity Query results are mapped with Entity class we records as Entity class objs

==========
Executing Native SQL Scalar Queries
==========

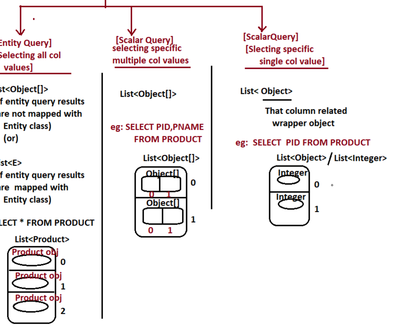
=>The Native SQL select Queries that gets specific single or multiple col values or aggragate results from db table is called Native SQL scalar query.

=>The scalar select query reuslts must be mapped hibernate data types in order to get results(col values) in controlled form
=> All hibernate data types are bridge data types b/w java data types and Db s/w data types .. these are like JDBC data types.

Hibernate type (org.hibernate.type package)	JDBC type	Java type
ShortType	SMALLINT	short, java.lang.Short
IntegerTypes	INTEGER	int, java.lang.Integer
LongType	BIGINT	long, java.lang.Long
FloatType	FLOAT	float, java.lang.Float

```
//Scalar Query (getting specific multiple col values)
    NativeQuery query=ses.createNativeQuery("SELECT PID,PRICE FROM PRODUCT WHERE PRICE BETWEEN ? AND ? ");
      //map scalar query results with hiberate data types
      query.addScalar("PID",StandardBasicTypes.INTEGER);
      query.addScalar("PRICE",StandardBasicTypes.FLOAT);
  // set query param values
      query.setParameter(1,1000.0f);
      query.setParameter(2, 50000.0f);
  //execute the Query
      List<Object[]> list=query.getResultList();
      list.forEach(row->{
          for(Object val:row) {
              System.out.print(val+" ");
          }//for
          System.out.println();
      });
```

```
//Scalar Query (getting specific single col values)
    NativeQuery query=ses.createNativeQuery("SELECT PID FROM PRODUCT WHERE QTY>=?1");
    //map scalar query results with hiberate data types
    query.addScalar("PID",StandardBasicTypes.INTEGER);
   // set query param values
    query.setParameter(1,20);
   //execute the Query
    List<Integer> list=query.getResultList();
    list.forEach(id->{
        System.out.println(id);
    });
```



HQL Supports only two types of params
a)Jpa style ordinal Positional params (?1,?2,?3,..)
b) named parameters (:<name1> ,:<name2> ,...)

Native SQL Supports 3 types of params
a)Jpa style ordinal Positional params (?1,?2,?3,..)
b) jdbc style positional params (?,?,?,...)
c) named parameters (:<name1> ,:<name2> ,...)

```
//Native SQL Entity query with named params
    NativeQuery query=
      ses.createNativeQuery("SELECT * FROM PRODUCT WHERE PRICE>=:priceRange AND QTY>=:qtyRange");
  // map entity query results with Entity class
  query.addEntity(Product.class);
  //set query param values
  query.setParameter("priceRange", 10000.0);
  query.setParameter("qtyRange", 20.0);
  //execute the Query
  List<Product> list=query.getResultList();
  //process the Results
  list.forEach(prod->{
      System.out.println(prod);
  });
```

note:: Every named parameter (:<name>) , every jpa style ordinal positional parameter (?<n>) will be converted to JDBC style positional param(?) in both HQL/Native SQL Queries before sending executing the SQL Query in Db s/w.

(?) (?<n>) :<name>

Q) Can we put all the 3 jdbc style positional params , jpa style ordina positional params and named parameters at a time our Native SQL Queries

Named Native SQL Queries
==========

=>To make Native SQL Queries as globally accessible native SQL queries across the multiple session objs 1 or more Client Apps we need to work with Named Native SQL queries

=>In xml driven cfgs , we use <sql-query> tag in mapping file to make the native SQL Query as the named native SQL Query
=> In annotation driven env.. we use @NamedNativeQuery for the same.

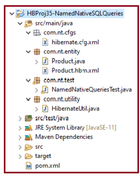
Example App in xml driven cfgs
==========

step1) place Named Native SQL Queries in mapping file
In Product.hbm.xml
---

after </class> tag

```
<sql-query name="GET_PRODS_BY_PRICE_RANGE" >
    <![CDATA[
      SELECT * FROM PRODUCT WHERE PRICE>=? AND PRICE<=?
    ]]>
  </sql-query>

  <sql-query name="HIKE_PRICE_BY_PROD_NAME">
    <![CDATA[
        UPDATE PRODUCT SET PRICE=PRICE+? WHERE PNAME=?
      ]]>
  </sql-query>
```

step2) get access to named native SQL queries and execute the Queries

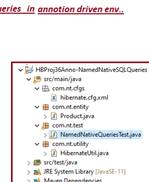



```
//NativeSelectTest1.java
package com.nt.test;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.query.NativeQuery;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class NamedNativeQueriesTest {

        public static void main(String[] args) {
                SessionFactory factory=HibernateUtil.getSessionFactory();
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(factory;ses){
                        //get access Named Native SQL Query1
                        NativeQuery query1=ses.getNamedNativeQuery("GET_PRODS_BY_PRICE_RANGE");
                        //map  Entity query results with Entity query
                        query1.addEntity(Product.class);
                        //set values to query params
                        query1.setParameter(1,1000.0f);
                        query1.setParameter(2,50000.0f);
                        //execute the Query
                        List<Product> list=query1.getResultList();
                        //process the Result
                        list.forEach(prod->{
                                System.out.println(prod);
                        });
                        System.out.println("==================================");
                        //begin the Tx
                        tx=ses.beginTransaction();
                        //get access Named Native SQL Query2
                        NativeQuery query2=ses.getNamedNativeQuery("HIKE_PRICE_BY_PROD_NAME");
                        //set query param values
                        query2.setParameter(1,200.0f);
                        query2.setParameter(2,"sofa");
                        //execute the qyery
                        int count=query2.executeUpdate();
                        System.out.println("no.of records that are effected are ::"+count);
                //commit the Tx
                        tx.commit();
                }//try
                catch(Exception e) {
                        e.printStackTrace();
                        if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                          System.out.println("Problem in Query execution");
                                tx.rollback();
                        }//if
                }//catch

        }//main
}//class
```

Making the Native SQL Queries as the Named Native SQL Queries in annotion driven env..

step1) Place Named Native SQL Queries in Entity class

Product.java




```
//Product.java (Entity class)
package com.nt.entity;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.NamedNativeQuery;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="PRODUCT")
@NamedNativeQuery(name ="GET_PRODS_BY_PRICE_RANGE",
                  query = "SELECT * FROM PRODUCT WHERE PRICE>=? AND PRICE<=?",
                  resultClass =Product.class)
@NamedNativeQuery(name ="HIKE_PRICE_BY_PROD_NAME",
                  query = "UPDATE PRODUCT SET PRICE=PRICE+? WHERE PNAME=?")


public class Product {
        @Id
        @GeneratedValue(strategy = GenerationType.AUTO)
        private  int pid;
        private String pname;
        private Float price;
        private  Float qty;

        public Product() {
                System.out.println("Product:: 0-param constructor"+this.hashCode());
        }

}
```

step2) Get Access Named Native Queries and execute the Queries

```
//NamedNativeQueriesTest.java
package com.nt.test;

import java.util.List;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.query.NativeQuery;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class NamedNativeQueriesTest {

        public static void main(String[] args) {
                SessionFactory factory=HibernateUtil.getSessionFactory();
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(factory;ses){
                        //get access Named Native SQL Query1
                        NativeQuery query1=ses.getNamedNativeQuery("GET_PRODS_BY_PRICE_RANGE");
                        //map  Entity query results with Entity query
                        query1.addEntity(Product.class);
                        //set values to query params
                        query1.setParameter(1,1000.0f);
                        query1.setParameter(2,50000.0f);
                        //execute the Query
                        List<Product> list=query1.getResultList();
                        //process the Result
                        list.forEach(prod->{
                                System.out.println(prod);
                        });
                        System.out.println("==================================");
                        //begin the Tx
                        tx=ses.beginTransaction();
                        //get access Named Native SQL Query2
                        NativeQuery query2=ses.getNamedNativeQuery("HIKE_PRICE_BY_PROD_NAME");
                        //set query param values
                        query2.setParameter(1,200.0f);
                        query2.setParameter(2,"sofa");
                        //execute the qyery
                        int count=query2.executeUpdate();
                        System.out.println("no.of records that are effected are ::"+count);
                //commit the Tx
                        tx.commit();
                }//try
                catch(Exception e) {
                        e.printStackTrace();
                        if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                          System.out.println("Problem in Query execution");
                                tx.rollback();
                        }//if
                }//catch

        }//main
}//class
```

Different options for the bulk operations
==========

(a) HQL/JPQL Queries (ii)
(b) Native SQL Queries (iii)
(c) HB criteria API (Deprecated and removed from hibernate 5.x)
(d) JPA Criteria API (alternate to HB criteria API) (i)

Criteria API (Query By Criteria--- QBC)
==========

=>It is objects based persistence logic where no queries will be written .. the entire Persistence logic will be developed using java statements.

=> This Persistence 100% java statements based persistence logic , So it is Db s/w idepenedent persistence logic

=>While developing these Persistence logic we use class names and property names

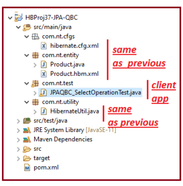
Two types of Criteria APIs
==========

a) Hibernate Criteria API -->Legacy API --> part of hibernate from the begining of hibernate
(HB QBC) ==========(outdated)
=>Just allows to perform select opeations.. i.e non-select operations are not possible
=> There is no support for this API from Hibernate 5.3 version
=> this criteria api logic is specific to hibernate framework.

b) JPA Criteria APIs --> Modren API --> given by JPA and supported by hibernate framework from
(JPA QBC) ==========
(latest and new)
-> Introduced hibernate 4.x
-> Suppots both select and non-select opeations
-> Supports both Entity select operations(all col values) and scalar select operations (specific col vlaues)
-> This criteria api can be used in multiple ORM frameworks

JPA Criteria API
==========

=> Main object or base object in this api is CriteriaBuilder object and this object can be used to create CriteriaQuery objects, Root objs , Parmaeter objs , Predicate objs, Order objs, CriteriaUpdate objects, CriteriaDelete objs and etc..




```
//JPAQBC_SelectOperationTest.java
package com.nt.test;

import java.util.List;

import javax.persistence.Query;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Product;
import com.nt.utility.HibernateUtil;

public class JPAQBC_SelectOperationTest {

    public static void main(String[] args) {
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(factory;ses){
            //create CriteriaBuilder obj
            CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
            //create CriteriaQuery object
    CriteriaQuery<Product> ctQuery=ctBuilder.createQuery(Product.class);
     // create Root object specifying the from operation
      Root<Product> root=ctQuery.from(Product.class);  //select * from Product
     //  create Query object having CriteriaQuery object
      Query query=ses.createQuery(ctQuery);
      //execute the QBC logic
      List<Product> list=query.getResultList();
      //process the result
      list.forEach(System.out::println);
        }//try
          catch(Exception e) {
              e.printStackTrace();
          }//catch
    }//main
}//class
```

Entity Query with out condition

Example2:
==========

```
    //create CriteriaBuilder obj
   CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
   //create CriteriaQuery object
CriteriaQuery<Product> ctQuery=ctBuilder.createQuery(Product.class);
// create Root object specifying the from operation
    Root<Product> root=ctQuery.from(Product.class);  //select * from Product
    //  apply  where clause condition
    ctQuery.select(root).where(ctBuilder.and(ctBuilder.ge(root.get("price"), 5000.0f),
                                                ctBuilder.le(root.get("price"),20000.0f)));
    //prepare Query object having CriterQuery obj
    Query query=ses.createQuery(ctQuery);
    //execute  the QBC logics
    List<Product> list=query.getResultList();
    //process the Result
    list.forEach(System.out::println);
```

Geneated Query

```
    Hibernate:
      select
        product0_.PID as pid1_0_,
        product0_.PNAME as pname2_0_,
        product0_.PRICE as price3_0_,
        product0_.QTY as qty4_0_
      from
        PRODUCT product0_
      where
        product0_.PRICE>=5000.0
        and product0_.PRICE<=20000.0
    Product:: 0-param constructor12269624
    Product:: 0-param constructor12269624
    Product:: 0-param constructor12269624
    Product(pid=5654, pname=mobile, price=5544.0, qty=34.0)
    Product(pid=321, pname=TV1, price=5775.0, qty=44.0)
    Product(pid=337, pname=chair, price=5678.0, qty=32.0)
```

```
    //create CriteriaBuilder obj
   CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
   //create CriteriaQuery object
CriteriaQuery<Product> ctQuery=ctBuilder.createQuery(Product.class);
// create Root object specifying the from operation
    Root<Product> root=ctQuery.from(Product.class);  //select * from Product
    //apply where clause condition
    ctQuery.select(root).where(root.get("pname").in("table","sofa","chair")).orderBy(ctBuilder.asc(root.get("price")));
                                        //adds pname in ('table','soft','chair') order By price asc
    //prepare Query object having CriterQuery obj
    Query query=ses.createQuery(ctQuery);
    //execute  the QBC logics
    List<Product> list=query.getResultList();
    //process the Result
    list.forEach(System.out::println);
```

```
   Hibernate:
     select
       product0_.PID as pid1_0_,
       product0_.PNAME as pname2_0_,
       product0_.PRICE as price3_0_,
       product0_.QTY as qty4_0_
     from
       PRODUCT product0_
     where
       product0_.PNAME in (
         ? , ? , ?
       )
     order by
       product0_.PRICE asc
   Product:: 0-param constructor12269624
   Product:: 0-param constructor12269624
   Product(pid=337, pname=chair, price=5678.0, qty=32.0)
   Product(pid=34, pname=sofa, price=49167.0, qty=34.0)
```

```
//create CriteriaBuilder obj
            CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
            //create CriteriaQuery object
        CriteriaQuery<Product> ctQuery=ctBuilder.createQuery(Product.class);
       // create Root object specifying the from operation
            Root<Product> root=ctQuery.from(Product.class);  //select * from Product
           //apply where clause condition
           ctQuery.select(root).where(ctBuilder.like(root.get("pname"), "s%"));
            //prepare Query object having CriterQuery obj
            Query query=ses.createQuery(ctQuery);
            //execute  the QBC logics
            List<Product> list=query.getResultList();
            //process the Result
            list.forEach(System.out::println);
```

```
   ibernate:
     select
       product0_.PID as pid1_0_,
       product0_.PNAME as pname2_0_,
       product0_.PRICE as price3_0_,
       product0_.QTY as qty4_0_
     from
       PRODUCT product0_
     where
       product0_.PNAME like ?
   Product:: 0-param constructor12269624
   Product:: 0-param constructor12269624
   Product(pid=345, pname=sofa1, price=54555.0, qty=44.0)
   Product(pid=34, pname=sofa, price=49167.0, qty=34.0)
```

Assingment1:: Using JPA Criteria perform select operation on db table and the Product Infomration whose qty is in the given ranage

Assignment2:: Using JPA Criteria perform all select operations to get Product who are highest price

=> we use CriteriaQuery.select(-) method for Entity select operations (getting all col values of db table)
=> we use CriteriaQuery.multiSelect(-) method for scalar select operations (getting specific multiple col values or spefic single col values of the of db table)

JPA-QBC or JAP Criteria Examples on Scalar select operations
(getting specific multiple col vlaues or single col values or aggregate results)

JPQ-QBC getting specific multiple col values of db table

```
//create CriteriaBuilder obj
CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
//create CriteriaQuery object
CriteriaQuery<Object[]> ctQuery=ctBuilder.createQuery(Object[].class);
// create Root object specifying the from operation
Root<Product> root=ctQuery.from(Product.class);  // from Product
// specify specific multiple cols and where clause conditions
ctQuery.multiselect(root.get("pid"),root.get("pname")).where(ctBuilder.and(ctBuilder.ge(root.get("price"), 5000.0f),
        ctBuilder.le(root.get("price"),20000.0f))).orderBy(ctBuilder.asc(root.get("pname")));
//select pid,pname  from Product where price>=5000  and price<=20000 order by pname asc
//  create Query object having CriteriaQuery object
Query query=ses.createQuery(ctQuery);
//execute the QBC  logic
List<Object[]> list=query.getResultList();
//process the result
list.forEach(row->{
    for(Object val:row) {
        System.out.print(val+" ");
    }
    System.out.println();
});
```

JPQ-QBC logic to get specific single col values of db table

```
//create CriteriaBuilder obj
CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
//create CriteriaQuery object
CriteriaQuery<String> ctQuery=ctBuilder.createQuery(String.class);
// create Root object specifying the from operation
Root<Product> root=ctQuery.from(Product.class);  // from Product
// specify specific multiple cols and where clause conditions
ctQuery.multiselect(root.get("pname")).where(ctBuilder.like(root.get("pname"),"s%"));
    //selectpname  from Product where pname like 's%'

//  create Query object having CriteriaQuery object
Query query=ses.createQuery(ctQuery);
//execute the QBC  logic
List<String> list=query.getResultList();
//process the result
list.forEach(System.out::println);
```

JP-QBC Logic performing aggragate function

```
//create CriteriaBuilder obj
CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
//create CriteriaQuery object
CriteriaQuery<Long> ctQuery=ctBuilder.createQuery(Long.class);
// create Root object specifying the from operation
Root<Product> root=ctQuery.from(Product.class);  // from Product
// specify specific multiple cols and where clause conditions
ctQuery.multiselect(ctBuilder.count(root.get("pid")));
    //select count(pid)  from Product
//  create Query object having CriteriaQuery object
Query query=ses.createQuery(ctQuery);
//execute the QBC  logic
Long count=(Long) query.getSingleResult();
//process the result
System.out.println("count of records ::"+count);
```

JPQBC logic to get multiple aggrate results

```
//create CriteriaBuilder obj
CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
//create CriteriaQuery object
CriteriaQuery<Object[]> ctQuery=ctBuilder.createQuery(Object[].class);
// create Root object specifying the from operation
Root<Product> root=ctQuery.from(Product.class);  // from Product
// specify specific multiple cols and where clause conditions
ctQuery.multiselect(ctBuilder.count(root.get("pid")),
                    ctBuilder.sum(root.get("price")),
                    ctBuilder.avg(root.get("price")),
                    ctBuilder.max(root.get("price")),
                    ctBuilder.min(root.get("price")));
    //select count(pid)  from Product
//  create Query object having CriteriaQuery object
Query query=ses.createQuery(ctQuery);
//execute the QBC  logic
Object result[]=(Object[])query.getSingleResult();
//process the result
System.out.println("count of records:"+result[0]);
System.out.println("sum of prices:"+result[1]);
System.out.println("avg of prices:"+result[2]);
System.out.println("max  price value:"+result[3]);
System.out.println("min  price value:"+result[4]);
```

JP-QBC supports pagination

```
//create CriteriaBuilder obj
CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
//create CriteriaQuery object
CriteriaQuery<Product> ctQuery=ctBuilder.createQuery(Product.class);
// create Root object specifying the from operation
Root<Product> root=ctQuery.from(Product.class);  // from Product
//  create Query object having CriteriaQuery object
Query query=ses.createQuery(ctQuery);
//perform pagination activities
query.setFirstResult(0);
query.setMaxResults(3);
//process the Result
List<Product> list=query.getResultList();
list.forEach(prod->{
    System.out.println(prod);
});
```

In JPA-QBC logics we need to use following objects for different operations

a) CriteriaQuery object for select operations
b) CriteriaUpdate object for update operations
c) CriteriaDelete object for Delete operations
note:: There CriteriaInsert object becoz we can not bulk insert operation
more over for single record insertion we can use ses.save(-)

Example app on CriterUpdate object
==========

```
//create CriteriaBuilder obj
CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
//create CriteriaUpdate object
CriteriaUpdate<Product> ctUpdate=ctBuilder.createCriteriaUpdate(Product.class);
// create Root object specifying the from operation
Root<Product> root=ctUpdate.from(Product.class);  // from Product
//specify values to set and the conditions to apply
ctUpdate.set("price", 20000.0f).where(ctBuilder.le(root.get("qty"),10));
//  create Query object having CriteriaUpdate object
Query query=ses.createQuery(ctUpdate);
//process the Result
tx=ses.beginTransaction();
int count=query.executeUpdate();
System.out.println("no.of records that are effected::"+count);
tx.commit();
}//try
catch(Exception e) {
    if(tx!=null || tx.getStatus()!=null || tx.getRollbackOnly()) {
        tx.rollback();
        System.out.println("Problem in update operation");
    }
```

Example app on CriteriaDelete object

```
//create CriteriaDelete object
CriteriaDelete<Product> ctDelete=ctBuilder.createCriteriaDelete(Product.class);
// create Root object specifying the from operation
Root<Product> root=ctDelete.from(Product.class);  // from Product
//specify values to set and the conditions to apply
ctDelete.where(ctBuilder.ge(root.get("price"), 50000));
//  create Query object having CriteriaDelete object
Query query=ses.createQuery(ctDelete);
//process the Result
tx=ses.beginTransaction();
int count=query.executeUpdate();
System.out.println("no.of records that are effected::"+count);
tx.commit();

}//try
catch(Exception e) {
    if(tx!=null || tx.getStatus()!=null || tx.getRollbackOnly()) {
        tx.rollback();
        System.out.println("Problem in delete operation");
    }
```

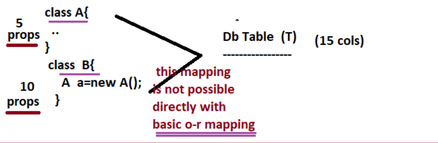
Advanced o-r mappings
==========

=>Mapping one db table with one entity class comes under basic o-r mapping..
=> The Entity classes which are java classes can be there in different relationships like HAS- relation (Composition), Inheritence relationship , association relationship like one to one , one to many , many to many ,many to one and etc.. To suport these special entity clases mapping with Db tables .. we need to use advanced o-r mappings

The advanced o-r mapping are
a) Component Mapping (if the Entity classes are in Composition(HAS-A relationship))
b) Inheritence Mapping (if the Entity classes are in Inheritence)
c) Collection Mapping (if Entity classes are having collection type(List/Set/Map) properties
d) Association mapping (if Entity classes are in association/relationship like 1 to 1 , 1 toM, M to 1 and M to M)

Problem of Granuality
==========

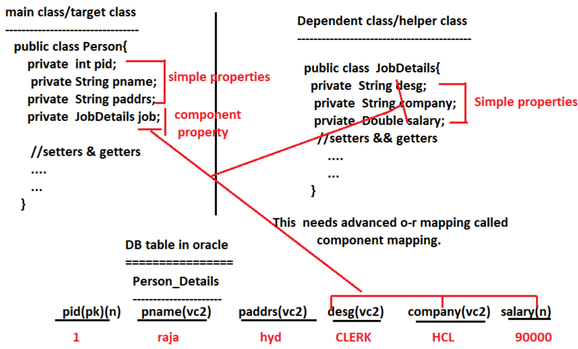
=>We can keep Entity classes in composition relation (with support HAS-A property) i.e One class maintaining other class reference variable as the property .. but we can not keep Db tables in the same composition relationship in fact such concept is not in db tables.. So we need map the classes of composition with multiple cols of single db table i.e multiple granuals (multiple pieces) should linked with single db table using basic o-r mapping .. which is not possible directly .. So that is "problem of granuality"



To overcome this problem i.e to map the classes of composition with single db table with multiple cols we need to go for advanced o-r mapping that component mapping using <componet> tag in xml driven cfgs and using @Embedded and @EmbeddableId annotations in annotation driven cfgs.

=>The property that is mapped with single col of dbtable is called simple property
For this mapping we can use <property> (xml driven cfgs) , @Column annotation (in annotation driven cfgs)
=>The property that is mapped with multiple cols of dbtable is called component property.Generally this is reference type nothing but HAS-A type property
For this mapping we can use <component> tag (xml driven cfgs) , @Embedded , @Embeddable annotations.

note:: Between Java features and RDBMS features there is lots of differences and mismatches like java suppots composition, inheritence but DB s/w does not support them .. DB s/w supports FK colums.. but In java such features are not there .. To resolve mismatches to map java classes with and db tables we need to take the support of advanced o-r mapping.



DB table in oracle
==========

Person_Details

pid(pk)(n)	pname(vc2)	paddrs(vc2)	desg(vc2)	company(vc2)	salary(n)
1	raja	hyd	CLERK	HCL	90000

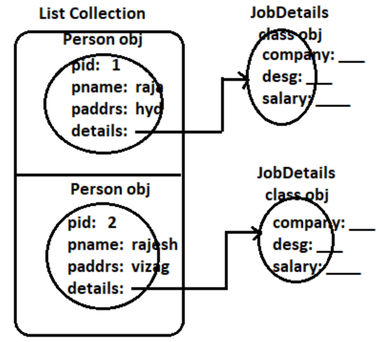
In Mapping file *(Person.hbm.xml)*
==========

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Person" table="HB_COMP_PERSON" >
    <id name="pid" column="PID">
      <generator class="increment"/>
    </id>
    <property name="pname" column="PNAME" length="20"/>
    <property name="paddrs" column="PADDRS" length="20"/>
    <component name="details" class="com.nt.entity.JobDetails">
      <property name="desg" column="DESG" length="20"/>
      <property name="company" column="COMPANY" length="20"/>
      <property name="salary" column="SALARY"/>
    </component>
  </class>
</hibernate-mapping>
```

Client App
==========

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.JobDetails;
import com.nt.entity.Person;
import com.nt.utility.HibernateUtil;

public class ComponentMappingInsertTest {

    public static void main(String[] args) {
        //create SessionFactory,Session objs
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try {
            //begin Tx
            tx=ses.beginTransaction();
              //prepare objs
              JobDetails details=new JobDetails();
              details.setDesg("CLERK"); details.setCompany("HCL"); details.setSalary(90000.0);
              Person per=new Person();
             per.setPname("rakesh");per.setPaddrs("hyd");
              per.setDetails(details);

              //save object
               int idVal=(Integer)ses.save(per);
               System.out.println("Person is saved with id value :: "+idVal);
        //commit tx
                tx.commit();
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Problem in saving data");
            }//if
        }
    }//main
}//class
```

```
//ComponentMappingSelectTest.java
package com.nt.test;

import java.util.List;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.internal.build.AllowSysOut;

import com.nt.entity.JobDetails;
import com.nt.entity.Person;
import com.nt.utility.HibernateUtil;

public class ComponentMappingSelectTest {

	public static void main(String[] args) {
		//create SessionFactory,Session objs
		SessionFactory factory=HibernateUtil.getSessionFactory();
		Session ses=HibernateUtil.getSession();
		try {
			//execute  HQL/JPQL Query
			Query query1=ses.createQuery("from Person");
			//execute the query
			List<Person> list=query1.getResultList();
			//process the Collection
			list.forEach(per->{
				System.out.println("person details ::"+per.getPid()+" "+per.getPname()+"  "+per.getPaddrs());
				JobDetails details=per.getDetails();
				System.out.println("Job Details ::"+details.getCompany()+"  "+details.getDesg()+"  "+details.getSalary());
			});

		}//try
		catch(HibernateException he) {
			he.printStackTrace();
		}
	}//main
}//class
```



Involving the sub property of main property select HQL Query condition

```
//execute  HQL/JPQL Query
Query query1=ses.createQuery("from Person where details.company=?1");
//seet query param value
query1.setParameter(1,"HCL");
//execute the query
List<Person> list=query1.getResultList();
//process the Collection
list.forEach(per->{
	System.out.println("person details ::"+per.getPid()+" "+per.getPname()+"  "+per.getPaddrs());
	JobDetails details=per.getDetails();
	System.out.println("Job Details ::"+details.getCompany()+"  "+details.getDesg()+"  "+details.getSalary());
});
```

component property.sub property

JobDetaqils.java

Annotation driven Component Mapping
==========

=> use @Embedded on the Component property and place @Embeddable on the helper class

=> @Embeddable is placed on the Helper class while working component mapping and composite Id field cfg.

=>@Embeddable says that this is not main class.. it is helper for another Entity class

=>@Embeddable classes need not to configure hibernate cfg file.

```
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;


@Data
@Table(name=" HB_COMP_PERSON")
@Entity
public class Person  implements Serializable {
	@Id
	@GeneratedValue
	private Integer pid;
	@Column(length = 20)
	private String pname;
	@Column(length = 20)
	private  String paddrs;
	@Embedded   //for component property
	private JobDetails details;

}
```

Inheritence Mapping
==========

=> We can keep entity classes in the inheritence .. but we can not keep db tables in the inheritence but we would like map the classes of inheritence with one or more db tables .. which is not possible using basic o-r mapping.this is called "sub types problem". To overcome this problem take the support of "inheritence mapping".

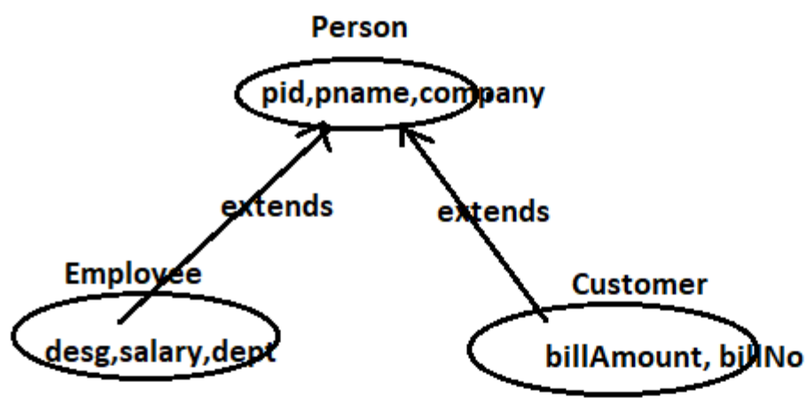
Different strategies of Inheritence Mapping
==========

a) Table per class Hierarchy (TPCH)
b) Table per sub class (TPSC)
c) Table per concrete class (TPCC)
d) Table per concrete class with Implicit Polymorphism

a) Table per class Hierarchy (TPCH)

=> All the classes of inheritence hierachy will be mapped with single db table i.e properties of super class and sub classes togather points to single db table .. For this the db table must have more number of columns.

=> In this strategy the Db table must maintain one extra column called Descriminator column to hold dirscriminator value which indicates the record in inserted using which class of the inheritence hierarchy



All these classes of inheritence hierachy will be mappd with single db table having one extra discriminator column

This called discriminator column will not have any link with Entity class properties .. It is treated as special column of this Inheritence mapping strtegy

HB_INH_PERSON (db table)
==========

pid(pk)(n)	pname(vc2)	company(vc2)	desg(vc2)	salary(float)	deptno(n)	billAmount(float)	billNo (n)	person_type(vc2)
1	raja	HCL	___	___	___	___	___	per
2	ramesh	HCL	CLERK	90000	1001	___	___	emp
3	kiran	samsung	___	___	___	9000	1212	cust

Discriminator values will be decided based on Entity classs object using which we are planning to insert the record.. These values will be specified in mapping file one 1 per class entity class basis

In xml driven cfgs

=> To cfg root class inheritence hierarchy use <class> tag
=> To cfg sub classes of the inheritence hierarchy use <subclass> tags as the sub tags of <class> tag
=> To cfg discriminator value for every entity class take the support of "discriminator-value" attribute of <class> ,<subclass> tags.

In xml driven cfgs ---------- *of Table per class hierarchy*

=> To cfg root class inheritence hierarchy use <class> tag
=> To cfg sub classes of the inheritence hierarchy use <subclass> tags as the sub tags of <class> tag
=> To cfg discriminator value for every entity class take the support of "discriminator-value" attribute of <class> ,<subclass> tags.

=> To cfg discriminator column use <discriminator> tag (after <id> tag)

super class

```
package com.nt.entity;

import lombok.Data;

@Data
public class Person {
	private Integer pid;
	private String pname;
	private String company;
	@Override
	public String toString() {
		return "Person [pid=" + pid + ", pname=" + pname + ", company=" + company + "]";
	}


}
```

sub class1

```
package com.nt.entity;

import lombok.Data;

@Data
public class Customer extends Person {
	private Double billAmt;
	private Integer billNo;

	@Override
	public String toString() {
		return "Customer [billAmt=" + billAmt + ", billNo=" + billNo + "]"+super.toString();
	}

}
```

subclass2

```
package com.nt.entity;

import lombok.Data;

@Data
public class Employee extends Person {
	private String desg;
	private Double salary;
	private Integer deptno;
	@Override
	public String toString() {
		return "Employee [desg=" + desg + ", salary=" + salary + ", deptno=" + deptno + "]-->;"+super.toString();

	}
}
```

Person.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Person" table="HB_INH_PERSON_TPCH" discriminator-value="per" >
    <id name="pid" column="PID">
      <generator class="increment"/>
    </id>
    <discriminator column="PERSON_TYPE" length="10" type="string"/>
    <property name="pname" column="PNAME" type="string"/>
    <property name="company" column="COMPANY" type="string"/>

    <subclass name="com.nt.entity.Employee" discriminator-value="emp">
      <property name="desg" column="DESG" length="10" />
      <property name="salary" column="SALARY"/>
      <property name="deptno" column="DEPTNO"/>
    </subclass>
    <subclass name="com.nt.entity.Customer" discriminator-value="cust">
      <property name="billNo" />
      <property name="billAmt"/>
    </subclass>

  </class>
</hibernate-mapping>
```

HB_INH_PERSON_TPCH

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashbacks | Dependencies | Details | Partitions | Indexes | SQL

Sort... Filter:

	PID	PERSON_TYPE	PNAME	COMPANY	DESG	SALARY	DEPTNO	BILLNO	BILLAMT
1	1	per	raja	HCL	(null)	(null)	(null)	(null)	(null)
2	2	emp	rajesh	Wipro	DEVELOPER	80000	1001	(null)	(null)
3	3	cust	karan	SAMSUNG	(null)	(null)	(null)	2345	90000

discriminator column

Limitations of Table per class hierarchy Inheritence Mapping
==========

(a) Since all the classes inheritence hierarchy are mapped with single db table .. The db table looks bit heavy and bulky
(b) Every record should maintain some null vlaues (empty values)
(c) discriminator column is mandatory to configure
(d) Since discriminator column is not mapped with any entity class property , So we should use Native SQL to work with discriminator columns
(e) It is not industry standard approach of Inheritence mapping.

(f) Designing single db table with huge no.of columns making that db table storing multiple entities data is against of Normalization rules.

hibernate.cfg.xml
same as previous

HibernateUtl.java
---
same as previous

ClientApp1
==========

```
package com.nt.test;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

import com.nt.entity.Customer;
import com.nt.entity.Employee;
import com.nt.entity.Person;
import com.nt.utility.HibernateUtil;

public class TPCH_InheritenceMappingInsertTest {

	public static void main(String[] args) {
	  // Get SessionFactory,Session objs
		SessionFactory factory=HibernateUtil.getSessionFactory();
		Session ses=HibernateUtil.getSession();
		Transaction tx=null;
		try(factory;ses){
			//prepare objects
			Person per=new Person();
			per.setPname("raja");
			per.setCompany("HCL");

			Employee emp=new Employee();
			emp.setPname("rajesh");
			emp.setCompany("Wipro");
			emp.setDesg("DEVELOPER");
			emp.setDeptno(1001);
			emp.setSalary(80000.0);

			Customer cust=new Customer();
			cust.setPname("karan");
			cust.setCompany("SAMSUNG");
			cust.setBillAmt(90000.0);
			cust.setBillNo(2345);
			 //begin Tx
			 tx=ses.beginTransaction();
			   //save objs
			  ses.save(per);
			  ses.save(emp);
			  ses.save(cust);
		tx.commit();
		System.out.println("Objects are saved");
		}//try
		catch(HibernateException he) {
			he.printStackTrace();
			if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Problem is saving objects..");
			}
		}
		catch(Exception  e) {
			e.printStackTrace();
		}
	}

}
```

client App2

```
package com.nt.test;

import java.util.List;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.nt.entity.Customer;
import com.nt.entity.Employee;
import com.nt.entity.Person;
import com.nt.utility.HibernateUtil;

public class TPCH_InheritenceMappingSelectTest {

	public static void main(String[] args) {
	  // Get SessionFactory,Session objs
		SessionFactory factory=HibernateUtil.getSessionFactory();
		Session ses=HibernateUtil.getSession();
		try(factory;ses){
	// execute HQL Query1
			Query query1= ses.createQuery("from Employee");
			List<Employee> list1=query1.getResultList();
			list1.forEach(emp->{
				System.out.println(emp);
			});
			System.out.println("================");
			Query query2= ses.createQuery("from Customer");
			List<Customer> list2=query2.getResultList();
			list2.forEach(cust->{
				System.out.println(cust);
			});
			System.out.println("================");
			Query query3= ses.createQuery("from Person");
			List<Person> list3=query3.getResultList();
			list3.forEach(per->{
				System.out.println(per);
			});
		}//try
		catch(HibernateException he) {
			he.printStackTrace();
		}
		catch(Exception  e) {
			e.printStackTrace();
		}
	}

}
```

```
Hibernate:
    select
        employee0_.PID as pid1_0_,
        employee0_.PNAME as pname3_0_,
        employee0_.COMPANY as company4_0_,
        employee0_.DESG as desg5_0_,
        employee0_.SALARY as salary6_0_,
        employee0_.DEPTNO as deptno7_0_
    from
        HB_INH_PERSON_TPCH employee0_
    where
        employee0_.PERSON_TYPE='emp'
Employee [desg=DEVELOPER, salary=80000.0, deptno=1001]-->;Person [pid=2, pname=rajesh, company=Wipro]
================
Hibernate:
    select
        customer0_.PID as pid1_0_,
        customer0_.PNAME as pname3_0_,
        customer0_.COMPANY as company4_0_,
        customer0_.billNo as billno8_0_,
        customer0_.billAmt as billamt9_0_
    from
        HB_INH_PERSON_TPCH customer0_
    where
        customer0_.PERSON_TYPE='cust'
Customer [billAmt=90000.0, billNo=2345]Person [pid=3, pname=karan, company=SAMSUNG]
================
Hibernate:
    select
        person0_.PID as pid1_0_,
        person0_.PNAME as pname3_0_,
        person0_.COMPANY as company4_0_,
        person0_.DESG as desg5_0_,
        person0_.SALARY as salary6_0_,
        person0_.DEPTNO as deptno7_0_,
        person0_.billNo as billno8_0_,
        person0_.billAmt as billamt9_0_,
        person0_.PERSON_TYPE as person_type2_0_
    from
        HB_INH_PERSON_TPCH person0_
Person [pid=1, pname=raja, company=HCL]
Employee [desg=DEVELOPER, salary=80000.0, deptno=1001]-->;Person [pid=2, pname=rajesh, company=Wipro]
Customer [billAmt=90000.0, billNo=2345]Person [pid=3, pname=karan, company=SAMSUNG]
```

Since the super class ref variable can ref either its own class or dependent class objs .. So select operation performed by using super class name give super related records and the associated child classes related records.

Table per sub class (TPSC) of Inheritence Mapping
==========

Here each sub class record will be inserted into super class table and sub class table who are one to one association using FK column i.e Sub class obj's common data goes to parent db table and specific data goes to child db table by having one to one relatioship b/w those two db tables.

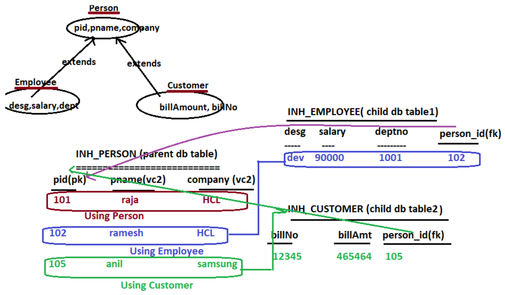
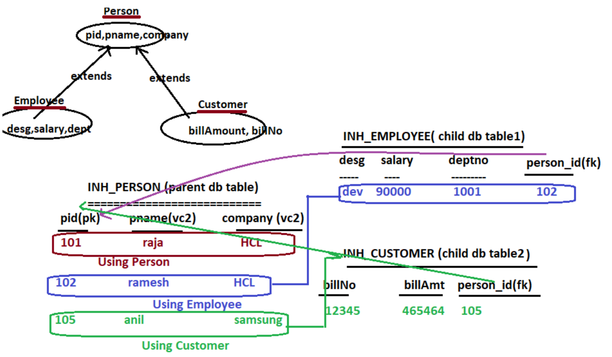


Table per sub class (TPSC) of Inheritence Mapping
==========

Here each sub class record will be inserted into super class table and sub class table who are one to one association using FK column i.e Sub class obj's common data goes to parent db table and specific data goes to child db table by having one to one relatioship b/w those two db tables.



=>The save operation done by the sub class objs will insert common property values in super class db table (parent table) and inserts the specific property values in the db tables of sub classes (child db tables) having one to one relationship.

=>Here taking seperate discriminator column is optional .. beoz there are seperate db tables for specfic properties of sub classes.

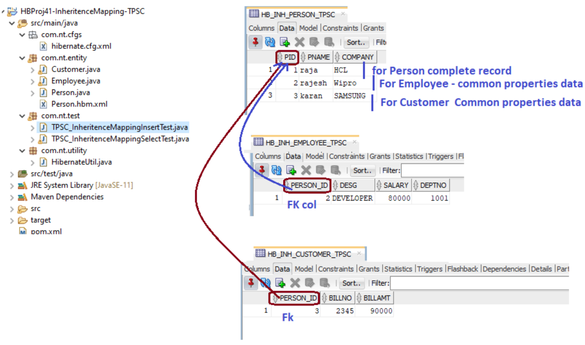
in xml driven cfgs for table per sub class model

=>use <class> tag to cfg super class

=>use <joined-subclass> tags for configuring sub classes

(<joined-subclass> tags are sub tags of <class> tags)

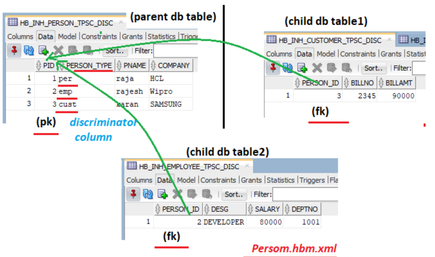
=>use <key> tag under the <joined-subclass> tags to configure FK column..

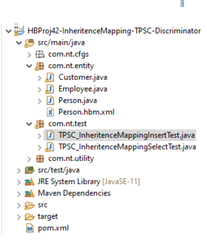


Person.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Person" table="HB_INH_PERSON_TPSC">
    <id name="pid" column="PID">          becomes parent db table
      <generator class="increment"/>
    </id>
    <property name="pname" column="PNAME" length="15" />
    <property name="company" column="COMPANY" length="15"/>
    <joined-subclass name="com.nt.entity.Employee" table="HB_INH_EMPLOYEE_TPSC">
      <key column="PERSON_ID"/> | FK column in child table1      becomes child db table1
      <property name="desg" column="DESG" length="10" />
      <property name="salary" column="SALARY"/>
      <property name="deptno" column="DEPTNO"/>
    </joined-subclass>
    <joined-subclass name="com.nt.entity.Customer" table="HB_INH_CUSTOMER_TPSC">
      <key column="PERSON_ID"/> | FK column in child table2  becomes  child table2
      <property name="billNo" column="BILLNO" />
      <property name="billAmt" column="BILLAMT"/>
    </joined-subclass>
  </class>
</hibernate-mapping>
```

=>In TPSC configurations the descriminator column and discriminator value is optional but discriminator column and vlaue is very much required in parent table becoz the sub classes related common properties data will be stored in parent db table.. For that we need to mix up TPSC and TPCH by taking <joined> and <subclass> tags seperately.. as shown below





Persom.hbm.xml

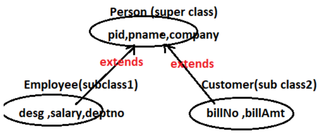
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Person" table="HB_INH_PERSON_TPSC_DISC" discriminator-value="per">
    <id name="pid" column="PID">
      <generator class="increment"/>
    </id>
    <discriminator column="PERSON_TYPE" type="string" length="10"/>
    <property name="pname" column="PNAME" length="15" />
    <property name="company" column="COMPANY" length="15"/>


      <subclass name="com.nt.entity.Employee" discriminator-value="emp" >
        <join table="HB_INH_EMPLOYEE_TPSC_DISC">
          <key column="PERSON_ID"/>
            <property name="desg" column="DESG" length="10" />
            <property name="salary" column="SALARY"/>
            <property name="deptno" column="DEPTNO"/>
          </join>
      </subclass>

      <subclass name="com.nt.entity.Customer" discriminator-value="cust">
        <join table="HB_INH_CUSTOMER_TPSC_DISC">
          <key column="PERSON_ID"/>
            <property name="billNo" column="BILLNO" />
            <property name="billAmt" column="BILLAMT"/>
          </join>
      </subclass>
    </class>
</hibernate-mapping>
```

advantages of table per sub class model Inheritence mapping
---

a) No lengthy db tables are used .. Infact the db tables are in relationship supporting normalization
b) Taking discriminator column is purely optiona
c) It is industry standard approach for inheritence mapping

==========
Table per concrete class (Inheritence mapping strategy no:3)
==========

=> Here the super class record will be saved in super class mapped db table and sub classes record will be saved in the sub classes mapped db tables.. But the db tables of sub classes will maintain colums representing super class common properites and specific properties of sub classses.



HB_INH_PERSON_TPCC (db table1)

pid(pk)	pname	company
101	raja	HCL

Using Person

HB_INH_EMPLOYEE_TPCC (db table2)

pid(pk)	pname	company	desg	salary	deptno
103	rajesh	HCL	CLERK	90000	1001

Using Employee

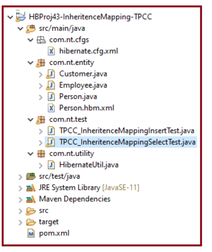
HB_INH_CUSTOMER_TPCC(db table3)

pid(pk)	pname	company	billNo	billAmt
102	suresh	HCL	9999	10000

Using Customer

=>To cfg super class use <class> tag
=>To cfg sub classes use <union-subclass> tags

note:: Since every class of the inheritence hierarchy is having its own db table.. So there is no need of configuring seperate dicsriminator column

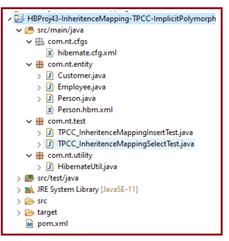


Person.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Person" table="HB_INH_PERSON_TPCC">
    <id name="pid" column="PID">
      <generator class="increment"/>
    </id>
    <property name="pname" column="PNAME" length="15" />
    <property name="company" column="COMPANY" length="15"/>
    <union-subclass name="com.nt.entity.Employee" table="HB_INH_EMPLOYEE_TPCC">
      <property name="desg" column="DESG" length="10" />
      <property name="salary" column="SALARY"/>
      <property name="deptno" column="DEPTNO"/>
    </union-subclass>
    <union-subclass name="com.nt.entity.Customer" table="HB_INH_CUSTOMER_TPCC">
      <property name="billNo" column="BILLNO" />
      <property name="billAmt" column="BILLAMT"/>
    </union-subclass>
  </class>
</hibernate-mapping>
```

==========
Table per concreate class with Implicit polymorphism (strategy no 4 of Inheritence mapping)
==========

=> It is same as Table per concrete class .. but the classes will be configured in the mapping file using indidual <class> tags i.e Entity classes of inheritence are configured in the mapping file with out hightlight their inheritence .. still their inheritence will be recognized becoz of the class level physical inheritence.



Person.hbm.xml (HB mapping file)
==========

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="com.nt.entity.Person" table="HB_INH_PERSON_TPCC1">
      <id name="pid" column="PID">
        <generator class="increment"/>
      </id>
      <property name="pname" column="PNAME" length="15" />
      <property name="company" column="COMPANY" length="15"/>
      </class>
      <class name="com.nt.entity.Employee" table="HB_INH_EMPLOYEE_TPCC1">
       <id name="pid" column="PID">
        <generator class="increment"/>
       </id>
      <property name="pname" column="PNAME" length="15" />
      <property name="company" column="COMPANY" length="15"/>
        <property name="desg" column="DESG" length="10" />
        <property name="salary" column="SALARY"/>
        <property name="deptno" column="DEPTNO"/>
      </class>

    <class name="com.nt.entity.Customer" table="HB_INH_CUSTOMER_TPCC1">
      <id name="pid" column="PID">
        <generator class="increment"/>
      </id>
      <property name="pname" column="PNAME" length="15" />
      <property name="company" column="COMPANY" length="15"/>
        <property name="billNo" column="BILLNO" />
        <property name="billAmt" column="BILLAMT"/>
      </class>

</hibernate-mapping>
```

(db table3)

HB_INH_EMPLOYEE_TPCC1

	PID	PNAME	COMPANY	DESG	SALARY	DEPTNO
1	1	rajesh	Wipro	DEVELOPER	80000	1001

(db table1)

HB_INH_PERSON_TPCC1

	PID	PNAME	COMPANY
1	1	raja	HCL

(db table2)

HB_INH_CUSTOMER_TPCC1

	PID	PNAME	COMPANY	BILLNO	BILLAMT
1	1	karan	SAMSUNG	2345	90000

Limitations of Table per concrete class and Table per concrete class with Implicit polymorphism
==========

(a) The db tables of sub classes having duplicate colums that are there in the table of super class
(b) in Immplicit polymorphism we are configured the inerited common propertie specific properties
(c) It is not industry standard
(d) The db tables of sub classes may become lengthy db tables with huge no.of

what is implicit polymorphism and what is explicit polymorphism?

Implicit polymorphism

=>when the Entity classes are in the inheritence .. The HQL Query that is executed using super class is giving super class records and associated child class records then we can say implicit polymorphism is enabled

=> In all strategies implicit polymorphism is enabled by default and this can not diabled.

=> In TPCC if we configure entity classes of inheritence using multiple <class> tags then we can enable or disable implicit polymorphism

=> IN TPCC example (where multiple <class> tags are used)
==========

```
<class name="com.nt.entity.Person"
        table="HB_INH_PERSON_TPCC" polymorphism="explicit">
 ...
</class>
```

explicit polymorphism

"from Person" gives only Person class related records not the sub classes records

=> IN TPCC example (where multiple <class> tags are used)
==========

```
<class name="com.nt.entity.Person"
        table="HB_INH_PERSON_TPCC" polymorphism="implicit">
 ...
</class>
```

(default value)

implicit polymorphism

"from Person" gives only Person class related records and the sub classes related records

Annotation driven Inheritence mapping

=> This supports only 3 strategies of Inheritence mapping

a) Tabler per class hierarchy (TPCH)

(@Inheritance(strategy = InheritanceType.SINGLE_TABLE)

b) Tabler per sub class (TPSC)

@Inheritance(strategy = InheritanceType.JOINED)

c) Table per concrete class (TCC)

@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)

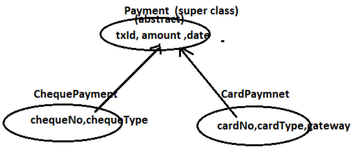
note: @Inheritence annotation must be placed on the top of root in the inheritence hierarchy..
if no inheritence strategy is specifed , the default strategy is "table per class hierarchy" (SINGLE_TABLE)

Tabler class Hierarchy (TPCH)

=> All the classes of inheritence hierarchy will be mapped with single db table that is having discrmminator column

The Annotations are

@Inheritence :: To specify inheritence mapping strategy
@DiscriminatorColumn : To specify Discrimiator column
@DiscriminatorValue : To specify Discrimiator value



HB_ANNO_INH_PAYMENT_TPCH

txId	amount	date	chequeNo	chequeType	cardNo	cardType	gateway	payment_type
1	9000	10-OCT-22	...	...	23456	credit	visa	card
2	5000	11-OCT-22	1111	self	.....	....	....	cheque

Discriminator column

example App

Payment.java

```
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="ANNO_INH_PAYMENT_TPCH")
@Inheritance(strategy =InheritanceType.SINGLE_TABLE) //TPCH
@DiscriminatorColumn(name = "PAYMENT_TYPE",length = 15,discriminatorType =
DiscriminatorType.STRING)
//@DiscriminatorValue("payment") //optional

public abstract class Payment {
    @Id
    @GeneratedValue
    private Long txId;
    private Double amount;
    private LocalDateTime txDate;

}
```

CardPaymebnt.ajva

```
package com.nt.entity;

import java.time.LocalDateTime;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

import lombok.Data;

@Data
@Entity
@DiscriminatorValue("card")
public class CardPayment extends Payment {
    private Long cardNo;
    @Column(length = 15)
    private String cardType;
    @Column(length = 15)
    private String gateway;
    @Override
    public String toString() {
        return "CardPayment [cardNo=" + cardNo +
        ", cardType=" + cardType + ", gateway=" + gateway +super.toString()+ "]";
    }

}
```

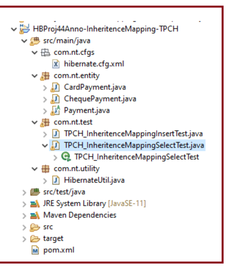
ANNO_INH_PAYMENT_TPCH

	PAYMENT_TYPE	TXID	AMOUNT	TXDATE	CHEQUENO	CHEQUETYPE	CARDNO	CARDTYPE	GATEWAY
1	cheque	73	45677	10-MAR-22 04.30.12.000000000 AM	11223344	Self	(null)	(null)	(null)
2	card	74	77655	10-JAN-22 04.31.12.000000000 AM	(null)	(null)	436778	CREDIT	VISA

Discriminator Column

ChequepPayment.java

```
package com.nt.entity;

import java.time.LocalDateTime;

import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

import lombok.Data;

@Data
@Entity
@DiscriminatorValue("cheque")
public class ChequePayment extends Payment {
    private Long chequeNo;
    @Column(length = 15)
    private String chequeType;
    @Override
    public String toString() {
        return "ChequePayment [chequeNo=" + chequeNo + ", chequeType=" + chequeType + super.toString()+ "]";
    }
}
```



client App1

```
public class TPCH_InheritenceMappingInsertTest {

    public static void main(String[] args) {
      // Get SessionFactory,Session objs
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(factory;ses){
            //prepare objects
              ChequePayment cheque=new ChequePayment();
              cheque.setAmount(45677.0);
              cheque.setTxDate(LocalDateTime.of(2022,03,10,4,30,12));
              cheque.setChequeNo(11223344L);
              cheque.setChequeType("Self");

              CardPayment card=new CardPayment();
              card.setAmount(77655.0);
              card.setTxDate(LocalDateTime.of(2022,01,10,4,31,12));
              card.setCardNo(436778L);
              card.setCardType("CREDIT");
              card.setGateway("VISA");
            //begin Tx
            tx=ses.beginTransaction();
              //save objs
              ses.save(cheque);
              ses.save(card);
    tx.commit();
    System.out.println("Objects are saved");
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getStatus()!=null && tx.getStatus().canRollback()) {
                tx.rollback();
                System.out.println("Problem is saving objects.. ");
            }
        }
        catch(Exception  e) {
            e.printStackTrace();
        }
    }//main
}//classs
```

CardaPayment.java

```
@Data
@Entity
@DiscriminatorValue("card")
public class CardPayment extends Payment {
    private Long cardNo;
    @Column(length = 15)
    private String cardType;
    @Column(length = 15)
    private String gateway;
    @Override
    public String toString() {
        return "CardPayment [cardNo=" + cardNo + ", cardType=" + cardType
+ ", gateway=" + gateway +super.toString()+ "]";
    }
```

TPCH _InheritenceMappingSelectTest .java

```
public class TPCH_InheritenceMappingSelectTest {

    public static void main(String[] args) {
      // Get SessionFactory,Session objs
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();
        try(factory;ses){
// execute HQL Query1
            Query query1= ses.createQuery("from CardPayment");
            List<CardPayment> list1=query1.getResultList();
            list1.forEach(card->{
                System.out.println(card);
            });
            System.out.println("================");
            Query query2= ses.createQuery("from ChequePayment");
            List<ChequePayment> list2=query2.getResultList();
            list2.forEach(ch->{
                System.out.println(ch);
            });
            System.out.println("================");
            Query query3= ses.createQuery("from Payment");
            List<Payment> list3=query3.getResultList();
            list3.forEach(pay->{
                System.out.println(pay);
            });
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
        }
        catch(Exception  e) {
            e.printStackTrace();
        }
    }

}
```

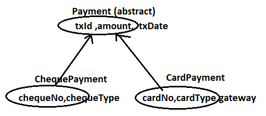
In hibernate.cfg.xml

```
<!-- mapping annoations -->
    <mapping class="com.nt.entity.Payment"/>
    <mapping class="com.nt.entity.ChequePayment"/>
    <mapping class="com.nt.entity.CardPayment"/>
```

Annotation driven Inheritence mapping (Table per sub class)

==========

=>if the objects are saved using sub classes then the common property values will be shared in in parent db table records and specic property values will be shared with child table records maintaining one to one relationship amon the records.



HB_INH_ANNO_PAYMENT_TPSC (parent db table)

txId(pk)	amount	txDate
1	10000	20-MAR-22
2	40000	10-MAR-22

HB_INH_ANNO_CHEQUEPAYMENT_TPSC (child table1)

chequeNo (pk)	chequeType	payment_id (fk)
12344	self	1

HB_INH_ANNO_CARDPAYMENT_TPSC (child table2)

cardNo (pk)	cardType	gateway	payment_id(fk)
23456	debit	visa	2

Required annotations are

(a) @Inheritence :: To inheritence mappin strategy as strtegy=InheritenceType.JOINED

(b) @PrimarykeyJoinColumn :: To specify FK column of the child table pointing to parent table pk column

The entity classes are

Payment.java

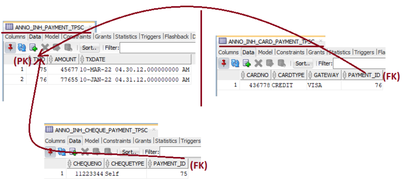
```
package com.nt.entity;

import java.time.LocalDateTime;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="ANNO_INH_PAYMENT_TPSC")
@Inheritance(strategy =InheritanceType.JOINED)  //TPSC
public abstract class Payment {
        @Id
        @GeneratedValue
        private Long txId;
        private  Double amount;
        private  LocalDateTime txDate;

}
```

CardPayment.java

```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="ANNO_INH_CARD_PAYMENT_TPSC")
@PrimaryKeyJoinColumn(name = "PAYMENT_ID",referencedColumnName = "TXID")
public class CardPayment extends Payment {
        private  Long cardNo;
        @Column(length = 15)
        private   String cardType;
        @Column(length = 15)
        private  String gateway;
        @Override
        public String toString() {
                return "CardPayment [cardNo=" + cardNo +

                ", cardType=" + cardType + ", gateway=" + gateway +super.toString()+ "]";
        }

}
```

ChequePayment.java

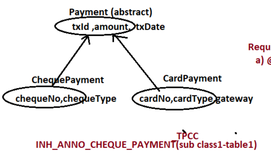
```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="ANNO_INH_CHEQUE_PAYMENT_TPSC")
@PrimaryKeyJoinColumn(name="PAYMENT_ID",referencedColumnName = "TXID")
public class ChequePayment extends Payment {
        private  Long chequeNo;
        @Column(length = 15)
        private   String chequeType;
        @Override
        public String toString() {
                return "ChequePayment [chequeNo=" + chequeNo + ", chequeType=" + chequeType + super.toString()+ "]";
        }
}
```



Annotation driven
Table per concrete class (TPCC)

==========

=> Each sub classs of ineritence hiearchu will have seperate db table .. In that it will main columns reprseneting common properties and specific propeties



Required Annotations are

a) @Inheritence -- To specify strategy as InheriteceType.TABLE_PER_CLASS

TPCC

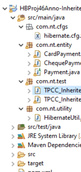
INH_ANNO_CHEQUE_PAYMENT(sub class1-table1)

txId(pk)	amount	txDate	chequeNo	chequeType
1	99999	10-MAR-22	12345	self

INH_ANNO_CARD_PAYMENT_TPCC (sub class2 -table2)

txId(pk)	amount	txDAte	cardNo	cardType	gateway
2	88888	10-APR-22	4575	credit	VISA

The column repsenting comm properties are repeated ..So it is bad strategy to implement



super class

```
@Data
@Entity
@Table(name="ANNO_INH_PAYMENT_TPCC")  //remain empty db table
@Inheritance(strategy =InheritanceType.TABLE_PER_CLASS)  //TPCC
public abstract class Payment {
        @Id
        @GeneratedValue
        private Long txId;
        private  Double amount;
        private  LocalDateTime txDate;

}
```

sub class1

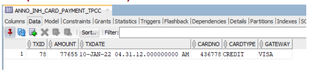
```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="ANNO_INH_CHEQUE_PAYMENT_TPCC")
public class ChequePayment extends Payment {
        private  Long chequeNo;
        @Column(length = 15)
        private   String chequeType;
        @Override
        public String toString() {
                return "ChequePayment [chequeNo=" + chequeNo + ", chequeType=" + chequeType + super.toString()+ "]";
        }
}
```

Sub class2

```
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
import lombok.Data;

@Data
@Entity
@Table(name="ANNO_INH_CARD_PAYMENT_TPCC")
public class CardPayment extends Payment {
        private  Long cardNo;
        @Column(length = 15)
        private   String cardType;
        @Column(length = 15)
        private  String gateway;
        @Override
        public String toString() {
                return "CardPayment [cardNo=" + cardNo +
                ", cardType=" + cardType + ", gateway=" + gateway +super.toString()+ "]";
        }
}
```

(db table for sub class1)

(db table for sub class2)



Collection Mapping

=> we need to go for collection mapping when Entity class having Collection type property.. like List /Set /Map type property.

=> To cfg this collection type propety we need to use following tags

collection type	xml tag
java.util.List	<list> /<bag>
java.util.Set	<set>
java.util.Map	<map>

In collection mapping collection element simple values will come as child db table elements having one to many association with parent db table records with the support of FK colum

```
public class PersonDetails{
  private int pid;
  private String pname;
  private String paddrs;
  private List<String> nickNames;
  private List<String> friends;
  private Set<Long> contactNumbers;
  private Map<String,Long> idDetails;
  //setters and getters
   ....
   ...
 }
```

=>List collection maintains index for elements and allows duplicates

=>Set collection does not maintain index for elements and does not allow duplicates

=>Map collection maintain index for elements (keys) and allows duplicate values but keys can not be duplicated.

if u r cfg List collection property using <list> we must provide column child table to maintain the index value.. but if we cfg same List collection property using <bag> tag the index column cfg is optional

HB_Person_Details_Collection (parent db table)

pid(pk)	pname	paddrs
1	raja	hyd
2	anitha	delhi

person_nicknames_collection (child table1)

nickname	index	person_id(fk)
king	0	1
raj	1	1
ani	0	2
annu	1	2

element values of List collection

person_friends_collection (child table2)

friend	person_id(fk)
ravi	1
ramesh	1
suresh	1
sunitha	2
kiran	2
karan	2

element values of List collection

person_contactnumbers_collection(child table3)

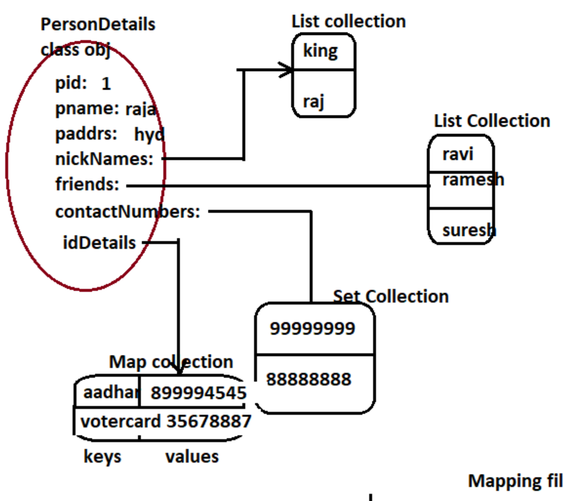
contactnumber	person_id(fk)
99999999	1
88888888	1
77777777	2
66666666	2

element values of Set collection

Peron_idDetails_collection (child table4)

idType	idnumber	person_id(fk)
aadhar	899994545	1
votercard	35678887	1
aadhar	45435665	2
votercard	5467887	2

Map collection keys — Map collection values



example code

Entity class

```
package com.nt.entity;

import java.util.List;
import java.util.Map;
import java.util.Set;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PersonDetails {
        private Integer pid;
        private String pname;
        private String paddrs;
        private List<String> nickNames;
        private List<String> friends;
        private Set<Long> contactNumbers;
        private Map<String,Long> idDetails;


}
```

Mapping file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.PersonDetails" table="HB_PER_DETAILS_COLLECTION">
    <id name="pid" column="PID">
      <generator class="increment"/>
    </id>
    <property name="pname" column="PNAME" length="15" />
    <property name="paddrs" column="PADDRS" length="15"/>
    <list name="nickNames" table="HB_PER_NICKNAMES_COLLECTION">
      <key column="PERSON_ID"/> <!-- FK column -->
      <index column="INDX" type="int" length="3"/> <!-- index column> -->
      <element column="NICKNAME" type="string"/> <!-- List collectin element value column -->
    </list>
    <bag name="friends" table="HB_PER_FRIENDS_COLLECTION">
      <key column="PERSON_ID"/>
      <element column="FRIEND" type="string"/>
    </bag>
    <set name="contactNumbers" table="HB_PER_CONTACT_COLLECTION">
      <key column="PERSON_ID"/>
      <element column="CONTACT_NUMBER" type="long"/>
    </set>
    <map name="idDetails" table="HB_PER_IDETAILS_COLLECTION">
       <key column="PERSON_ID"/>
       <index column="IDTYPE" type="string" length="10"/>
       <element column="IDNUMBER" type="long" length="14"/>
    </map>

  </class>
</hibernate-mapping>
```

Collection Mapping

=> we need to go for collection mapping when Entity class having Collection type property.. like List /Set /Map type property.

=> To cfg this collection type property we need to use following tags

collection type	xml tag
java.util.List	<list> /</bag>
java.util.Set	<set>
java.util.Map	<map>

In collection mapping collection element simple values will come as child db table elements having one to many association with parent db table records with the support of FK colum

=>List collection maintains index for elements and allows duplicates

=>Set collection does not maintain index for elements and does not allow duplicates

=>Map collection maintain index for elements (keys) and allows duplicate values but keys can not be duplicated.

If u r cfg List collection property using <list> we must provide column child table to maintain the index value.. but if we cfg same List collection property using <bag> tag the index column cfg is optional

```
public class PersonDetails{
    private int pid;
    private String pname;
    private String paddrs;
    private List<String> nickNames;
    private List<String> friends;
    private Set<Long> contactNumbers;
    private Map<String,Long> IdDetails;
    //setters and getters
     ....
     ...
}
```

HB_Person_Details_Collection (parent db table)

pid(pk)	pname	paddrs
1	raja	hyd
2	anitha	delhi

person_nicknames_collection (child table1)

nickname	index	person_id(fk)
king	0	1
raj	1	1
ani	0	2
annu	1	2

element values of List collection

person_friends_collection (child table2)

friend	person_id(fk)
ravi	1
ramesh	1
suresh	1
sunitha	2
kiran	2
karan	2

element values of List collection

person_contactnumbers_collection(child table3)

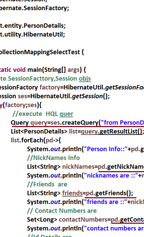
contactnumber	person_id(fk)
99999999	1
88888888	1
77777777	2
66666666	2

element values of Set collection

Person_idDetails_collection (child table4)

idType	Idnumber	person_id(fk)
aadhar	899994545	1
votercard	35678887	1
aadhar	45435665	2
votercard	5467887	2

Map collection keys — Map collection values



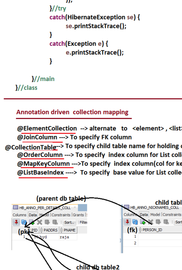
example code
==========

Entity class

```
package com.nt.entity;

import java.util.List;
import java.util.Map;
import java.util.Set;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
public class PersonDetails {
    private Integer pid;
    private String pname;
    private String paddrs;
    private List<String> nickNames;
    private List<String> friends;
    private Set<Long> contactNumbers;
    private Map<String,Long> IdDetails;

}
```

Mapping file
==========

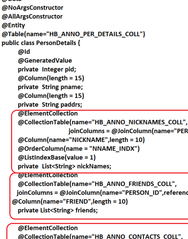
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class  name="com.nt.entity.PersonDetails"  table="HB_PER_DETAILS_COLLECTION">
        <id name="pid"  column="PID">
            <generator class="increment"/>
        </id>
        <property name="pname"  column="PNAME" length="15" />
        <property name="paddrs" column="PADDRS"  length="15"/>
        <list name="nickNames"  table="HB_PER_NICKNAMES_COLLECTION">
            <key column="PERSON_ID"/>  <!-- FK column -->
            <index column="INDX" type="int" length="3"/> <!-- index column> -->
            <element column="NICKNAME"  type="string"/> <!-- List collection element value column -->
        </list>
        <bag  name="friends"  table="HB_PER_FRIENDS_COLLECTION">
            <key column="PERSON_ID"/>
            <element column="FRIEND"  type="string"/>
        </bag>
        <set name="contactNumbers"  table="HB_PER_CONTACT_COLLECTION">
            <key column="PERSON_ID"/>
            <element column="CONTACT_NUMBER"  type="long"/>
        </set>
        <map name="IdDetails"  table="HB_PER_IDETAILS_COLLECTION">
            <key column="PERSON_ID"/>
            <index column="IDTYPE" type="string" length="20"/>
            <element column="IDNUMBER" type="long" length="24"/>
        </map>

    </class>
</hibernate-mapping>
```



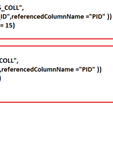

```
//Select operation in Collection Mapping
==========================================

package com.nt.test;

import java.util.List;
import java.util.Map;
import java.util.Set;

import java.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import com.nt.entity.PersonDetails;
import com.nt.utility.HibernateUtil;

public class CollectionMappingSelectTest {

    public static void main(String[] args) {
    // create SessionFactory,Session obds
        SessionFactory factory=HibernateUtil.getSessionFactory();
        Session ses=HibernateUtil.getSession();
        try{factory=null;
            //execute HQL quer
            Query query=ses.createQuery("from PersonDetails");
            List<PersonDetails> listquery=query.getResultList();
            list.forEach(pd->{
                System.out.println("Person Info:"+pd.getPid()+"  "+pd.getPname()+"  "+pd.getPaddrs());
                //NickNames Info
                List<String> nickNames=pd.getNickNames();
                System.out.println("nicknames are :: "+nickNames.toString());
                //Friends are
                List<String> friends=pd.getFriends();
                System.out.println("Friends are :: "+nickNames.toString());
                // Contact Numbers are
                Set<Long> contactNumbers=pd.getContactNumbers();
                System.out.println("contact numbers are :: "+contactNumbers);
                //Id Details are
                Map<String,Long> idDetails=pd.getIdDetails();
                System.out.println("idDetails  are:: "+idDetails);
            });
        }//try
        catch(HibernateException se) {
            se.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }

    }//main
}//class
```

Annotation driven collection mapping

@ElementCollection --> alternate to <element> , <list>/<set>/<map>/<bag> tags
@JoinColumn --> To specify FK column
@CollectionTable --> To specify child table name for holding collection elements
@OrderColumn --> To specify index column for list collection elements
@MapKeyColumn --> To specify index column(col for keys) for Map collection elements
@ListBaseIndex --> To specify base value for List collection elements



Entity class
==========



```
package com.nt.entity;

import java.util.List;
import java.util.Map;
import java.util.Set;

import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.MapKeyColumn;
import javax.persistence.OrderColumn;
import javax.persistence.Table;

import org.hibernate.annotations.ListIndexBase;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name="HB_ANNO_PER_DETAILS_COLL")
public class PersonDetails {
    @Id
    @GeneratedValue
    private Integer pid;
    @Column(length = 15)
    private String pname;
    @Column(length = 15)
    private String paddrs;
    @ElementCollection
    @CollectionTable(name="HB_ANNO_NICKNAMES_COLL",
                     joinColumns = @JoinColumn(name="PERSON_ID",referencedColumnName ="PID" ))
    @Column(name="NICKNAME",length = 10)
    @OrderColumn(name = "NNAME_INDX")
    @ListIndexBase(value = 1)
    private List<String> nickNames;
    @ElementCollection
    @CollectionTable(name="HB_ANNO_FRIENDS_COLL",
    joinColumns = @JoinColumn(name="PERSON_ID",referencedColumnName ="PID" ))
    @Column(name="FRIEND",length = 10)
    private List<String> friends;

    @ElementCollection
    @CollectionTable(name="HB_ANNO_CONTACTS_COLL",
      joinColumns = @JoinColumn(name="PERSON_ID",referencedColumnName ="PID" ))
      @Column(name="CONTACT_NUMBER",length = 15)
    private Set<Long> contactNumbers;

    @ElementCollection
    @CollectionTable(name="HB_ANNO_IDETAILS_COLL",
   joinColumns = @JoinColumn(name="PERSON_ID",referencedColumnName ="PID" ))
      @MapKeyColumn(name = "IDTYPE",length = 10)
      @Column(name="ID_NUMBER",length = 15)
    private  Map<String,Long> IdDetails;

}
```

Need of Association Mapping /Relationships o -r mapping

Problem:: Storing multiple entities data in single db table is not recomanded becoz it gives data redundency problem and data management problem.

User_Phone_Details (db table)

userId(n)	uname (vc2)	addrs(vc2)	mobileNo(n)	numberType(vc2)	provider(vc2)
101	raja	hyd	99999999	personal	airtel
101	raja	hyd	88888888	office	jio
102	sharat	vizag	77777777	personal	airtel
102	mahesh	vizag	66666666	office	VI

It has got two limitations

a) Data Redundecy Problem (Data Repeatation)

b) Data management problem (taking huge no.of cols in single db table is not recomanded)

Solution1 :: Take two entities data in two different db tables (Bad solution)

User_Details

userId(n) (pk)	uname (vc2)	addrs(vc2)
101	raja	hyd
102	mahesh	vizag

Phone_numbers

regNo(n) (pk)	mobileNo(n)	numberType(vc2)	provider(vc2)
1	99999999	personal	airtel
2	88888888	office	jio
3	77777777	personal	airtel
4	66666666	office	VI

pros

=>Data redudency Problem is solved

=>Data management problem is solved (less no.of cols in each db table)

cons

=> Data Navigation problem is created i.e we can not access user details from phone number details and vice-versa (reverse also true) becoz the db tables are not in relationship

Solution2 :: Keep two entities data in two different db tabels and keep those db tables in relationship by taking FK column parent db table (Bad solution)

User_Details (parent db table)

userId(n)	uname (vc2)	addrs(vc2)
101	raja	hyd
101	raja	hyd
102	mahesh	vizag
102	mahesh	vizag

Data redundency problem

Phone_numbers (child table)

phone_regNo(FK)	regNo(n) (pk)	mobileNo(n)	numberType(vc2)	provider(vc2)
1	1	99999999	personal	airtel
2	2	88888888	office	jio
3	3	77777777	personal	airtel
4	4	66666666	office	VI

pros

=> Data navigation problem is solved becoz FK i.e we can access phone number details from user info and vice-versa

=> Data mangement problem is not there..

cons

=> Data redundency problem is still there.

Solution3 :: Keep entities data in two different db tables having FK column in child db table (Best Solution)

User_Details (Parent db table)

userId(n) (pk)	uname (vc2)	addrs(vc2)
101	raja	hyd
102	mahesh	vizag

Phone_numbers (Child db table)

regNo(n) (pk)	mobileNo(n)	numberType(vc2)	provider(vc2)	user_id(fk)
1	99999999	personal	airtel	101
2	88888888	office	jio	101
3	77777777	personal	airtel	102
4	66666666	office	VI	102

pros

a) Data redundency problem is solved

b) Data management is good

c) Data Nagivation is possible as needed.

=>if db tables are in relationship the associated entity classes should also be in relationship but db tables will be in relationship using FK column and such concept is not there in java...we keep java entity classes in relationship using the support of HAS-A properties (composition) .. So to map the db tables of relationship and the entity classes of relationship we need to take the support of advanced o-r mapping called Association mapping.

Association Mapiing between Entity classes are possible in the following ways

a) One to One

b) One To many

c) Many To One

d) Many To Many

The Association b/w entity classes can be implemented in two ways

(a) As Uni-Directional Association

[we can access either child records from parent(c to p) or parent records from child (p to c)]

(b) As Bi-Directional Association

[we can access child records/objects from parent records/objects(c to p) and parent records/objects from child records/objects (p to c)]

note:: one to one , one to many , many to one can be implemented in both parent to child and child to parent mode association but the many to many association should always be implemented as bi-directional association.,

=>Developing Entity classes for association mapping can done using two types of special properties

a) Collection Type special Property

[One to many and Many To Many]

b) Non Collection Type Special Property

[One To One and Many to One]

use-cases on differnt assoications

One To One Association

a) Person and Passport

b) Person and VoterId

c) Person and DrivingLicense

d) Person and BloodGroupDetails

e) collegeStudent and CollectLibraryMembership

and etc..

One to many Association

a) User and Mobile numbers

b) ParentDetails and KidDetaisl

c) Person and BankAccounts

d) Country and States

and etc..

Many To One Association

a) Employees and Depeartment

b) Students and College (many students studies in a college)

c) Books and Library

d) States and conuntry

and etc..

Many to Many Assocation

=>Each parent contains multiple childs and each child contains multiple parents

=> It is combination of 1 to many from parent side and 1 to many from child side

=> each parent contains multiple childs but each child again contains multiple parents

a) Doctors and patients (each docots gets mutliple patients.. but each patient can take treatement from multiple doctors)

b) Students and Teachers (each student takes traning from mutliple Teachers.. but each teacher again gives training to multiple students)

c) Vendors and Customers

and etc..

One to Many Uni-Directional Association (parent to child)
==========

=> This association says each parent will have multiple associated records
=> To build this association , we need to take Collection type special property having capability to hold bunch of child class objs.
=> To build this association in xml driven cfgs we need to use <one-to-many> tag with <list>/<set>/<bag>/<map> tags

usecase :: Person ------ PhoneNumber :: 1 Person can have more than 1 phone number

Person_Details (Parent db table)
==========

pid (n) (pk)	pname (vc2)	addrs(vc2)
101	raja	hyd
102	mahesh	vizag

Phone_numbers (Child db table)
==========

regNo(n) (pk)	mobileNo(n)	numberType(vc2)	provider(vc2)	person_id(fk)
1	99999999	personal	airtel	101
2	88888888	office	jio	101
3	77777777	personal	airtel	102
4	66666666	office	VI	102

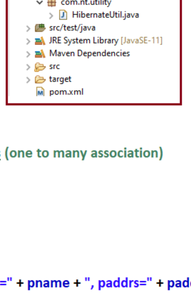
Entity classes Designing
==========

Parent class

```
public class PersonDetails implements Serializable
  private Integer pid;
  private String pname;
  private String paddrs;
  private Set<PhoneNumber> phones;

  //setters && getters
   ...
   ...
   ..
}
```

child class

```
public class PhoneNumber implements Serializable{
  private Integer regno;
  private Long mobileNumber;
  private String numberType;
  private String provider;

  //setters and getters

}
```



PersonDetails.java

```
// PersonDetails.java (parent class)
package com.nt.entity;

import java.io.Serializable;
import java.util.Set;

import lombok.Data;

@Data
public class PersonDetails implements Serializable {
        private Integer pid;
        private String pname;
        private String paddrs;
        //special property to hold bunch of child class objs (one to many association)
        private Set<PhoneNumber> phones;


        //toString
        @Override
        public String toString() {
                return "PersonDetails [pid=" + pid + ", pname=" + pname + ", paddrs=" + paddrs + "]";
        }
}
```

PhoneNumber.java

```
//PhoneNumber.java (child class)
package com.nt.entity;

import java.io.Serializable;

import lombok.Data;

@Data
public class PhoneNumber implements Serializable{
   private Integer regNo;
   private Long mobileNo;
   private String numberType;
   private String provider;

    //taking property for FK column is pure optional
   @Override
   public String toString() {
      return "PhoneNumber [regNo=" + regNo + ",
                mobileNo=" + mobileNo + ", numberType=" +
                        numberType + ", provider=" + provider + "]";

   }
}
```

PersonDetails.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="com.nt.entity.PersonDetails" table="HB_PERSON_DETAILS_OTM">
      <id name="pid" column="PID">
         <generator class="increment"/>
      </id>
      <property name="pname" column="PNAME" length="20"/>
      <property name="paddrs" column="PADDRS" length="20"/>
      <set name="phones" cascade="all" >
        <key column="PERSON_ID"/> <!-- FK column -->
        <one-to-many class="com.nt.entity.PhoneNumber"/> <!-- child class cfg -->
      </set>
   </class>
</hibernate-mapping>
```

PhoneNumber.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class name="com.nt.entity.PhoneNumber" table="HB_PHONE_NUMBER_OTM">
      <id name="regNo" column="REGNO">
         <generator class="sequence">
           <param name="sequence_name">PID_SEQ1</param>
           <param name="initial_value">1000</param>
           <param name="increment_size">1</param>
         </generator>
      </id>
      <property name="mobileNo" column="MOBILENO"/>
      <property name="numberType" column="NUMBERTYPE" length="15"/>
      <property name="provider" column="PROVIDER" length="15"/>
   </class>
</hibernate-mapping>
```

DAO Impl class

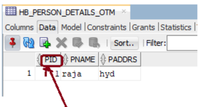
```
package com.nt.dao;

import java.util.Set;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.PersonDetails;
import com.nt.entity.PhoneNumber;
import com.nt.utility.HibernateUtil;

public class PersonDetailsDAOImpl implements IPersonDetailsDAO {

      @Override
      public void saveDataUsingParent() {
            //get Accesss to Session object
            Session ses=HibernateUtil.getSession();
            Transaction tx=null;
            try(ses){
                  tx=ses.beginTransaction();
                    //prepare parent obj
                    PersonDetails pDetails=new PersonDetails();
                    pDetails.setPname("raja"); pDetails.setPaddrs("hyd");
                    //prepare child objs
                    PhoneNumber ph1=new PhoneNumber();
                    ph1.setNumberType("residence"); ph1.setProvider("airtel");
                    ph1.setMobileNo(999999999L);

                    PhoneNumber ph2=new PhoneNumber();
                    ph2.setNumberType("office"); ph2.setProvider("jio");
                    ph2.setMobileNo(888888888L);
                    //set child objs to parent obj
                    pDetails.setPhones(Set.of(ph1,ph2));
                   //save parent obj
                    int idVal=(Integer)ses.save(pDetails);
                    System.out.println("Generated id value::"+idVal);
                  tx.commit();
                  System.out.println("Parent and the associated childs are saved");
            }//try
            catch(HibernateException he) {
                  he.printStackTrace();
                  if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                        tx.rollback();
                        System.out.println("Parent and the associated childs are not saved");
                  }//if
            }//catch
            catch(Exception e) {
                  e.printStackTrace();
            }

      }//method
}//class
```

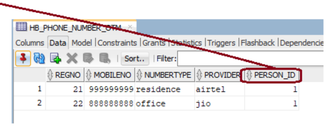
DAO Interface

```
package com.nt.dao;

public interface IPersonDetailsDAO {
    public void saveDataUsingParent();
}
```

Client Appl

```
package com.nt.test;

import com.nt.dao.IPersonDetailsDAO;
import com.nt.dao.PersonDetailsDAOImpl;
import com.nt.utility.HibernateUtil;

public class OneToManyUniTest {

      public static void main(String[] args) {
            IPersonDetailsDAO dao=new PersonDetailsDAOImpl();
            dao.saveDataUsingParent();

            //close SessionFactory
            HibernateUtil.closeSessionFactory();

      }
```





```
==========================================
Select operation in one to many Association
==========================================
```

=>In One to Many Association the parent objects will be loaded eargly and the associated child objs will be loaded lazily becoz the lazy attribute value "true" by default.

=>we can specify wheather the associated child records should be loaded lazily or eagerly or extra lazily by using "lazy" attribute of collection tag <set>/list>/<map>/<bag> tags.

```
@Override
    public void loadDataUsingParent() {
        //get Accesss to Session object
        Session ses=HibernateUtil.getSession();
        try{ses){
            Query query=ses.createQuery("from PersonDetails");
            List<PersonDetails> list=query.getResultList();
            list.forEach(pd->{
                System.out.println("parent ::"+pd);
                    Set<PhoneNumber> childs=pd.getPhones();
                    childs.forEach(ph->{
                        System.out.println("child::"+ph);
                    });
                System.out.println("............");
            });
        }
        catch(HibernateException he) {
            he.printStackTrace();
        }//catch
        ccatch(Exception e) {
            e.printStackTrace();
        }
    }//method
```

lazy="true" (default) :: loads parent objs normally ..but loads the associated child objs lazily on demand . Comment the following code of selecting child records and observe the generate queries

```
/*      Set<PhoneNumber> childs=pd.getPhones();
        childs.forEach(ph->{
            System.out.println("child::"+ph);
        });  */
```

```
Generated Queries are::
  Hibernate:
    select
      persondeta0_.PID as pid1_0_,
      persondeta0_.PNAME as pname2_0_,
      persondeta0_.PADDRS as paddrs3_0_
  from
    HB_PERSON_DETAILS_OTM persondeta0_
```

gives all parents

```
Hibernate:
  select
    phones0_.PERSON_ID as person_id5_1_0_,
    phones0_.REGNO as regno1_1_0_,
    phones0_.REGNO as regno1_1_1_,
    phones0_.MOBILENO as mobileno2_1_1_,
    phones0_.NUMBERTYPE as numbertype3_1_1_,
    phones0_.PROVIDER as provider4_1_1_
  from
    HB_PHONE_NUMBER_OTM phones0_
  where
    phones0_.PERSON_ID=?
```

gives childs of first parent

```
Hibernate:
  select
    phones0_.PERSON_ID as person_id5_1_0_,
    phones0_.REGNO as regno1_1_0_,
    phones0_.REGNO as regno1_1_1_,
    phones0_.MOBILENO as mobileno2_1_1_,
    phones0_.NUMBERTYPE as numbertype3_1_1_,
    phones0_.PROVIDER as provider4_1_1_
  from
    HB_PHONE_NUMBER_OTM phones0_
  where
    phones0_.PERSON_ID=?
```

Gives childs of second parent

lazy="false" :: performs eager loading i.e child records/objs will be loaded along with the parent records/objs eagerly irrepsective of wheather child objs are used or not.

[Execute the above setup code and observe the generated queries]

```
Hibernate:
  select
    persondeta0_.PID as pid1_0_,
    persondeta0_.PNAME as pname2_0_,
    persondeta0_.PADDRS as paddrs3_0_
  from
    HB_PERSON_DETAILS_OTM persondeta0_
parent ::PersonDetails [pid=1, pname=raja, paddrs=hyd]
Hibernate:
  select
    phones0_.PERSON_ID as person_id5_1_0_,
    phones0_.REGNO as regno1_1_0_,
    phones0_.REGNO as regno1_1_1_,
    phones0_.MOBILENO as mobileno2_1_1_,
    phones0_.NUMBERTYPE as numbertype3_1_1_,
    phones0_.PROVIDER as provider4_1_1_
  from
    HB_PHONE_NUMBER_OTM phones0_
  where
    phones0_.PERSON_ID=?
childs count::2
...........
parent ::PersonDetails [pid=2, pname=raja, paddrs=hyd]
Hibernate:
  select
    persondeta0_.PID as pid1_0_,
    persondeta0_.PNAME as pname2_0_,
    persondeta0_.PADDRS as paddrs3_0_
  from
    HB_PERSON_DETAILS_OTM persondeta0_
parent ::PersonDetails [pid=2, pname=raja, paddrs=hyd]
```

lazy="extra" (best) : Performs extra lazy loading operation i.e loads parent objs normally ..but the loads the associated child objs extra lazy i.e if we try to perform aggregate operations on childs collection like getting childs count , checking childs are empty or not and etc.. operations does not load childs objs infact generate seperate aggragate query to complete those those aggrate operations.

```
Query query=ses.createQuery("from PersonDetails");
List<PersonDetails> list=query.getResultList();
list.forEach(pd->{
  System.out.println("parent ::"+pd);
    Set<PhoneNumber> childs=pd.getPhones();
    System.out.println("childs count::"+childs.size());
    /*childs.forEach(ph->{
     System.out.println("child::"+ph);
    });*/
  System.out.println("............");
```

Generated SQL Queries

```
Hibernate:
  select
    persondeta0_.PID as pid1_0_,
    persondeta0_.PNAME as pname2_0_,
    persondeta0_.PADDRS as paddrs3_0_
  from
    HB_PERSON_DETAILS_OTM persondeta0_
parent ::PersonDetails [pid=1, pname=raja, paddrs=hyd]
```

gives all parent records

```
Hibernate:
  select
    count(REGNO)
  from
    HB_PHONE_NUMBER_OTM
  where
    PERSON_ID =?
childs count::2
...........
parent ::PersonDetails [pid=2, pname=raja, paddrs=hyd]
Hibernate:
  select
    count(REGNO)
  from
    HB_PHONE_NUMBER_OTM
  where
    PERSON_ID =?
childs count::2
```

query for childs collection aggregation

query for childs collection aggregation

note:: if the same code is executed using lazy="true" then it generated select queries to load childs objs just for aggrgate operation performed on the chids collection

```
Hibernate:
  select
    persondeta0_.PID as pid1_0_,
    persondeta0_.PNAME as pname2_0_,
    persondeta0_.PADDRS as paddrs3_0_
  from
    HB_PERSON_DETAILS_OTM persondeta0_
parent ::PersonDetails [pid=1, pname=raja, paddrs=hyd]
Hibernate:
  select
    phones0_.PERSON_ID as person_id5_1_0_,
    phones0_.REGNO as regno1_1_0_,
    phones0_.REGNO as regno1_1_1_,
    phones0_.MOBILENO as mobileno2_1_1_,
    phones0_.NUMBERTYPE as numbertype3_1_1_,
    phones0_.PROVIDER as provider4_1_1_
  from
    HB_PHONE_NUMBER_OTM phones0_
  where
    phones0_.PERSON_ID=?
childs count::2
...........
parent ::PersonDetails [pid=2, pname=raja, paddrs=hyd]
Hibernate:
  select
    phones0_.PERSON_ID as person_id5_1_0_,
    phones0_.REGNO as regno1_1_0_,
    phones0_.REGNO as regno1_1_1_,
    phones0_.MOBILENO as mobileno2_1_1_,
    phones0_.NUMBERTYPE as numbertype3_1_1_,
    phones0_.PROVIDER as provider4_1_1_
  from
    HB_PHONE_NUMBER_OTM phones0_
  where
    phones0_.PERSON_ID=?
childs count::2
...........
```

Loads the parent objs

Loads the childs objs of a prent for cmpleting aggragate operation

Loads the childs objs of a prent for cmpleting aggragate operation

```
In PersonDetails.hbm.xml
-------------------------------------------
                                  false or extra
<set name="phones" cascade="all" lazy="true" >
        <key column="PERSON_ID"/> <!-- FK column -->
        <one-to-many class="com.nt.entity.PhoneNumber"/> <!-- child class cfg -->
    </set>
```

lazy="true" (default) :: lazy loading .. parent objs will be load normally but the child objs will be loaded on demand /lazily .. To perform aggregate operations on childs collection also the childs objs will be loaded which is bad

lazy="extra" (best) :: extra lazy loading .. parent objs will be load normally but the child objs will be loaded on demand /lazily .. To perform aggregate operations on childs collection the childs objs will not be loaded which is good .. just select query with aggregate operation will be generated

lazy="false" :: eager loading .. The child objs will be loaded along with the parent objs eagerly.

note: "cascade" is required towards non-select operations of association mapping.. where "lazy" is required towards select operations of association mapping.

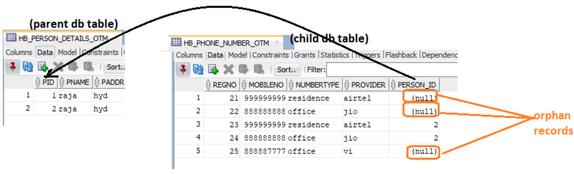
```
How to perform lazy loading in hibernate apps
==============================================
  Ans1)  using   ses.load(-,-) with lazy="true" in single row select operation
  Ans2)  Using   query.iterate(-) while executing HQL Select query
  Ans3)  Using   lazy="true" or lazy="extra" in one to many and many to many association
  Ans4)  Using   lazy="proxy" or lazy="no-proxy" in one to one and many to one association

Adding additional child for existing parent object
==================================================
@Override
    public void addChildToExistingParent() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try{ses){
            tx=ses.beginTransaction();
            //Load existing parent
        PersonDetails pd=ses.get(PersonDetails.class, 1);
        //get all childs of parent
        Set<PhoneNumber> childs=pd.getPhones();
        //create new Child obj
        PhoneNumber ph=new PhoneNumber();
        ph.setMobileNo(8888877771);
        ph.setNumberType("office");
        ph.setProvider("vi");
        //add new child to existing parent
        childs.add(ph);
            tx.commit();
            System.out.println("Child is added to existing childs of a parent");
        }
        catch(HibernateException he) {
            he.printStackTrace();
            if(txl!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                tx.rollback();
            }
        }//catch
        catch(Exception e) {
            e.printStackTrace();
        }
    }//method
```

delete operation in One to many Assocation
==========

Deleting all childs childs of a parent

```
@Override
    public void deleteAllChildsOfAParent() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(ses){
            tx=ses.beginTransaction();
            //Load a Parent
            PersonDetails details=ses.get(PersonDetails.class,1);
            // get All childs of a parent
            Set<PhoneNumber>  childs=details.getPhones();
              childs.removeAll(childs);
            tx.commit();
            System.out.println("all childs of parent are removed");
        }
        catch(HibernateException he) {
             he.printStackTrace();
            if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                tx.rollback();
                System.out.println("Problem in removing childs of a parent");
            }
        }//catch
        catch(Exception e) {
            e.printStackTrace();
        }
    }//meethod
```

=>While deleting all childs or specific child of a parent in One To Many association having cascade="all" it actually does not delete childs records .. in fact makes the childs records the orphan record by breaking their link with parent record i.e assigning null values in fk column of child records belonging to a parent record as show below.



What is orphan record?

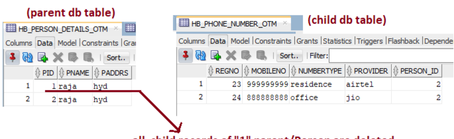
The record in child db table which has lost link with parent table record by getting null values in FK column is called orphan record.

note:: In One To many association , while deleting all childs of a parent or specific child of parent getting orhphan records is not acceptable ..So the value of "cascade" attribute must be changed to "delete-orphan" or "all-delete-orphan" (best)
gives all , delete-orphan behaviour

=>The best of "cascade" attribute in one to many association is "all-delete-orphan"

In PersonDetails.hbm.xml

```
<set name="phones" cascade="all-delete-orphan" lazy="extra" >
    <key column="PERSON_ID"/> <!--  FK column -->
    <one-to-many class="com.nt.entity.PhoneNumber"/> <!--  child class cfg -->
</set>
```

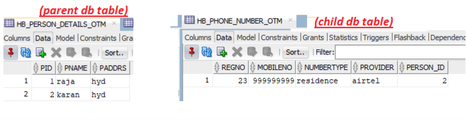


all child records of "1" parent/Person are deleted with out making them as orphan records.

==========
Deleting one child from collection of childs belonging to a parent
==========

=> Can we u access the elements of set collection through index ? no
=> Can we u delete the element of set collection through index ? no
=> how to delete the element of set collection ?
ans) By passing elemement obj itself as argument value for remove(-) method

```
@Override
    public void deleteOneChildFromCollectionOfChildsOfAParent() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(ses){
            tx=ses.beginTransaction();
            //Load a Parent
            PersonDetails details=ses.get(PersonDetails.class,2);
            //get all childs of a parent
            Set<PhoneNumber> childs=details.getPhones();
            //Load Specific child that u want to delete
            PhoneNumber ph=ses.get(PhoneNumber.class, 24);
            childs.remove(ph);
            tx.commit();
            System.out.println("Specific child of parent is deleted");
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                tx.rollback();
                System.out.println("Problem in removing specific child of a parent");
            }//if
        }//catch
    }//method
```



==========
Deleting a parent and its associated child objs
==========

if cascade="all" or "all-delete-orphan" the delete operation peformed on the parent object also takes place on the associated child objs

```
@Override
    public void deleteParentAndItsChilds() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(ses){
            tx=ses.beginTransaction();
              //Load a parent  which leads to loading the childs
              PersonDetails details= ses.get(PersonDetails.class,2);
               //delete parent   which leads deleting the childs
              ses.delete(details);
            tx.commit();
             System.out.println("parent its associated childs are deleted");
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                tx.rollback();
                System.out.println("parent its associated childs are not deleted");
            }//if
        }//catch

    }
```

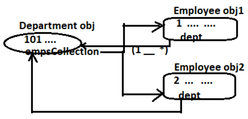
==========
One To Many /Many To one Bi-Directional Association
==========

=>It is one to many from parent prospective and many to one from child prospective

=> Since this is Bi-Directional Association, we can access child objs from parent and parent objs from child

=> To Build this association the parent class should have collection type property to hold bunch of child objs (Building one to many from parent) and Child class should have parent class type property to make multiple child objs refering single parent obj.

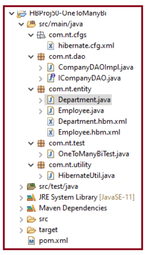
Department ——> Employee (One Department can have multiple employees which one to many from Department prospective ..Similarly Employees can belong single Department that is many to one from Employee prospective)



=>We need to cfg Department class highlighting one to many association

=>Similarly ,We need to cfg Employee class highlighting Many To One association

example code
==========




```
//parent class
------------------
package com.nt.entity;

import java.util.Set;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Department {
	private Integer deptNo;
	private String deptName;
	private String deptHead;
	private Set<Employee> employees; //for one to many association
	@Override
	public String toString() {
		return "Department [deptNo=" + deptNo +
		", deptName=" + deptName + ", deptHead=" + deptHead + "]";
	}

}
```

```
child class
-------------
package com.nt.entity;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Employee {
	private Integer empNo;
	private String empName;
	private Float empSalary;
	private Department dept; //for many to one association

	@Override
	public String toString() {
		return "Employee [empNo=" + empNo + ", empName=" +
		empName + ", empSalary=" + empSalary + "]";
	}
}//class
```

```
Department.hbm.xml
---------------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Department" table="BI_OTM_DEPARTMENT">
	<id name="deptNo" column="DEPTNO">
	  <generator class="increment"/>
	</id>
	<property name="deptName" column="DEPTNAME" length="15"/>
	<property name="deptHead" column="DEPTHEAD" length="15"/>
	<set name="employees" cascade="all-delete-orphan" lazy="extra">
	  <key column="DEPT_ID"/>
	  <one-to-many class="com.nt.entity.Employee"/>
	</set>
  </class>
</hibernate-mapping>
```

```
Employee.hbm.xml
----------------------
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Employee" table="BI_OTM_EMPLOYEE">
	<id name="empNo" column="EMPNO">
	  <generator class="sequence">
	    <param name="sequence_name">EID_SEQ</param>
	    <param name="initial_value">1000</param>
	    <param name="increment_size">1</param>
	  </generator>
	</id>
	<property name="empName" column="EMPNAME" length="20"/>
	<property name="empSalary" column="EMPSALARY"/>
	<many-to-one name="dept" class="com.nt.entity.Department"
	  column="DEPT_ID" cascade="all" lazy="proxy"/>
<!-- (all possible values are : false proxy ,no-proxy for lazy ) -->
<!-- (No possibility of getting orphan records so "all-delete-orphan", "delete-orphan" is not
  required for "cascade" attribute ) -->
	</class>
</hibernate-mapping>
```

```
DAO Impl class

package com.nt.dao;

import java.util.Set;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.Department;
import com.nt.entity.Employee;
import com.nt.utility.HibernateUtil;

public class CompanyDAOImpl implements ICompanyDAO {

	@Override
	public void saveDataUsingParent() {
		Session ses=HibernateUtil.getSession();
		Transaction tx=null;
		try(ses){
			//begin Tx
			tx=ses.beginTransaction();
			 //prepare parent objs
			 Department dept=new Department();
			 dept.setDeptName("IT"); dept.setDeptHead("JAMES");
			 //prepare child objs
			 Employee emp1=new Employee();
			 emp1.setEmpName("rajesh"); emp1.setEmpSalary(898989.0f);
			 Employee emp2=new Employee();
			 emp2.setEmpName("ramesh"); emp2.setEmpSalary(898989.0f);
			 //set childs to parent (for one to many)
			 dept.setEmployees(Set.of(emp1,emp2));
			 //set parent to childs (for many to one)
			 emp1.setDept(dept); emp2.setDept(dept);

			 //save through parent
			 ses.save(dept);
			tx.commit();
			System.out.println("parent to childs are saved");
		}
		catch(HibernateException he) {
			he.printStackTrace();
			if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Problem in saving objs (parent to childs)");
			}
		}//catch
	}//method

	@Override
	public void saveDataUsingChild() {
		Session ses=HibernateUtil.getSession();
		Transaction tx=null;
		try(ses){
			//begin Tx
			tx=ses.beginTransaction();
			 //prepare parent objs
			 Department dept=new Department();
			 dept.setDeptName("ACCOUNTS"); dept.setDeptHead("JHON");
			 //prepare child objs
			 Employee emp1=new Employee();
			 emp1.setEmpName("suresh"); emp1.setEmpSalary(818981.0f);
			 Employee emp2=new Employee();
			 emp2.setEmpName("naresh"); emp2.setEmpSalary(828282.0f);
			 //set childs to parent (for one to many)
			 dept.setEmployees(Set.of(emp1,emp2));
			 //set parent to childs (for many to one)
			 emp1.setDept(dept); emp2.setDept(dept);

			 //save through childs
			 ses.save(emp1); ses.save(emp2);
			tx.commit();
			System.out.println(" childs to parent are saved");
		}
		catch(HibernateException he) {
			he.printStackTrace();
			if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
				tx.rollback();
				System.out.println("Problem in saving objs ( childs to parent)");
			}
		}//catch

	}//method
}//class
```

```
Client App
==========
package com.nt.test;

import com.nt.dao.CompanyDAOImpl;
import com.nt.dao.ICompanyDAO;
import com.nt.utility.HibernateUtil;

public class OneToManyBiTest {

	public static void main(String[] args) {
		ICompanyDAO dao=new CompanyDAOImpl();
		 //dao.saveDataUsingParent();
		dao.saveDataUsingChild();

		 //close SessionFactory
		 HibernateUtil.closeSessionFactory();
	}//method
}//class
```

BI_OTM_EMPLOYEE | BI_OTM_DEPARTMENT

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Fla

Sort.. | Filter:

(pk)	DEPTNO	DEPTNAME	DEPTHEAD
1	1	IT	JAMES
2	2	ACCOUNTS	JHON

BI_OTM_EMPLOYEE

Columns | Data | Model | Constraints | Grants | Statistics | Triggers | Flashback | Dep

Sort.. | Filter:

(pk)	EMPNO	EMPNAME	EMPSALARY	DEPT_ID (fl)
1	1004	ramesh	898989	1
2	1005	rajesh	898989	1
3	1007	naresh	828282	2
4	1006	suresh	818981	2

what is inverse="true" in bi-directional association ?

=>By placing inverse="true" in bi-diretional association we are informing the hibernate the opposite side of the association reverse of current side of association.. so do not build it from scratch level by generating unnecessary queries .. just build the opposite side of the association as the reverse of the existing association.

=> we need to place inverse="true"

In one to many/many to one association one side (parent side)

In one to one / many to many association any side

In Department -- Employee example we need to place inverse="true" in parent side collection tag <list>/<set>/<map>/<bag> tags

```
<set name="employees" cascade="all-delete-orphan" lazy="extra" inverse="true">
    <key column="DEPT_ID"/>
    <one-to-many class="com.nt.entity.Employee"/>
</set>
```

SElect operations in One to many Bi-Driectional Assoication

==========

=> we can select records using parent and we can also select the records using child

```
@Override
public void loadDataUsingParent() {
    Session ses=HibernateUtil.getSession();
    try(ses){
        //Load parents and the associated childs
        Query query=ses.createQuery("from Department");
        List<Department> list=query.getResultList();
        list.forEach(dept->{
            System.out.println("parent ::"+dept);
            Set<Employee> childs=dept.getEmployees();
            /*childs.forEach(emp->{
                System.out.println(emp);
            });*/
        });
    }//try
    catch(HibernateException he) {
        he.printStackTrace();
    }
    catch(Exception e) {
        e.printStackTrace();
    }

}//method
```

In select operations of one to many association the possible lazy loading values are lazy="true" ,lazy="extra" , lazy="false"

lazy="false" --> For eager loading
lazy="true" --> For lazy loading
lazy="extra" --> For extra lazy loading

```
@Override
public void loadDataUsingChild() {
    Session ses=HibernateUtil.getSession();
    try(ses){
        //Load parents and the associated childs
        Query query=ses.createQuery("from Employee");
        List<Employee> list=query.getResultList();
        list.forEach(emp->{
            System.out.println("child ::"+emp);
            Department dept=emp.getDept();
            System.out.println("parent::"+dept);
        });
    }//try
    catch(HibernateException he) {
        he.printStackTrace();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}
```

In Many to One Select operations the possible values for lazy attribute are lazy="proxy" , lazy="false", lazy="no-proxy"

lazy="fasle" --> performs eager loading (parent childs will be loaded along with child objs)

lazy="proxy" --> performs lazy loading by generating proxy obs (deafult -best) (child objs will be loaded normally ..having proxy objs for parent objs but the associated parent objs will be loaded lazily through those proxy objs)

lazy="no-proxy" --> Performs lazy loading but no proxy objs will be generated. For this we need to specify proxy interface implemented by the parent class in the proxy attribute <class> tag where parent class is cfg (same as ses.load(-,-) method's proxy interface concept)

Code improvisions in One to many / many to one bi-directional association example App for lazy="no-proxy" value

==========

step1) place lazy="no-proxy" in <many-to-one> tag of Employee.hbm.xml file

```
<many-to-one name="dept" class="com.nt.entity.Department"
    column="DEPT_ID" cascade="all" lazy="no-proxy" />
```

lazy loading with out proxy objects

step2) make parent class implementing Proxy Interface and specify that proxy interface in proxy attribute of <class> in Department.hbm.xml file

IDepartment.java

```
package com.nt.entity;

import java.util.Set;

public interface IDepartment {
    public Integer getDeptNo();
    public void setDeptNo(Integer deptNo);
    public void setDeptName(String deptName);
    public String getDeptName();
    public void setDeptHead(String deptHead);
    public String getDeptHead();
    public void setEmployees(Set<Employee> list);
    public Set<Employee> getEmployees();
}
```

Department.java

```
@Setter
@Getter
public class Department implements IDepartment {
    private Integer deptNo;
    private String deptName;
    private String deptHead;
    private Set<Employee> employees; //for one to many association

    public Department() {
        System.out.println("Department:: 0-param constructor"+this.getClass());
    }

    @Override
    public String toString() {
        return "Department [deptNo=" + deptNo + ", deptName=" + deptName + ", deptHead=" + deptHead + "]";
    }


}
```

In Department.hbm.xml

```
<class name="com.nt.entity.Department" table="BI_OTM_DEPARTMENT" proxy="com.nt.entity.IDepartment">
  ...
</class>
```

tep3) Do the following chage in child entity class

```
@Setter
@Getter
public class Employee {
    private Integer empNo;
    private String empName;
    private Float empSalary;
    private IDepartment dept; //for many to one association

    public Employee() {
      System.out.println("Employee:: 0-param constructor"+this.getClass());
    }

    @Override
    public String toString() {
        return "Employee [empNo=" + empNo + ", empName=" + empName + ", empSalary=" + empSalary + "]";
    }
}//class
```

step4) write the collowing code in DAO impl class

```
@Override
public void loadDataUsingChildWithProxy() {
    Session ses=HibernateUtil.getSession();
    try(ses){
        //Load parents and the associated childs
        Query query=ses.createQuery("from Employee");
        List<Employee> list=query.getResultList();
        list.forEach(emp->{
            System.out.println("child ::"+emp);
            IDepartment dept=emp.getDept();
            System.out.println("parent::"+dept);
        });
    }//try
    catch(HibernateException he) {
        he.printStackTrace();
    }
    catch(Exception e) {
        e.printStackTrace();
    }

}
```

Many To Many Association
==========

=>It is bi-directional Association
=> It one to many association from both sides i.e each parent will have set of child objs and each child will have set of parent objs.

=> To build association through entity classes , the parent class should collection type property to hold set of child objs and child class should also have collection property to hold set of parent objs
=> To build this association through db tables , we need to take total 3 db tables

table1 :: To hold parent records
table2 :: To hold child records
table3 :: Association table where the relatiohship will be build

eg:: Doctors and patients , Programmers and Projects , Faculties and students

Example db tables

Doctors (table1)

docId(pk)	docName	hospital
1	raja	apollo
2	ramesh	care

Patients (table2)

patId(pk)	patName	problem
1001	ramesh	heart
1002	suresh	kidney
1003	anil	corona

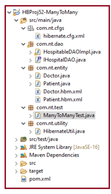
Doctors_Patients (table3 --Association table)

doctor_id(fk)	patient_id(fk)
1	1001
1	1002
2	1001
2	1002
2	1003

composite PK

note :: In hibernate to build many to many association we need to take 2 entity classes and 3 db tables

```
//parent class
package com.nt.entity;

import java.util.Set;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Doctor {
        private Integer docId;
        private String docName;
        private String hospital;
        private Set<Patient> patients;  //To  hold bunch of patients
        public Doctor() {
                System.out.println("Doctor:: 0-param constructor");
        }
        @Override
        public String toString() {
                return "Doctor [docId=" +
                        docId + ", docName=" + docName + ", hospital=" + hospital + "]";

        }
}
```




```
//Child class
package com.nt.entity;

import java.util.Set;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Patient {
        private Integer patId;
        private String patName;
        private String problem;
        private Set<Doctor> doctors;  //To hold bunch of doctors

        public Patient() {
                System.out.println("Patient:: 0-param constructor");
        }

        @Override
        public String toString() {
                return "Patient [patId=" +
                patId + ", patName=" + patName + ", problem=" + problem + "]";
        }

}//class
```

Doctor.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class  name="com.nt.entity.Doctor"  table="DOCTORS">
    <id name="docId"  column="DOCID" >
      <generator class="increment"/>
    </id>
    <property name="docName" column="DOCNAME" length="15"/>
    <property name="hospital"  column="HOSPITAL"  length="15"/>
    <set name="patients" table="DOCTORS_PATIENTS" cascade="all"  lazy="extra" inverse="true">
      <key column="DOCTOR_ID"/>  <!--owning side FK column  -->
      <many-to-many  class="com.nt.entity.Patient" column="PATIENT_ID"/>  <!-- Non owning FK column -->
    </set>

  </class>
</hibernate-mapping>
```

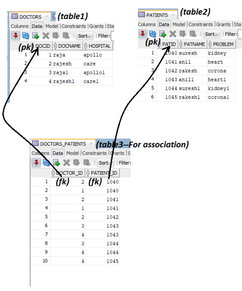
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class  name="com.nt.entity.Patient"  table="PATIENTS">
    <id name="patId"  column="PATID" >
      <generator class="sequence">
        <param name="sequence_name">PAT_SEQ</param>
        <param name="increment_size">1</param>
        <param name="initial_value">1000</param>
      </generator>
    </id>
    <property name="patName" column="PATNAME" length="15"/>
    <property name="problem"  column="PROBLEM"  length="15"/>
    <set name="doctors" table="DOCTORS_PATIENTS" cascade="all"  lazy="extra" >
      <key column="PATIENT_ID"/>  <!--owning side FK column  -->
<many-to-many  class="com.nt.entity.Doctor" column="DOCTOR_ID"/>  <!-- Non owning FK column -->
    </set>

  </class>
</hibernate-mapping>
```

```
package com.nt.dao;

import java.util.List;
import java.util.Set;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.Doctor;
import com.nt.entity.Patient;
import com.nt.utility.HibernateUtil;

public class HospitableDAOImpl implements IHospitalDAO {

    @Override
    public void saveDataUsingParents() {
    Session ses=HibernateUtil.getSession();
    Transaction tx=null;
    try(ses){
      //prepare parent objs
      Doctor doc1=new Doctor();
      doc1.setDocName("raja"); doc1.setHospital("apollo");
      Doctor doc2=new Doctor();
      doc2.setDocName("rajesh"); doc2.setHospital("care");
      //prepare child objs
      Patient pat1=new Patient();
      pat1.setPatName("anil"); pat1.setProblem("heart");
      Patient pat2=new Patient();
      pat2.setPatName("suresh"); pat2.setProblem("kidney");
      Patient pat3=new Patient();
      pat3.setPatName("rakesh"); pat3.setProblem("corona");
      // assign  childs to parents
      doc1.setPatients(Set.of(pat1,pat2));
      doc2.setPatients(Set.of(pat1,pat2,pat3));
      // assign   parents to childs
      pat1.setDoctors(Set.of(doc1,doc2));
      pat2.setDoctors(Set.of(doc1,doc2));
      pat3.setDoctors(Set.of(doc2));
      //begin tx
      tx=ses.beginTransaction();
        //save using parents
          ses.save(doc1);ses.save(doc2);
       tx.commit();
       System.out.println("Objects are saved");
    }//try
    catch(HibernateException he) {
      he.printStackTrace();
      if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
            tx.rollback();
            System.out.println("Problem is  saving objs");
      }
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    }//method

    @Override
    public void saveDataUsingChilds() {
            Session ses=HibernateUtil.getSession();
    Transaction tx=null;
    try(ses){
      //prepare parent objs
      Doctor doc1=new Doctor();
      doc1.setDocName("raja1"); doc1.setHospital("apollo1");
      Doctor doc2=new Doctor();
      doc2.setDocName("rajesh1"); doc2.setHospital("care1");
      //prepare child objs
      Patient pat1=new Patient();
      pat1.setPatName("anil1"); pat1.setProblem("heart1");
      Patient pat2=new Patient();
      pat2.setPatName("suresh1"); pat2.setProblem("kidney1");
      Patient pat3=new Patient();
      pat3.setPatName("rakesh1"); pat3.setProblem("corona1");
      // assign  childs to parents
      doc1.setPatients(Set.of(pat1,pat2));
      doc2.setPatients(Set.of(pat1,pat2,pat3));
     // assign   parents to childs
      pat1.setDoctors(Set.of(doc1,doc2));
      pat2.setDoctors(Set.of(doc1,doc2));
      pat3.setDoctors(Set.of(doc2));
      //begin tx
      tx=ses.beginTransaction();
        //save using childs
          ses.save(pat1); ses.save(pat2); ses.save(pat3);
       tx.commit();
       System.out.println("Objects are saved");
    }//try
    catch(HibernateException he) {
      he.printStackTrace();
      if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
            tx.rollback();
            System.out.println("Problem is  saving objs");
      }
    }
    catch(Exception e) {
      e.printStackTrace();
    }

    }//method

    @Override
    public void loadDataUsingParent() {
            Session ses=HibernateUtil.getSession();
    Transaction tx=null;
    try(ses){
      Query query=ses.createQuery("from Doctor");
      List<Doctor> list=query.getResultList();
      list.forEach(doc->{
            System.out.println("parent::"+doc);
            Set<Patient> childs=doc.getPatients();
            childs.forEach(pat->{
                    System.out.println("child::"+pat);
            });
      });
    }//try
    catch(HibernateException he) {
      he.printStackTrace();
    }
    catch(Exception e) {
      e.printStackTrace();
    }

    }//method

}//class
```

DAO interface

```
package com.nt.dao;

public interface IHospitalDAO {
    public void saveDataUsingParents();
    public void saveDataUsingChilds();
    public void loadDataUsingParent();
}
```

db tables



Cleint App

```
package com.nt.test;

import com.nt.dao.HospitableDAOImpl;
import com.nt.dao.IHospitalDAO;
import com.nt.utility.HibernateUtil;

public class ManyToManyTest {

        public static void main(String[] args) {
                // create DAO class obj
                IHospitalDAO dao=new  HospitableDAOImpl();
                  //dao.saveDataUsingParents();
                  //dao.saveDataUsingChilds();
                dao.loadDataUsingParent();
        //close Session Factory
                HibernateUtil.closeSessionFactory();
        }

}
```

One to One Association
==========

This can be implemented in two ways
a) As One to One PK (if Both parent and child objs are ready to use same id value)
b) As One to One FK (if Both parent and child objs are not ready to use same id value)

a) As One to One PK

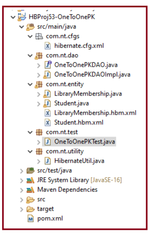
=> Here parent obj and the associated child obj share the same id value i.e either child obj use parent obj's id value or parent obj use child obj's id value.

=> Here FK column is not required becoz One side PK column itself acts FK colum becoz Child obj id value (pk col vlaue) will be gathered from the associated Parent obj (id value). So here FK column is not required .. In fact "foreign" generator is required
note:: "foreign" generator is given to get current obj's id value from the associated parent obj's id value

usecases :: 1)Student and LibrayMembership (Student id will be used as the LibraryMembership id)
2) FlatOwner and ClubHouseMemberShip (Flat no will be used as the ClubHouseMembership id)
3) Person and IncomeTaxFiling (Person's panNo will be utilized as the TxFiling id value)
and etc..

Student.java (parent class)

```
package com.nt.entity;

import java.io.Serializable;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Student implements Serializable {
        private  Integer sno;
        private String sname;
        private  String sadd;
        private LibraryMembership  libraryDetails;

        public Student() {
                System.out.println("Student:: 0-param constructor");
        }

        @Override
        public String toString() {
                return "Student [sno=" + sno + ", sname=" + sname + ", sadd=" + sadd + "]";
        }

}
```



LibraryMembeship.java (child class)

```
package com.nt.entity;

import java.io.Serializable;
import java.time.LocalDate;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class LibraryMembership  implements Serializable {
        private Integer lid;
        private String type;
        private  LocalDate  doj;
        private  Student  studentDetails;

        public LibraryMembership() {
                System.out.println("LibraryMembership:: 0-param constructor");
        }


        @Override
        public String toString() {
                return "LibraryMembership [lid=" + lid + ", type=" + type + ", doj=" + doj + "]";
        }
}
```

Student.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class  name="com.nt.entity.Student"  table="OTOPK_STUDENT">
     <id name="sno"  >
       <generator class="increment"/>
     </id>
     <property name="sname"  length="15"/>
     <property name="sadd" length="15"/>
     <one-to-one name="libraryDetails"
                 class="com.nt.entity.LibraryMembership"
                 cascade="all"
                 lazy="proxy" />
   </class>
</hibernate-mapping>
```

No FK colum here but PK column child table also acts as FK colum i.e no way of getting null value and orphan records so the best value for cascade is "all"

LibraryMembership.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
   <class  name="com.nt.entity.LibraryMembership"  table="OTOPK_LIBRARYMEMBERSHIP">
     <id name="lid">
       <generator class="foreign">
         <param name="property">studentDetails</param>  <!-- HAS -A property name -->
       </generator>
     </id>
     <property name="type" length="15"/>
     <property name="doj"/>
       <one-to-one name="studentDetails" class="com.nt.entity.Student" cascade="all" lazy="proxy"/>
   </class>
</hibernate-mapping>
```

=>"foreign" generator gets associated parent obj through "studentDetails" property and collects its id value(parent) and makes it as current objs(child) id value

The possible values for lazy are "proxy","no-proxy","false{ the best value is "proxy".

DAO Impl class

```
package com.nt.dao;

import java.time.LocalDate;
import java.util.List;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.LibraryMembership;
import com.nt.entity.Student;
import com.nt.utility.HibernateUtil;

public class OneToOnePKDAOImpl implements OneToOnePKDAO {

        @Override
        public void saveDataUsingParent() {
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(ses){
                        //begin tx
                        tx=ses.beginTransaction();
                        //prepare objs
                        Student stud=new Student();
                          stud.setSname("raja"); stud.setSadd("hyd");
                          LibraryMembership lib=new LibraryMembership();
                          lib.setType("gold"); lib.setDoj(LocalDate.now());
                          //set parent to child
                          lib.setStudentDetails(stud);
                          stud.setLibraryDetails(lib);
                          //save objs
                           ses.save(stud);
                        tx.commit();
                        System.out.println("Objects are saved");

                }//try
                catch(HibernateException he) {
                        he.printStackTrace();
         if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
          tx.rollback();
          System.out.println("Problem in objs saving");
         }
                }//catch

        }//method

        @Override
        public void saveDataUsingChild() {
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(ses){
                        //begin tx
                        tx=ses.beginTransaction();
                        //prepare objs
                        Student stud=new Student();
                          stud.setSname("suresh"); stud.setSadd("vizag");
                          LibraryMembership lib=new LibraryMembership();
                          lib.setType("silver"); lib.setDoj(LocalDate.now());
                          //set parent to child
                          lib.setStudentDetails(stud);
                          stud.setLibraryDetails(lib);
                          //save objs
                           ses.save(lib);
                        tx.commit();
                        System.out.println("Objects are saved");
                }//try
                catch(HibernateException he) {
                        he.printStackTrace();
         if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
          tx.rollback();
          System.out.println("Problem in objs saving");
         }
                }//catch
        }//method

        @Override
        public void loadDataUsingParent() {
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try(ses){
                        Query query=ses.createQuery("from Student");
                        List<Student> list=query.getResultList();
                        list.forEach(stud->{
                                System.out.println("parent::"+stud);
                                LibraryMembership lib=stud.getLibraryDetails();
                                System.out.println("child::"+lib);
                        });
                }//try
                catch(HibernateException he) {
                        he.printStackTrace();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }//method

}//class
```

DAO Interface

```
package com.nt.dao;

public interface OneToOnePKDAO {
    public void  saveDataUsingParent();
    public void  saveDataUsingChild();
    public void loadDataUsingParent();
}
```

```
package com.nt.test;

import com.nt.dao.OneToOnePKDAO;
import com.nt.dao.OneToOnePKDAOImpl;
import com.nt.utility.HibernateUtil;

public class OneToOnePKTest {

        public static void main(String[] args) {
                //create DAO class obj
                  OneToOnePKDAO  dao=new OneToOnePKDAOImpl();
                  //dao.saveDataUsingParent();
                  //dao.saveDataUsingChild();
                  dao.loadDataUsingParent();
        //close Session Factory
                  HibernateUtil.closeSessionFactory();
        }

}
```

One to One FK association

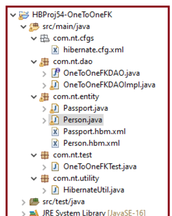
=>if parent and child objs are not ready to use id value then go for one to one FK Association
=> Here Parent obj contains its own id value .. child obj contains its own id value but the child obj will be linked to the parent obj using FL column..

eg:: Person and Passport (pids and passport nos are different but passport will be linked to Person using FK column. Every person need not to have passport but every passport belongs to a Person)

eg:: Person and DrivingLicense (pids and license nos are different but License will be linked to Person using FK column. Every person need not to have DrivingLicense but every License belongs to a Person)

=> In <one-to-one> tag there is no provision to specify FK column becoz it is given to build one to one pk association.. So use <many-to-one> having not-null=true, unique="true" to build One to one FK association.

One to one FK Child to Parent association (uni-directional association)



Person.java

```
package com.nt.entity;

import java.io.Serializable;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Person implements Serializable{
        private Integer pid;
        private String pname;
        private String paddrs;

        public Person() {
                System.out.println("Person:: 0-param constructor");
        }

        @Override
        public String toString() {
                return "Person [pid=" + pid + ", pname=" + pname + ", paddrs=" +
paddrs + "]";
        }
}
```

Passport.java

```
package com.nt.entity;

import java.io.Serializable;
import java.time.LocalDate;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
public class Passport implements Serializable {
        private Integer passportNo;
        private String country;
        private LocalDate expiryDate;
        private Person personDetails; //for OneToOne Association
        public Passport() {
                System.out.println("Passport:: 0-param constructor");
        }
        @Override
        public String toString() {
                return "Passport [passportNo=" + passportNo +
        }       ", country=" + country + ", exipiryDate=" + expiryDate + "]";
}
```

Person.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <class name="com.nt.entity.Person" table="AM_OTOFK_PERSON">
    <id name="pid">
      <generator class="increment"/>
    </id>
    <property name="pname" length="20"/>
    <property name="paddrs" length="20"/>
  </class>
</hibernate-mapping>
```

Passport.hbm.xml

```
<hibernate-mapping>
  <class name="com.nt.entity.Passport" table="AM_OTOFK_PASSPORT">
    <id name="passportNo">
      <generator class="sequence">
        <param name="sequence_name">pspt_seq</param>
        <param name="initial_value">1000</param>
        <param name="increment_size">1</param>
      </generator>
    </id>
    <property name="country" length="20"/>
    <property name="expiryDate" />
    <many-to-one name="personDetails" class="com.nt.entity.Person"
                 column="PERSON_ID"
                 not-null ="true"
                 unique="true"
                 cascade="all"
                 lazy="proxy"/>
  </class>
</hibernate-mapping>
```

DAO Impl class

```
//DAOImpl class
package com.nt.dao;

import java.time.LocalDate;
import java.util.List;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.Passport;
import com.nt.entity.Person;
import com.nt.utility.HibernateUtil;

public class OneToOneFKDAOImpl implements OneToOneFKDAO {


    @Override
    public void saveDataUsingChild() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try(ses){
            //begin tx
            tx=ses.beginTransaction();
            //prepare objs
              Person per=new Person();
              per.setPname("raja"); per.setPaddrs("hyd");
              Passport pspt=new Passport();
              pspt.setCountry("India"); pspt.setExpiryDate(LocalDate.of(2030,10, 2));
              //set parent to child
              pspt.setPersonDetails(per);
             //save objs
              ses.save(pspt);
            tx.commit();
            System.out.println("Objects are saved");
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
    if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
     tx.rollback();
     System.out.println("Problem in objs saving");
    }
        }//catch
    }//method

    @Override
    public void loadDataUsingChild() {
        Session ses=HibernateUtil.getSession();
        try(ses){
            Query query=ses.createQuery("from Passport");
            List<Passport> list=query.getResultList();
            list.forEach(pspt->{
                System.out.println("child::"+pspt);
                Person per=pspt.getPersonDetails();
                System.out.println("parent::"+per);
            });
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }

    }//method


}//class
```

Client App

```
package com.nt.test;

import com.nt.dao.OneToOneFKDAO;
import com.nt.dao.OneToOneFKDAOImpl;
import com.nt.utility.HibernateUtil;

public class OneToOneFKTest {

    public static void main(String[] args) {
        //create DAO class obj
          OneToOneFKDAO dao=new OneToOneFKDAOImpl();
          //dao.saveDataUsingChild();
           dao.loadDataUsingChild();
    //close Session Factory
          HibernateUtil.closeSessionFactory();
    }

}
```

DAO Interface

```
package com.nt.dao;

public interface OneToOneFKDAO {
    public void saveDataUsingChild();
    public void loadDataUsingChild();

}
```

Annotation driven Association Mapping

==========

comonly used annotations ::

=>@OneToMany , @ManyToOne ,@OneToOne ,@ManyToMany :: To build the assoication mapping
=>@JoinColumn --> To specify FK column pointing to the PK column
=>@PrimaryKeyJoinColumn -->To specify PK column as FK column (Required in One to one PK)
=>@JoinTable ---> To specify the association db table (Third db table many to many association)
an etc..

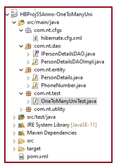
Annotation driven Association Mapping

```
==================================
 comonly used annotations ::
 ---------------------------
    =>@OneToMany , @ManyToOne ,@OneToOne ,@ManyToMany :: To build the assoication mapping
    =>@JoinColumn  --> To specify FK column pointing to the PK column
    =>@PrimaryKeyJoinColumn -->To specify PK column as FK column (Required in One to one PK)
    =>@JoinTable  --> To specify the association db table  (Third db table  many to many  association)
    an etc..

     ------------------------------------------------------------
     One to  Many  Uni -directional Association  (Annotation driven)
     ------------------------------------------------------------
       Annotations required are::
         @OneToMany  :: To  build the association
         @JoinColumn  ::  To specify the FK column pointing to PK column
```

JPA Annoations

DAO Interface

```
package com.nt.dao;

public interface IPersonDetailsDAO {
    public void saveDataUsingParent();
    public void   loadDataUsingParent();
    public void  addChildToExistingParent();
    public void  deleteAllChildsOfAParent();
    public void deleteOneChildFromCollectionOfChildsOfAParent();
    public void  deleteParentAndItsChilds();
}
```



DAOImpl class

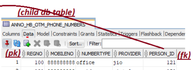
```
package com.nt.dao;

import java.util.List;
import java.util.Set;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.PersonDetails;
import com.nt.entity.PhoneNumber;
import com.nt.utility.HibernateUtil;

public class PersonDetailsDAOImpl implements IPersonDetailsDAO {

    @Override
    public void saveDataUsingParent() {
        //get Accesss to Session object
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try{ses){
            tx=ses.beginTransaction();
              //prepare  parent obj
              PersonDetails  pDetails=new PersonDetails();
              pDetails.setPname("raja"); pDetails.setPaddrs("hyd");
              //prepare child objs
              PhoneNumber ph1=new PhoneNumber();
              ph1.setNumberType("residence"); ph1.setProvider("airtel");
              ph1.setMobileNo(999999999L);

              PhoneNumber ph2=new PhoneNumber();
              ph2.setNumberType("office"); ph2.setProvider("jio");
              ph2.setMobileNo(888888888L);
              //set child objs to parent obj
              pDetails.setPhones(Set.of(ph1,ph2));
              //save parent obj
               int idVal=(Integer)ses.save(pDetails);
               System.out.println("Generated id value::"+idVal);
            tx.commit();
            System.out.println("Parent and the associated childs are saved");
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
                tx.rollback();
                System.out.println("Parent and the associated childs are not saved");
            }//if
        }//catch
        catch(Exception e) {
            e.printStackTrace();
        }
    }//method

    @Override
    public void loadDataUsingParent() {
        //get Accesss to Session object
        Session ses=HibernateUtil.getSession();
        try{ses){
            Query  query=ses.createQuery("from PersonDetails");
            List<PersonDetails> list=query.getResultList();
            list.forEach(pd->{
              System.out.println("parent ::"+pd);
                Set<PhoneNumber>  childs=pd.getPhones();
                System.out.println("childs count::"+childs.size());
                /*childs.forEach(ph->{
                 System.out.println("child::"+ph);
                });
                 System.out.println(".............");*/
            });
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
        }//catch
        catch(Exception e) {
            e.printStackTrace();
        }

    }//method

    @Override
    public void addChildToExistingParent() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try{ses){
            tx=ses.beginTransaction();
            //Load existing parent
      PersonDetails pd=ses.get(PersonDetails.class, 121);
      //get all childs of parent
      Set<PhoneNumber> childs=pd.getPhones();
      //create new Child obj
      PhoneNumber ph=new PhoneNumber();
      ph.setMobileNo(888887777L);
      ph.setNumberType("office");
      ph.setProvider("vi");
      //add new child to existing parent
      childs.add(ph);
            tx.commit();
            System.out.println("Child is added to existing childs of a parent");
        }
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                tx.rollback();
            }
        }//catch
        catch(Exception e) {
            e.printStackTrace();
        }
}//method

    @Override
    public void deleteAllChildsOfAParent() {
        Session ses=HibernateUtil.getSession();
        Transaction tx=null;
        try{ses){
            tx=ses.beginTransaction();
            //Load a Parent
            PersonDetails details=ses.get(PersonDetails.class,121);
            // get All childs of a parent
            Set<PhoneNumber>  childs=details.getPhones();
            //remove all childs
             childs.removeAll(childs);
            tx.commit();
            System.out.println("all childs of parent are removed");
        }
        catch(HibernateException he) {
            he.printStackTrace();
            if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                tx.rollback();
                System.out.println("Problem in removing childs of a parent");
            }
        }//catch
        catch(Exception e) {
            e.printStackTrace();
        }
}//meethod

    @Override
    public void deleteOneChildFromCollectionOfChildsOfAParent() {
        Session ses=HibernateUtil.getSession();

                Transaction tx=null;
                try{ses){
                    tx=ses.beginTransaction();
                    //Load a Parent
                    PersonDetails details=ses.get(PersonDetails.class,2);
                    //get all childs of a parent
                    Set<PhoneNumber> childs=details.getPhones();
                    //Load Specific child that u want to delete
                    PhoneNumber ph=ses.get(PhoneNumber.class, 24);
                    childs.remove(ph);
                    tx.commit();
                    System.out.println("Specific child of parent is deleted");
                }//try
                catch(HibernateException he) {
                    he.printStackTrace();
                    if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                        tx.rollback();
                        System.out.println("Problem in removing specific child of a parent");
                    }//if
                }//catch
            }//method

            @Override
            public void deleteParentAndItsChilds() {
                Session ses=HibernateUtil.getSession();
                Transaction tx=null;
                try{ses){
                    tx=ses.beginTransaction();
                      //Load a parent  which leads to loading the childs
                      PersonDetails details= ses.get(PersonDetails.class,122);
                       //delete parent   which leads deleting the childs
                      ses.delete(details);
                    tx.commit();
                    System.out.println("parent its associated childs are deleted");
                }//try
                catch(HibernateException he) {
                    he.printStackTrace();
                    if(tx!=null || tx.getRollbackOnly() || tx.getStatus()!=null) {
                        tx.rollback();
                        System.out.println("parent its associated childs are not deleted");
                    }//if
                }//catch

            }

    }//class
```

```
// PersonDetails.java (parent class)
package com.nt.entity;

import java.io.Serializable;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="ANNO_HB_OTM_PERSON_DETAILS")
public class PersonDetails  implements Serializable {
    @Id
    @GeneratedValue
    private Integer  pid;
    @Column(length=20)
    private String  pname;
    @Column(length=20)
    private String paddrs;
    //special property to hold  bunch of child class objs (one to many association)
    @OneToMany(targetEntity = PhoneNumber.class,cascade = CascadeType.ALL,
                    fetch = FetchType.LAZY,orphanRemoval = true)
    @JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
    private  Set<PhoneNumber> phones;

    public PersonDetails() {
        System.out.println("PersonDetails:: 0-param constructor"+this.getClass());
    }


    //toString
    @Override
    public String toString() {
        return "PersonDetails [pid=" + pid + ", pname=" + pname + ", paddrs=" + paddrs + "]";
    }
}
```

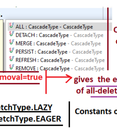
```
//PhoneNumber.java (child class)
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Table(name="ANNO_HB_OTM_PHONE_NUMBERS")
@Entity
public class PhoneNumber implements Serializable{
    @SequenceGenerator(name = "gen1",sequenceName = "PHONE_SEQ",initialValue = 100,allocationSize = 1)
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    @Id
    private Integer regNo;
    private  Long  mobileNo;
    @Column(length = 20)
    private  String numberType;
    @Column(length = 20)
    private  String  provider;

    //taking  property for FK column is pure optional
    @Override
    public String toString() {
        return "PhoneNumber [regNo=" + regNo + ", mobileNo=" + mobileNo + ", numberType=" + numberType + ", provider="
                + provider + "]";
    }

}
```

(parent db table)

(child db table)



The possible vlaues for cascade attribute are CascadeType.



Constants of CascadeType enum

cascade=CascadeType.ALL + orphanRemoval=true → gives the effect of all-delete-orphan

The possible values for fetch are FetchType.LAZY, FetchType.EAGER — Constants of FetchType enum

The fetch attribute @OneToMany Association given by JPA allows us to work with FetchType.LAZY or FetchType.EAGER but to bring the effect of EXTRA,TRUE,FALSE we need to use @LazyCollection annotation

In PhoneDetails.java

```
//special property to hold  bunch of child class objs (one to many association)
        @OneToMany(targetEntity = PhoneNumber.class,cascade = CascadeType.ALL,
                    fetch = FetchType.EAGER,orphanRemoval = true)
        @JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
        @LazyCollection(LazyCollectionOption.EXTRA) ]for extra lazy loading)
        private  Set<PhoneNumber> phones;   TRUE  (for lazy loading)
                                            FALSE  (for eager loading)
```

HB Annotation

(TRUE/FALSE/EXTRA)

Q) If jpa fetch type(EAGER/LAZY) is different from HB LazyCollection then which will be effected?

Ans) The HB LazyCollection (TRUE/FALSE/EXTRA) will be applied)

note:: Without orphanRemoval="true" if we try to delete either all childs of parent or specific child of parent then will not really delete child record.. In fact It will make fk colum value empty to cut child records relation with parent records nothing but makes the records as the orphan records
So it is always recomanded to enable orphanRemove="true"

```
//special property to hold  bunch of child class objs (one to many association)
        @OneToMany(targetEntity = PhoneNumber.class,cascade = CascadeType.ALL,
                    fetch = FetchType.EAGER,orphanRemoval = true)
        @JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
        @LazyCollection(LazyCollectionOption.TRUE)
        private  Set<PhoneNumber> phones;
```

Client app

```
package com.nt.test;

import com.nt.dao.IPersonDetailsDAO;
import com.nt.dao.PersonDetailsDAOImpl;
import com.nt.utility.HibernateUtil;

public class OneToManyUniTest {

    public static void main(String[] args) {
            IPersonDetailsDAO dao=new PersonDetailsDAOImpl();
            //dao.saveDataUsingParent();

            //dao.loadDataUsingParent();

            //dao.addChildToExistingParent();

            // dao.deleteAllChildsOfAParent();

            // dao.deleteOneChildFromCollectionOfChildsOfAParent();

             dao.deleteParentAndItsChilds();

             //close SessionFactory
             HibernateUtil.closeSessionFactory();

    }

}
```

```
Priority  order of choosing annotations
-------------------------------------------
(a) JPA annotations
(b) HB specific annotations
(c) Third party annotations
```

One To many Bi-Driectional Association

Annotation required:

- @OneToMany, @ManyToOne — To build the association
- @JoinColum — To specify FK column

note:: In bi-directional Association we need not to write @JoinColumn annotation both sides ..we just need to write one side and specify that property name using "mappedBy" attribute.

=> This "mappedBy" must be specfied at many side (parent side) in One To Many bi-directional Association.

=> This "mappedBy" must be specfied at any side in Many To Many bi-directional Association.

=>In any association mapping , the "fetch" supports only FetchType.LAZY, FetchType.EAGER from JPA side .. But hibernate is having addtional lazy loading options ..and they can applied using HB specific annotations

=> In One to many and In many to one we can use @LazyCollection to specify TRUE ,FALSE ,EXTRA modes of lazy loading

=> In many to one and In one to one we can use @LazyToOne to specify PROXY,FALSE,NOPROXY modes of lazy loading

```java
//Department.java
package com.nt.entity;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="ANNO_HB_DEPARTMENT")
public class Department {
	@Id
	@GeneratedValue
	private Integer deptNo;
	@Column(length = 20)
	private String deptName;
	@Column(length = 20)
	private String deptHead;
	@OneToMany(targetEntity = Employee.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY,
		mappedBy = "dept",orphanRemoval = true)
	@LazyCollection(LazyCollectionOption.EXTRA)
	//@JoinColumn(name="DEPARTMENT_NO",referencedColumnName ="DEPTNO")
	private Set<Employee> employees; //for one to many association

	public Department() {
		System.out.println("Department:: 0-param constructor"+this.getClass());
	}

	@Override
	public String toString() {
		return "Department [deptNo=" + deptNo + ", deptName=" + deptName + ", deptHead=" + deptHead + "]";
	}


}
```

```java
//Employee.java
package com.nt.entity;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;
import org.hibernate.annotations.LazyToOne;
import org.hibernate.annotations.LazyToOneOption;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_EMPLOYEE")
public class Employee {
	@Id
	@SequenceGenerator(name="gen1",sequenceName = "EMP_SEQ",initialValue = 1000,allocationSize = 1)
	@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
	private Integer empNo;
	@Column(length = 20)
	private String empName;
	private Float empSalary;
	@ManyToOne(targetEntity = Department.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
	@JoinColumn(name="DEPARTMENT_NO",referencedColumnName ="DEPTNO")
	@LazyToOne(LazyToOneOption.FALSE)
	private Department dept; //for many to one association

	public Employee() {
	 System.out.println("Employee:: 0-param constructor"+this.getClass());
	}

	@Override
	public String toString() {
		return "Employee [empNo=" + empNo + ", empName=" + empName + ", empSalary=" + empSalary + "]";
	}
}//class
```

Annotation driven many to many Association

required annotations are

- @ManyToMany :: To build the assocation
- @JoinTable :: To specify third db table
- @JoinColum :: To specify ownside and non-owning side FK columns

```java
//Parent class
package com.nt.entity;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="ANNO_HB_DOCTOR")
public class Doctor {
	@Id
	@GeneratedValue
	private Integer docId;
	@Column(length = 20)
	private String docName;
	@Column(length = 20)
	private String hospital;
	@ManyToMany(targetEntity = Patient.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
	@JoinTable(name = "ANNO_HB_DOCTORS_PATIENTS",
		joinColumns =@JoinColumn(name="DOCTOR_ID",referencedColumnName = "DOCID"),
		inverseJoinColumns =@JoinColumn(name="PATIENT_ID",referencedColumnName = "PATID") )
	@LazyCollection(LazyCollectionOption.EXTRA)
	private Set<Patient> patients; //To hold bunch of patients
	public Doctor() {
		System.out.println("Doctor:: 0-param constructor");
	}
	@Override
	public String toString() {
		return "Doctor [docId=" + docId + ", docName=" + docName + ", hospital=" + hospital + "]";
	}


}
```

```java
//Child class

package com.nt.entity;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="ANNO_HB_PATIENT")
public class Patient {
	@SequenceGenerator(name="gen1",sequenceName = "PAT_SEQ",initialValue = 1000,allocationSize = 1)
	@GeneratedValue(generator = "gen1",strategy =GenerationType.SEQUENCE )
	@Id
	private Integer patId;
	@Column(length = 20)
	private String patName;
	@Column(length = 20)
	private String problem;
	@ManyToMany(targetEntity = Doctor.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY,mappedBy = "patients")
	@LazyCollection(LazyCollectionOption.EXTRA)
	private Set<Doctor> doctors; //To hold bunch of doctors

	public Patient() {
		System.out.println("Patient:: 0-param constructor");
	}

	@Override
	public String toString() {
		return "Patient [patId=" + patId + ", patName=" + patName + ", problem=" + problem + "]";
	}

}//class
```

Annotation driven One to One Association
==========
One to One Association can be implemented in two ways
a) One to One PK (Parent and child objs will use same id value)
b) One To One FK (Parent and child objs will use different id values and they are linked using FK column)

a) One to One PK

=> The required annotation are .
a) @OneToOne :: To build the association
b) @PrimaryKeyJoinColumn :: To cfg PK column as the FK column
c) @GenericGenerator :: To cfg foreign generator to get and use the associated obj's id value as the current obj's id value.
HB annotations we use

JPA Annotations

=>Here we do not use FK column but foreign generator to make child obj to get the id value from the associated parent obj

note:: If @OneToOne is used with @PrimaryKeyJoinColumn then it is called One to One PK Association
If @OneToOne is used with @JoinColumn annotation then it is called One to One FK association.

Parent class

//Student.java (parent class)
package com.nt.entity;

import java.io.Serializable;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_STUDENT")
public class Student implements Serializable {
@Id
@GeneratedValue
private Integer sno;
@Column(length = 20)
private String sname;
@Column(length = 20)
private String sadd;
@OneToOne(targetEntity = LibraryMembership.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
@PrimaryKeyJoinColumn(name = "lid",referencedColumnName = "sno")
private LibraryMembership libraryDetails;

public Student() {
System.out.println("Student:: 0-param constructor");
}

@Override
public String toString() {
return "Student [sno=" + sno + ", sname=" + sname + ", sadd=" + sadd + "]";
}

}

DAO Interface

package com.nt.dao;

public interface OneToOnePKDAO {
public void saveDataUsingParent();
public void saveDataUsingChild();
public void loadDataUsingParent();
}

DAO Impl class

package com.nt.dao;

import java.time.LocalDate;
import java.util.List;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.LibraryMembership;
import com.nt.entity.Student;
import com.nt.utility.HibernateUtil;

public class OneToOnePKDAOImpl implements OneToOnePKDAO {

@Override
public void saveDataUsingParent() {
Session ses=HibernateUtil.getSession();
Transaction tx=null;
try{ses){
//begin tx
tx=ses.beginTransaction();
//prepare objs
Student stud=new Student();
stud.setSname("raja"); stud.setSadd("hyd");
LibraryMembership lib=new LibraryMembership();
lib.setType("gold"); lib.setDoj(LocalDate.now());
//set parent to child
lib.setStudentDetails(stud);
stud.setLibraryDetails(lib);
//save objs
ses.save(stud);
tx.commit();
System.out.println("Objects are saved");

}//try
catch(HibernateException he) {
he.printStackTrace();
if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
tx.rollback();
System.out.println("Problem in objs saving");
}
}//catch
}//method

@Override
public void saveDataUsingChild() {
Session ses=HibernateUtil.getSession();
Transaction tx=null;
try{ses){
//begin tx
tx=ses.beginTransaction();
//prepare objs
Student stud=new Student();
stud.setSname("suresh"); stud.setSadd("vizag");
LibraryMembership lib=new LibraryMembership();
lib.setType("silver"); lib.setDoj(LocalDate.now());
//set parent to child
lib.setStudentDetails(stud);
stud.setLibraryDetails(lib);
//save objs
ses.save(lib);
tx.commit();
System.out.println("Objects are saved");
}//try
catch(HibernateException he) {
he.printStackTrace();
if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
tx.rollback();
System.out.println("Problem in objs saving");
}
}//catch
}//method

@Override
public void loadDataUsingParent() {
Session ses=HibernateUtil.getSession();
Transaction tx=null;
try{ses){
Query query=ses.createQuery("from Student");
List<Student> list=query.getResultList();
list.forEach(stud->{
System.out.println("Parent::"+stud);
LibraryMembership lib=stud.getLibraryDetails();
System.out.println("Child::"+lib);
});
}//try
catch(HibernateException he) {
he.printStackTrace();
}
catch(Exception e) {
e.printStackTrace();
}
}//method

}//class

//LibraryMembership.java
package com.nt.entity;

import java.io.Serializable;
import java.time.LocalDate;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_LIBRARY_MEMBERSHIP")
public class LibraryMembership implements Serializable {

@GenericGenerator(name = "gen1",strategy = "foreign",
parameters = @Parameter(name = "property", value = "studentDetails"))
@GeneratedValue(generator = "gen1")
@Id
private Integer lid;
@Column(length=20)
private String type;
private LocalDate doj;
@OneToOne(targetEntity = Student.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY,mappedby = "libraryDetails")
private Student studentDetails;

public LibraryMembership() {
System.out.println("LibraryMembership:: 0-param constructor");
}

@Override
public String toString() {
return "LibraryMembership [lid=" + lid + ", type=" + type + ", doj=" + doj + "]";
}

}

Client app
==========
package com.nt.test;

import com.nt.dao.OneToOnePKDAO;
import com.nt.dao.OneToOnePKDAOImpl;
import com.nt.utility.HibernateUtil;

public class OneToOnePKTest {

public static void main(String[] args) {
//create DAO class obj
OneToOnePKDAO dao=new OneToOnePKDAOImpl();
// dao.saveDataUsingParent();
//dao.saveDataUsingChild();
dao.loadDataUsingParent();
//close Session Factory
HibernateUtil.closeSessionFactory();
}

}

=>Child table gets its pk column value from associated Parent table's pk col value.

==========
One To One FK association
==========
=>Here parent and child objs want to have seperate id values but they will be mapped using FK column
Required annotations are

@OneToOne :: To build the association
@JoinColumn :: To specify FK column linked with PK column

usecase :: Person and Passport (person id is different from passport no ..but passport will be linked with Person using FK column)

Person.java (parent class)

package com.nt.entity;

import java.io.Serializable;

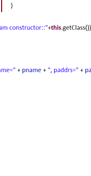
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

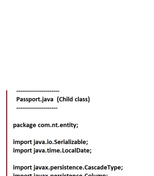
import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_PERSON")
public class Person implements Serializable{
@GeneratedValue
@Id
private Integer pid;
@Column(length = 20)
private String pname;
@Column(length = 20)
private String paddrs;

public Person() {
System.out.println("Person:: 0-param constructor::"+this.getClass());
}

@Override
public String toString() {
return "Person [pid=" + pid + ", pname=" + pname + ", paddrs=" + paddrs + "]";
}
}





//DAO Interface

package com.nt.dao;

public interface OneToOneFKDAO {
public void saveDataUsingChild();
public void loadDataUsingChild();
}

/DAOImpl class
package com.nt.dao;

import java.time.LocalDate;
import java.util.List;

lmport javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.Passport;
import com.nt.entity.Person;
import com.nt.utility.HibernateUtil;

public class OneToOneFKDAOImpl implements OneToOneFKDAO {
@Override
public void saveDataUsingChild() {
Session ses=HibernateUtil.getSession();
Transaction tx=null;
try{ses){
//begin tx
tx=ses.beginTransaction();
//prepare objs
Person per=new Person();
per.setPname("raja"); per.setPaddrs("hyd");
Passport pspt=new Passport();
pspt.setCountry("India"); pspt.setExpiryDate(LocalDate.of(2030,10, 2));
//set parent to child
pspt.setPersonDetails(per);
//save objs
ses.save(per);
tx.commit();
System.out.println("Objects are saved");
}//try
catch(HibernateException he) {
he.printStackTrace();
if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
tx.rollback();
System.out.println("Problem in objs saving");
}
}//catch
}//method

@Override
public void loadDataUsingChild() {
Session ses=HibernateUtil.getSession();
try{ses){
Query query=ses.createQuery("from Passport");
List<Passport> list=query.getResultList();
list.forEach(pspt->{
System.out.println("child::"+pspt);
Person perpspt.getPersonDetails();
System.out.println("parent::"+per);
});
}//try
catch(HibernateException he) {
he.printStackTrace();
}
catch(Exception e) {
e.printStackTrace();
}

}//method

}//class

Passport.java (Child class)

package com.nt.entity;

import java.io.Serializable;
import java.time.LocalDate;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_PASSPORT")
public class Passport implements Serializable {
@Id
@SequenceGenerator(name="gen1",sequenceName = "PSPT_SEQ",initialValue = 1000,allocationSize = 1)
@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
private Integer passportNo;
@Column(length = 20)
private String country;
private LocalDate expiryDate;
@OneToOne(targetEntity = Person.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
@JoinColumn(name = "PERSON_ID",referencedColumnName = "PID")
private Person personDetails; //for OneToOne Association
public Passport() {
System.out.println("Passport:: 0-param constructor::"+this.getClass());
}
@Override
public String toString() {
return "Passport [passportNo=" + passportNo + ", country=" + country + ", exlpiryDate=" + expiryDate + "]";
}

}

Client App
package com.nt.test;

import com.nt.dao.OneToOneFKDAO;
import com.nt.dao.OneToOneFKDAOImpl;
import com.nt.utility.HibernateUtil;

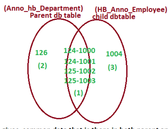
public class OneToOneFKTest {

public static void main(String[] args) {
//create DAO class obj
OneToOneFKDAO dao=new OneToOneFKDAOImpl();
//dao.saveDataUsingChild();
dao.loadDataUsingChild();
//close Session Factory
HibernateUtil.closeSessionFactory();
}

}

HQL Joins
=========

=>HQL JOINS are implicit conditions that are given to common and uncommon data of DB table that are in relationship
=> HQL Joins are similar to SQL Joins
=> 4 Types of HQL Joins
a) Inner Join
b) Left Join / Left outer Join
c) Right Join /Right Outer Join
d) Full Join



a) inner join :: gives common data that is there in both parent and child db tables {1}
b) left join :: gives common data + un common data of left side db table(parent) {1}+{2}
c) rigth join :: gives common data + un common data of right side db table(child) {1}+{3}
d) full join :: gives common data + un common data of both db tables(parent-child) {1}+{2}+{3}

Department.java (parent class)

```
//Department.java (parent class)
package com.nt.entity;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="ANNO_HB_DEPARTMENT")
public class Department {
        @Id
        @GeneratedValue
        private Integer deptNo;
        @Column(length = 20)
        private String deptName;
        @Column(length = 20)
        private String deptHead;
        @OneToMany(targetEntity = Employee.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY,
                        mappedBy = "dept",orphanRemoval = true)
        @LazyCollection(LazyCollectionOption.EXTRA)
        //@JoinColumn(name="DEPARTMENT_NO",referencedColumnName ="DEPTNO")
        private Set<Employee> employees; //for one to many association

        public Department() {
                System.out.println("Department:: 0-param constructor"+this.getClass());
        }

        @Override
        public String toString() {
                return "Department [deptNo=" + deptNo + ", deptName=" + deptName + ", deptHead=" + deptHead + "]";
        }



}
```

Employee.java (child class)

```
//Employee.java
package com.nt.entity;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import org.hibernate.annotations.LazyCollection;
import org.hibernate.annotations.LazyCollectionOption;
import org.hibernate.annotations.LazyToOne;
import org.hibernate.annotations.LazyToOneOption;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_EMPLOYEE")
public class Employee {
        @Id
        @SequenceGenerator(name="gen1",sequenceName = "EMP_SEQ",initialValue = 1000,allocationSize = 1)
        @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
        private Integer empNo;
        @Column(length = 20)
        private String empName;
        private Float empSalary;
        @ManyToOne(targetEntity = Department.class,cascade = CascadeType.ALL,fetch = FetchType.LAZY)
        @JoinColumn(name="DEPARTMENT_NO",referencedColumnName ="DEPTNO")
        @LazyToOne(LazyToOneOption.FALSE)
        private Department dept; //for many to one association

        public Employee() {
         System.out.println("Employee:: 0-param constructor"+this.getClass());
        }

        @Override
        public String toString() {
                return "Employee [empNo=" + empNo + ", empName=" + empName + ", empSalary=" + empSalary + "]";
        }
}//class
```

HQL Join Query from parent to child

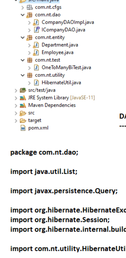
```
select d.deptno,d.deptName,d.deptHead,e.empNo,e.ename,e.empSalary from Department d inner join d.employees e
                properties of parent and child db tables
```

parent class (left side table)
child class (right side table)
property in parent class refering child class objs

HQL Join Query from child to parent

```
select e.empNo,e.ename,e.empSalary,d.deptNo,d.deptName,d.deptHead from Employee e inner join e.dept d
                properties of parent and child tables
```

(child class) (left side table)
(property in child class refering to parent class obj) (right side table)



DAO Interface

```
package com.nt.dao;

public interface ICompanyDAO {
  public  void  loadDataUsingJoinsParentToChild();
  public  void  loadDataUsingJoinsChildToParent();

}
```

DAO Impl class

```
package com.nt.dao;

import java.util.List;

import javax.persistence.Query;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.internal.build.AllowSysOut;

import com.nt.utility.HibernateUtil;

public class CompanyDAOImpl implements ICompanyDAO {

        @Override
        public void loadDataUsingJoinsParentToChild() {
                Session sess=HibernateUtil.getSession();
                try{ses){
                        //Query query=ses.createQuery("select d.deptNo,d.deptName,d.deptHead,e.empNo,e.empName,e.empSalary from Department d inner join d.employees e");
                        //Query query=ses.createQuery("select d.deptNo,d.deptName,d.deptHead,e.empNo,e.empName,e.empSalary from Department d left join d.employees e");
                        //Query query=ses.createQuery("select d.deptNo,d.deptName,d.deptHead,e.empNo,e.empName,e.empSalary from Department d right join d.employees e");
                        Query query=ses.createQuery("select d.deptNo,d.deptName,d.deptHead,e.empNo,e.empName,e.empSalary from Department d full join d.employees e");
                        List<Object[]> list=query.getResultList();
                        list.forEach(row->{
                                for(Object val:row) {
                                        System.out.print(val+"  ");
                                }//for
                                System.out.println();
                        });//forEach
                }//try
                catch(HibernateException he) {
                        he.printStackTrace();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }
        }//method

        @Override
        public void loadDataUsingJoinsChildToParent() {
                Session sess=HibernateUtil.getSession();
                try{ses){
                        //Query query=ses.createQuery("select e.empNo,e.empName,e.empSalary,d.deptNo,d.deptName,d.deptHead from Employee e inner join e.dept d");
                        //Query query=ses.createQuery("select e.empNo,e.empName,e.empSalary,d.deptNo,d.deptName,d.deptHead from Employee e left join e.dept d");
                        //Query query=ses.createQuery("select e.empNo,e.empName,e.empSalary,d.deptNo,d.deptName,d.deptHead from Employee e right join e.dept d");
                        Query query=ses.createQuery("select e.empNo,e.empName,e.empSalary,d.deptNo,d.deptName,d.deptHead from Employee e full  join e.dept d");
                        List<Object[]> list=query.getResultList();
                        list.forEach(row->{
                                for(Object val:row) {
                                        System.out.print(val+"  ");
                                }//for
                                System.out.println();
                        });//forEach
                }//try
                catch(HibernateException he) {
                        he.printStackTrace();
                }
                catch(Exception e) {
                        e.printStackTrace();
                }

        }
}//class
```

Client App
==========

```
package com.nt.test;

import com.nt.dao.CompanyDAOImpl;
import com.nt.dao.ICompanyDAO;
import com.nt.utility.HibernateUtil;

public class OneToManyBiTest {

        public static void main(String[] args) {
                ICompanyDAO dao=new CompanyDAOImpl();
                //dao.loadDataUsingJoinsParentToChild();
                dao.loadDataUsingJoinsChildToParent();



                //close SessionFactory
                HibernateUtil.closeSessionFactory();
        }//method
}//class
```

1 + N select Problem

(parent db table)

ANNO_HB_DEPARTMENT

pk	DEPTNO	DEPTHEAD	DEPTNAME
1	124 JAMES	IT	
2	125 JHON	ACCOUNTS	

(child db table)

HB_ANNO_EMPLOYEE

(pk)	EMPNO	EMPNAME	EMPSALARY	DEPARTMENT_NO (FK)
1	1000	rameh	898989	124
2	1001	rajesh	898989	124
3	1003	naresh	828282	125
4	1002	suresh	818981	125

```
@Override
public void loadDataUsingParentHQL() {
    Session ses=HibernateUtil.getSession();
    try(ses){
        Query query=ses.createQuery("from Department");
        List<Department> list=query.getResultList();
        list.forEach(dept->{
            System.out.println("parent ::"+dept);
            Set<Employee> childs=dept.getEmployees();
            childs.forEach(emp->{
                System.out.println("child::"+emp);
            });//forEach
        }); //forEach
    }//try
    catch(HibernateException he) {
        he.printStackTrace();
    }
    catch(Exception e) {
        e.printStackTrace();
    }

}//method
```

=>With respect to the above db table data the above code generates 1+n select SQL Queries where 1 query will be generated to get all the parents and n SQL Queries will be generated to get childs of all the n parents. w.r.t the above db tables data one query will be generated to get all the parents (2) and n queries(2) will be generated to get the childs of 2 parents .. So total 1+ 2 queries will be generated .. more queries generation more hits to db s/w which leads performence issue. if parent db table is having 10 recods then with childs records then total 1+10 Queries will be generated which is called 1+n select problem.

select * from Department (1 query to get all the N parents)

124 .. ..
125 .. ..

select * from Employee where deptno=124
....
....
select * from Employee where deptno=125
...
...

(N Queries to get childs of N parents)

Solutions for 1+n select problem
==========
a) Working with HQL Joins
b) Specifying Fetch type as the "INNER in JPA QBC Logic

Solution1:: a) Working with HQL Joins

```
@Override
public void loadDataUsingJoinsParentToChild() {
    Session ses=HibernateUtil.getSession();
    try(ses){
        Query query=ses.createQuery("select d.deptNo,d.deptName,d.deptHead,
            e.empNo,e.empName,e.empSalary from Department d inner join d.employees e");
        List<Object[]> list=query.getResultList();
        list.forEach(row->{
            for(Object val:row) {
                System.out.print(val+"  ");
            }//for
            System.out.println();
        });//forEach
    }//try
    catch(HibernateException he) {
        he.printStackTrace();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
}//method
```

output:

```
Hibernate:
    select
        department0_.deptNo as col_0_0_,
        department0_.deptName as col_1_0_,
        department0_.deptHead as col_2_0_,
        employees1_.empNo as col_3_0_,
        employees1_.empName as col_4_0_,
        employees1_.empSalary as col_5_0_
    from
        ANNO_HB_DEPARTMENT department0_
    inner join
        HB_ANNO_EMPLOYEE employees1_
            on department0_.deptNo=employees1_.DEPARTMENT_NO
124  IT  JAMES  1000  ramesh  898989.0
124  IT  JAMES  1001  rajesh  898989.0
125  ACCOUNTS  JHON  1003  naresh  828282.0
125  ACCOUNTS  JHON  1002  suresh  818981.0
```

Single Query only giving all the records.

Solution2: Specifying Fetch type as the "INNER" in JPA QBC Logic

```
@Override
public void loadDataUsingJPQBC() {
    Session ses=HibernateUtil.getSession();
    try(ses){
    //create CriteriaBuilder obj
    CriteriaBuilder ctBuilder=ses.getCriteriaBuilder();
    //create CriteriaQuery object
  CriteriaQuery<Department> ctQuery=ctBuilder.createQuery(Department.class);
  // create Root object specifying the from operation
  Root<Department> root=ctQuery.from(Department.class);  //select * from Product
  root.fetch("employees",JoinType.INNER);
  //  create Query object having CriteriaQuery object
  Query query=ses.createQuery(ctQuery);
  //execute the QBC logic
  List<Department> list=query.getResultList();
  //process the result
   list.forEach(dept->{
        System.out.println("parent::"+dept);
        //get childs
        Set<Employee> childs=dept.getEmployees();
        childs.forEach(emp->{
            System.out.println("Child::"+emp);
        });
   });
    }
    catch(HibernateException he) {
        he.printStackTrace();
    }

}
```

```
Hibernate:
    select
        department0_.deptNo as deptno1_0_0_,
        employees1_.empNo as empno1_1_1_,
        department0_.deptHead as depthead2_0_0_,
        department0_.deptName as deptname3_0_0_,
        employees1_.DEPARTMENT_NO as department_no4_1_1_,
        employees1_.empName as empname2_1_1_,
        employees1_.empSalary as empsalary3_1_1_,
        employees1_.DEPARTMENT_NO as department_no4_1_0__,
        employees1_.empNo as empno1_1_0__
    from
        ANNO_HB_DEPARTMENT department0_
    inner join
        HB_ANNO_EMPLOYEE employees1_
            on department0_.deptNo=employees1_.DEPARTMENT_NO
Department:: 0-param constructorclass com.nt.entity.Department
Employee:: 0-param constructorclass com.nt.entity.Employee
Employee:: 0-param constructorclass com.nt.entity.Employee
Department:: 0-param constructorclass com.nt.entity.Department
Employee:: 0-param constructorclass com.nt.entity.Employee
Employee:: 0-param constructorclass com.nt.entity.Employee
parent::Department [deptNo=124, deptName=IT, deptHead=JAMES]
Child::Employee [empNo=1001, empName=rajesh, empSalary=898989.0]
Child::Employee [empNo=1000, empName=ramesh, empSalary=898989.0]
parent::Department [deptNo=124, deptName=IT, deptHead=JAMES]
Child::Employee [empNo=1001, empName=rajesh, empSalary=898989.0]
Child::Employee [empNo=1000, empName=ramesh, empSalary=898989.0]
parent::Department [deptNo=125, deptName=ACCOUNTS, deptHead=JHON]
Child::Employee [empNo=1003, empName=naresh, empSalary=828282.0]
Child::Employee [empNo=1002, empName=suresh, empSalary=818981.0]
parent::Department [deptNo=125, deptName=ACCOUNTS, deptHead=JHON]
Child::Employee [empNo=1003, empName=naresh, empSalary=828282.0]
Child::Employee [empNo=1002, empName=suresh, empSalary=818981.0]
```

Working with LOBs (Large Objects --Files)

=> we have two types of LOBS

a) BLOB b) CLOB

=>BLOB:: Bindary LOB (contains binary data .. 0 ,1,0,1 ...)
eg:: images, audio files , video files, avi files and etc..
=>CLOB:: Character LOB (contains text data .. a-z , 0-9)
eg:: text files, rich text files and etc..

=>All most all major DB s/ws gives support for LOBs

=> while developing standalone apps dealing with files insertion and retrieving we need the support of LOBs
eg:: Matrimony Apps, job portal apps, social networking apps, profile mgmt apps and etc..

=> While developing entity classes having proeprties to map with BLOB and CLOB type cols we need to take properties of type byte[] (for BLOB col) or char[] (for CLOB col)

Working with LOBs [Large Objects --Files]

=> we have two types of LOBs

a) BLOB b) CLOB

=>BLOB:: Binary LOB (contains binary data .. 0 ,1,0,1 ...)
eg: Images, audio files , video files, exl files and pdf.
=>CLOB:: Character LOB (contains text data .. a e z , 0-9)
eg: text files, rich text files and etc...

=>All most all major DB s/ws gives support for LOBs

=> while developing standalone apps dealing with files insertion and retrieving we need the support of LOBs
eg: Matrimony Apps, Job portal apps, social networking apps, profile mgmt apps and etc..

=> While developing entity classes having properties to map with BLOB and CLOB type cols we need to take properties of type byte[] (for BLOB col) or char[] (for CLOB col) having annotation @Lob

//Employee.java
package com.nt.entity;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;

import lombok.Getter;
import lombok.Setter;

@Setter
@Getter
@Entity
@Table(name="HB_ANNO_EMPLOYEE_LOBs")
public class Employee {
@Id
@GeneratedValue
private Integer empNo;
@Column(length = 20)
private String empName;
private Float empSalary;
@Lob
private byte[] empPhoto;
@Lob
private char[] empResume;

public Employee() {
System.out.println("Employee:: 0-param constructor");
}

}//class

DAO Interface

package com.nt.dao;

public interface ICompanyDAO {
public void saveData();
public void loadData();

}

DAO Impl class

package com.nt.dao;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.Reader;
import java.io.Writer;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.entity.Employee;
import com.nt.utility.HibernateUtil;

public class CompanyDAOImpl implements ICompanyDAO {

@Override
public void saveData() {
Session ses=HibernateUtil.getSession();
Transaction tx=null;
try{ss{
//read Image file content to byte[]
File file1=new File("G:\\Workspaces\\Hibernate\\NTHB412\\HBProj61Anno-WorkingWithLOBS\\src\\main\\java\\LOBS\\deep.jpeg");
InputStream in=new FileInputStream(file1);
byte[] imgContent=new byte[(int) file1.length()];
is.read(imgContent);

//read text file content to char[]
File file2=new File("G:\\Workspaces\\Hibernate\\NTHB412\\HBProj61Anno-WorkingWithLOBS\\src\\main\\java\\LOBS\\resume.txt");
Reader reader=new FileReader(file2);
char resumeContent[]=new char[(int) file2.length()];
reader.read(resumeContent);
//prepare Entity class obj
Employee emp=new Employee();
emp.setEmpName("raja"); emp.setEmpSalary(9000.0f);
emp.setEmpPhoto(imgContent); emp.setEmpResume(resumeContent);

tx=ses.beginTransaction();
int idVal=(int) ses.save(emp);
tx.commit();
System.out.println("Object is saved with the idValue:"+idVal);
}
catch(HibernateException he) {
he.printStackTrace();
if(tx!=null && tx.getStatus()!=null && tx.getRollbackOnly()) {
tx.rollback();
System.out.println("Object is not saved");
}
}
catch(Exception e) {
e.printStackTrace();
}

}

@Override
public void loadData() {
Session ses=HibernateUtil.getSession();
try{ss{
Employee e=ses.get(Employee.class, 161);
if(e==null) {
System.out.println("Employee not found");
return ;
}
else {
System.out.println(e.getEmpNo()+" "+e.getEmpName()+" "+e.getEmpSalary());
byte[] photoContent=e.getEmpPhoto();
char[] resumeContent=e.getEmpResume();
//create output stream pointing to dest file
OutputStream os=new FileOutputStream("new_pkt.jpg");
os.write(photoContent);
os.flush();
os.close();
//create Writer stream pointing to dest file
Writer writer=new FileWriter("new_resume.txt");
writer.write(resumeContent);
writer.flush();
writer.close();
System.out.println("LOBs are retrieved successfully");
}
}//try
catch(HibernateException he) {
he.printStackTrace();
}
catch(Exception e) {
e.printStackTrace();
}

}//method
}//class

Client App

package com.nt.test;

import com.nt.dao.CompanyDAOImpl;
import com.nt.dao.CompanyDAO;
import com.nt.utility.HibernateUtil;

public class LobsTest {

public static void main(String[] args) {
ICompanyDAO dao=new CompanyDAOImpl();
//dao.saveData();
dao.loadData();

//close SessionFactory
HibernateUtil.closeSessionFactory();
}//method
}//class

L2 Cache in hibernate

=>Hibernate supports two levels of caching to hold the objects of entity classes representing the records across the multiple name requests b/w hibernate App (java app with hibernate code) and Db s/w.

1) Level1 /L1 cache/ Session cache /First Level cache
[One per each session object]

2) Level2/L2 cache/ SessionFactory cache/Global cache /second level cache
[one per each SessionFactory obj i.e one per each DB s/w]

Both these caches holds records in the form of Objects of entity classes and those objects will be reused across the multiple same requests.

Hibernate App (Java App with Hibernate Persistence logic)

a1 to a7 :: 1st request
b1 to b1 :: 2nd request
c1 to c5 :: 3rd request

all these 3 requests are same requests asking for same data.

d1 to d7 :: 4th request

=> Level1 /first level cache is Session obj level built-in cache and it can not be disabled .. becoz it is part of hibernate's echo system while performing all CURD operations ... but we can control cache using ses.close() ,ses.evict() and ses.clear() methods.

=>Level2/ second level cache is configurable cache i.e it is not built-in cache of hibernate app.. It will come only when it is applied/configured
=> Level2 cache can be enabled or disabled as per our intrests.. becoz it is part of hibernate's hibenate's echo system of performing curd operations.
=> There are multiple third party vendors offering second level cache for hibernate

a) EH cache (Easy Hibernate)
b) swarm cache
c) OS cache
d) Boss Tree cache
e) Tangosol cache

Free caching softwares

commercial

=> While configuring second level we must specify caching strategy
read-only: caching will work for read only operation.
nonstrict-read-write: caching will work for read and write but one at a time.
read-write: caching will work for read and write, can be used simultaneously.
transactional: caching will work for transaction.

In annotation driven cfg use
@Cacheable
@Cache(usage=CacheConcurrencyStrategy.READ_ONLY)
READ_ONLY
NONSTRICT_READ_WRITE
READ_WRITE
TRANSACTIONAL

on top of entity class to specify the caching strategy

In xml driven cfgs
<cache usage="read-only"/> (place before <id> tag)
"read-write"
"nonstrict_read_write"
"transactional"

==========
Procedure to configure EHCache as second level cache in Hibernate App
==========

step1) keep any hibernate app ready

step2) add the following jar files /dependencies related to EH cache to to pom.xml

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-ehcache</artifactId>
<version>5.6.8.Final</version>
</dependency>

step3) Specify caching strategy in mapping file
In Product.hbm.xml

<cache usage="read-only"/>
<id name="pid" column="PID" >
<generator class="increment"/>
</id>

step4) enable second level cache by addling the following entries in hibernate.cfg.xml file
In hibernate.cfg.xml file

<property name="cache.use_second_level_cache">true</property>
<property name="cache.region_factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

step5) write following code Client App

package com.nt.test;

import org.hibernate.Cache;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class L2CacheTest {

public static void main(String[] args) {
//Configuration obj
Configuration cfg=new Configuration();
cfg.configure("com/nt/cfg/hibernate.cfg.xml");
//build SessionFactory obj
SessionFactory factory=cfg.buildSessionFactory();
//build Session
Session ses=factory.openSession();
try{ss{ (a)
Product prod=ses.get(Product.class, 25); //gets from DB and keeps L2, L1 cache
System.out.println(prod);
System.out.println("----------");
Product prod1=ses.get(Product.class, 25); //gets from L1 cache
System.out.println(prod1);
System.out.println("----------");
ses.evict(prod1); //removes prod1 from L1 cache
Product prod2=ses.get(Product.class, 25); //gets from L2 cache and keeps L1 cache
System.out.println(prod2);
System.out.println("----------");
ses.clear(); //removes all from L1 cache
Cache cache=factory.getCache();
cache.evictAll(); //emptying L2 cache (d)
Product prod3=ses.get(Product.class, 25); //gets from DB and keeps in L2,L1 caches
System.out.println(prod3);
}//try
catch(HibernateException he) {
he.printStackTrace();
}

}//main
}//class

output
Hibernate:
select
product0_.PID as pid1_0_0_,
product0_.PNAME as pname2_0_0_,
product0_.PRICE as price3_0_0_,
product0_.QTY as qty4_0_0_
from
PRODUCT product0_
where
product0_.PID=?
Product:: 0-param constructorclass com.nt.entity.Product
Product[pid=25, pname=keyboard, price=3156.0, qty=666.0]

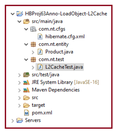
Product[pid=25, pname=keyboard, price=3156.0, qty=666.0] (b)

Product:: 0-param constructorclass com.nt.entity.Product
Product[pid=25, pname=keyboard, price=3156.0, qty=666.0] (c)

Hibernate:
select
product0_.PID as pid1_0_0_,
product0_.PNAME as pname2_0_0_,
product0_.PRICE as price3_0_0_,
product0_.QTY as qty4_0_0_
from
PRODUCT product0_
where
product0_.PID=?
Product:: 0-param constructorclass com.nt.entity.Product
Product[pid=25, pname=keyboard, price=3156.0, qty=666.0] (d)

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration SYSTEM
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- Connection properties -->
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">manager</property>

    <!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

    <property name="cache.use_second_level_cache">true</property>
    <property name="cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

    <!-- mapping file name -->
    <mapping class="com.nt.entity.Product"/>
  </session-factory>

</hibernate-configuration>
```



Product.java (Entity class)

```
package com.nt.entity;

import javax.persistence.Cacheable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.annotations.Cache;
import org.hibernate.annotations.CacheConcurrencyStrategy;

import lombok.Data;

@Data
@Entity
@Table(name="PRODUCT")
@Cacheable
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
public class Product {
    @GeneratedValue
    @Id
  private Integer pid;
  @Column(length = 20)
  private String pname;
  private Float price;
  private Float qty;

  public Product() {
      System.out.println("Product:: 0-param constructor"+this.getClass());
  }

}
```

L2CacheTest.java

```
package com.nt.test;

import org.hibernate.Cache;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

import com.nt.entity.Product;

public class L2CacheTest {

    public static void main(String[] args) {
        //Configuration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        try{factory;ses){
            Product prod=ses.get(Product.class, 25);  //gets from DB and keeps L2, L1 cache
            System.out.println(prod);
            System.out.println("-----------------");
            Product prod1=ses.get(Product.class, 25);  //gets from  L1 cache
            System.out.println(prod1);
            System.out.println("_____________________");
            ses.evict(prod1);  //removes prod1 from L1 cache
            Product prod2=ses.get(Product.class, 25);  //gets from  L2 cache and keeps L1 cache
            System.out.println(prod2);
            System.out.println("_____________________");
            ses.clear(); //removes all from L1 cache
            Cache cache=factory.getCache();
            cache.evictAll(); //emptying L2 cache
            Product prod3=ses.get(Product.class, 25);  //gets from DB and keeps  in L2,L1 caches
            System.out.println(prod3);
        }//try
        catch(HibernateException he) {
            he.printStackTrace();
        }

    }//main
}//class
```

if u r hitting the DB s/w for multiple times for same sets of data using different sesssion objs then prefer enabling L2 cache.

==========
Query cache
==========

=>It is part of L2 cache having ability to store HQL Query related results /objs in the seperate query cache region to use the across the multiple executions of the same HQL SELECT Query.

=> The L1 and L2 caches does not keep HQL Query results .. untill and unlees we enable Query cache.

```
Query query=ses.createQuery("from Product");
        List<Product> list=query.getResultList();
        list.forEach(prod->{
            System.out.println(prod);
        });

        System.out.println("-----------------------");
        List<Product> list1=query.getResultList();
        list1.forEach(prod->{
            System.out.println(prod);
        });
```

Hits the DB s/w by generating SELECT SQL Query

Hits the DB s/w by generating SELECT SQL Query

Solution :: Work with Query Cache

step1) make sure that application is enabled with L2 cache

step2) Write the following eaddtional entry in hibenrate.cfg.xml file to enable query cache

```
<property name="cache.use_query_cache">true</property>
```

step3) Write the following in the client App.

```
package com.nt.test;

import java.util.List;


import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;

import com.nt.entity.Product;

public class QueryCacheTest {

    public static void main(String[] args) {
        //Configuration obj
        Configuration cfg=new Configuration();
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        //build SEssionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();
        //build Session
        Session ses=factory.openSession();
        try{factory;ses){
            Query query=ses.createQuery("from Product");
            query.setCacheable(true);
            query.setCacheRegion("region1");
            List<Product> list=query.getResultList();
            list.forEach(prod->{
                System.out.println(prod);
            });
            System.out.println("-----------------------");
            List<Product> list1=query.getResultList();
            list1.forEach(prod->{
                System.out.println(prod);
            });
        }//try
    catch(HibernateException he) {
        he.printStackTrace();
    }

    }//main
}//class
```

uses the hibernate api query

Hits the DB and gets the records (generates SQL query)

gets the records from query cache region1 (no SQL Query will be generated)

List out different caches the hibenrate supports?

Ans1) 2 caches support
- a) First Level /L1/Level1 cache
- b) Second LEvel/L2 /Level2 cache (Query cache is part of L2 cache)

Ans2) 3 caches support
- a) First Level Cache
- b) Second Level cache
- b)Query cache (part of Second Level cache)

Bootstrapping in hibernate
==========
=> Bootstrapping keeps the neccessary things in ready mode
=> Bootstrapping in hibernate is nothing but keeping HB SessionFactory , HB Session and Configuration objs ready for utilization

HB supports two types of bootstrapping
- a) Legacy boot strapping (from the beging of hibernate to till current version)
- b) Modren boot strapping (from hibernate 4.x onwords with the support of services registries)

a) Legacy boot strapping code

```
//Configurtation obj
Configuration cfg=new Configuration();
cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
//build SEssionFactory obj
SessionFactory factory=cfg.buildSessionFactory();
//build Session
Session ses=factory.openSession();
```

note:: The very big limitation of this code is we can not add custom services in the process of creating SessionFactory object ... For hibernate supports very few DataSources like c3p0, proxool, hikaricp and etc.. for jdbc con pooling.. If we want to use ohter DataSources like Tomcat cp, apache dbcp2 user supplied connection and etc.. for DB connection.. that is not possible.

Conclusion:: In Legacy bootstapping approach , we can use only those services that are offered by Configuration object based on hibernate cfg file.. If u r looking for other services take we need to go for modren boot strapping.

Hibernate Reverse Engineering Using Eclipse
==========
Part 1: https://www.youtube.com/watch?v=U36InkC8X0I
Part 2: https://www.youtube.com/watch?v=Ud5qsNc1-Fg

Modren Bootstrapping (Introduced from HB 4.x taking the support Service Registries)

Maker interface /Tag interface
==========

=>The interface that makes underlying JVM/server/Framework/Container to provide special runtime capabilities to the impl class objs is called market interface /tag interface.

=> the Uderlying JVM/server/Framework/Container does not provide special runtime captaiblities of the beozo methods declarations that are in marker interface.. they will provide special run time capabilities based the marker interface that being implemented.

=> Most of the times marker interface are empty intefaces..
eg:: java.io.Serializable (I) -> To make impl class obj as serilizable obj
eg:: java.lang.Cloneable(I) --> To make impl class obj as cloneable obj
eg: org.hibernate.service.Service(I) --> To make impl class obj as HB service
eg: java.rmi.Remote(I) -->To make impl class obj as Remote object.

Service::

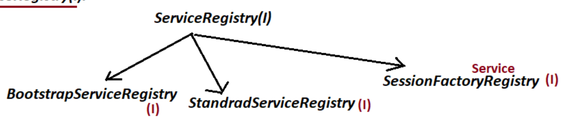
=>All services in hibernate that are bound with SessionFactory are the classes implementing org.hiberante.service.Service(I) marker interface.

=> Service (I) is having multiple sub interfaces and implementing these sub interfaces we have multiple impl classes and these classes represent various services .. all these serivces will be bound with SessionFactory object based on cfgs we do in the HB cfg file..

=>These sub interfaces represent .. different categoes of services in hibernate
=>we can develope custome services to add to SEssionFactory by implementing these sub interfaces..

ServiceRegitries::

=> ServiceRegistry is a registry where services can be registered .. All service registries are the implementation classes of ServiceRegistry(I).



=>Different services hibernate will be registered with different registries and they can be used in the creation of SessionFactory object.. So that all services registered with service registry will be bound with SessionFactory object..

The hierarchy is :: BootstrapServiceRegistry ----> StandardSErviceRegistry -->SessionFactoryServiceRegistry .

=>Service registries are hierarchal i.e if certain service is not available in one service registry then it will be verified in another service registry..

=>The procecess of keeping service with service registry is called service binding
and removing Service from serviceRegistry is called Service Unbiniding
=> The 3 service registries of hibernate are having built-in service binding and also allows custom services binding.

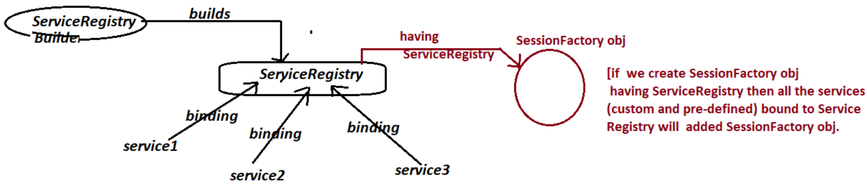
BootStrapServiceRegistry is having following services binding
=> ClassLoaderService , IntegratorService , StrategySelector and etc..

StandardServiceRegistry is having following services binding
=>ConnectionProvider/MultiTenantConnectionProvider, JdbcServices, TransactionFactory ,JtaPlatform and etc..

SessionFactoryServiceRegistry is having the following services binding
=>StatisticsImplementor and etc..

To create ServiceRegistries we take the support of ServiceRegistrybuilder .. we have multiple serviceRegistry builders

BootstrapServiceRegsitryBuilder to build BootstrapServiceRegistry
StandardServiceRegsitryBuilder to build StandardServiceRegistry
SessionFatcoryServiceRegsitryBuilder to build SessionFactoryServiceRegistry



In Legacy style , we use cfg.buidlSessionFactory() method to create SessionFactory obj
In modren style , we use cfg.buidlSessionFactory(registry) method to create SessionFactory obj

---

Hibernate BootStrapping in modrenStyle
==========

In the static block of HibernateUtil.java

```
private static SessionFactory factory;
    static {
        try {
        //Actiate hibernate
        Configuration cfg=new Configuration();
        //declarative approach
        cfg.configure("com/nt/cfgs/hibernate.cfg.xml");
        cfg.addResource("com/nt/entity/Product.hbm.xml");
        //create ServiceRegistryBuilder
        StandardServiceRegistryBuilder builder=new StandardServiceRegistryBuilder();
        //create Servicer registry
        StandardServiceRegistry registry=builder.applySettings(cfg.getProperties()).build();
        //build SessionFactory using registry
         factory=cfg.buildSessionFactory(registry);
        }
        catch(HibernateException he) {
            he.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }//static
```

we need to specify mapping file name seperately becoz cfg.getProperties give only properties tag info

Improvised above code

```
private static SessionFactory factory;
    static {
        try {
        //Actiate hibernate
        Configuration cfg=new Configuration();
        //create ServiceRegistryBuilder
        StandardServiceRegistryBuilder builder=new StandardServiceRegistryBuilder();
        //create Servicer registry
        StandardServiceRegistry registry=builder.configure("com/nt/cfgs/hibernate.cfg.xml").build();
        //build SessionFactory using registry
         factory=cfg.buildSessionFactory(registry);
        }
        catch(HibernateException he) {
            he.printStackTrace();
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }//static
```

No need of giving mapping file seperately

Here ServiceRegsitry is created having the default services binding acrroding HB cfg file entries.. the services are like JdbcSevice,ClassLoader,SchemaManagementService,DialectResolverService and etc..

---

note:: we can findout each category of service implementation by calling getService(-) on registry object ::

```
System.out.println(registry.getParentServiceRegistry().getClass()+" "+registry.getService(ConnectionProvider.class)+"
"+registry.getService(ClassLoaderService.class));
        System.out.println(registry.getService(DialectResolver.class));
```

Adding Custom service to Service Registry
==========

step1) create a custom service implementing org.hibernate.service.Service(I) directly or indirectly.

```
public class CustomJdbcConnectionService implements ConnectionProvider {

    @Override
    public boolean isUnwrappableAs(Class arg0) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public <T> T unwrap(Class<T> arg0) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public void closeConnection(Connection con) throws SQLException {
    System.out.println("CustomJdbcConnectionService.closeConnection()");
    con.close();
    }

    @Override
    public Connection getConnection() throws SQLException {
    System.out.println("CustomJdbcConnectionService.getConnection()");
    Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
        return con;
    }

    @Override
    public boolean supportsAggressiveRelease() {
        // TODO Auto-generated method stub
        return false;
    }

}
```

step2) Custom service to Registry throung RegistryBuilder

```
//Actiate hibernate
    Configuration cfg=new Configuration();
    //create ServiceRegistryBuilder
    StandardServiceRegistryBuilder builder=new StandardServiceRegistryBuilder();
    // add service
    builder.addService(ConnectionProvider.class, new CustomJdbcConnectionService());
```

service type — Custom serivce name

step3) Run The Application..

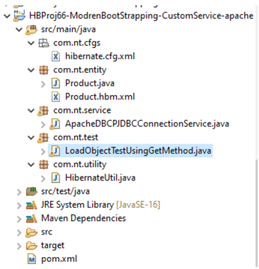
Procedure to make hibernate app working with apache DBCP con pool as custom service
---

step1) add apache dbcp2 dependency to pom.xml file

```
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-dbcp2</artifactId>
  <version>2.9.0</version>
</dependency>
```

step2) Develop the custom service class as ConnectionProvider service

```
public class ApacheDBCPJDBCConnectionService implements ConnectionProvider {

    @Override
    public boolean isUnwrappableAs(Class unwrapType) {
        // TODO Auto-generated method stub
        return false;
    }

    @Override
    public <T> T unwrap(Class<T> unwrapType) {
        // TODO Auto-generated method stub
        return null;
    }

    @Override
    public Connection getConnection() throws SQLException {
        System.out.println("ApacheDBCPJDBCConnectionService.getConnection()");
        BasicDataSource bds=new BasicDataSource();
        bds.setDriverClassName("oracle.jdbc.driver.OracleDriver");
        bds.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
        bds.setUsername("system");
        bds.setPassword("manager");
        bds.setMinIdle(10); bds.setMaxIdle(20);
        return bds.getConnection();

    }

    @Override
    public void closeConnection(Connection conn) throws SQLException {
        System.out.println("CustomJDBCConnectionService.closeConnection()");
      conn.close();
    }

    @Override
    public boolean supportsAggressiveRelease() {
        // TODO Auto-generated method stub
        return false;
    }

}
```



step3) add the Custom service to the registry

In the static block HibernateUtil.java

```
static {
        System.out.println("HibernateUtil.static block");
        //create Configuration object
        Configuration cfg=new Configuration();
        //create SErviceRegistryBuilder
        StandardServiceRegistryBuilder builder=new StandardServiceRegistryBuilder();
        builder.addService(ConnectionProvider.class, new ApacheDBCPJDBCConnectionService());
        //create SErvice Registry
        StandardServiceRegistry registry=builder.configure("com/nt/cfgs/hibernate.cfg.xml").build();
        System.out.println(registry.getParentServiceRegistry());
        System.out.println(registry.getService(ClassLoaderService.class)+" "+
            registry.getService(ConnectionProvider.class)+" "+registry.getService(StrategySelector.class));
        //build SessionFactory
        factory=cfg.buildSessionFactory(registry);
    }
```

step4) Run the application..