

# HIBERNATE

COURSE MATERIAL PART 1

BY

**Mr.NATRAJ SIR**



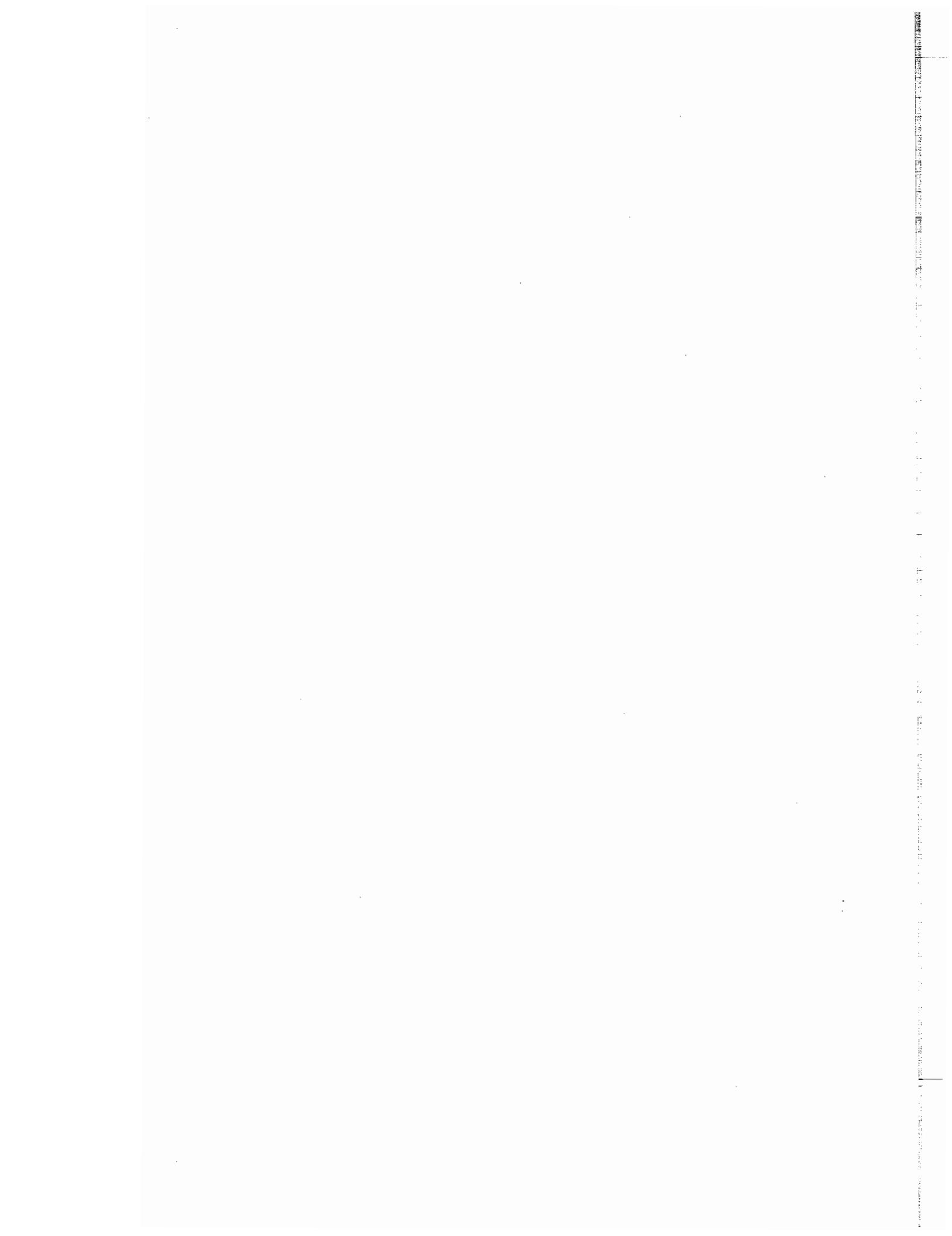
An ISO 9001 : 2008 Certified Company

**sri raghavendra Xerox**

All software language materials available

beside sathyam theatre line balkampet road ameerpet Hyderabad

**cell :9951596199**



# Hibernate Index

## Introduction

<b>Introduction</b>	<b>1 - 8</b>
<b>Hibernate Introduction</b>	<b>9 - 10</b>
<b>Various terminologies</b>	<b>11 - 14</b>
<b>POJO and POJI class</b>	<b>15 - 16</b>
<b>Hibernate Features</b>	<b>17 - 18</b>

## Single Line operations

<b>Save object</b>	<b>19 - 29</b>
<b>Loading object</b>	<b>30 - 31</b>
<b>Delete object</b>	<b>32</b>
<b>Update object</b>	<b>33 - 48</b>
<b>Hibernate Properties</b>	<b>49 - 56</b>
<b>Hbm2ddl.auto property</b>	<b>57 - 61</b>
<b>Service Registry</b>	<b>62</b>
<b>DAO</b>	<b>63 - 70</b>
<b>Layered Application</b>	<b>71 - 77</b>
<b>Working with MySQL</b>	<b>78 - 85</b>
<b>Cache</b>	<b>86 - 98</b>
<b>Generators</b>	<b>99 - 129</b>
<b>Object Versioning</b>	<b>130 - 133</b>
<b>Hibernate Timestamp</b>	<b>134 - 147</b>

## **Connection pooling**

<b>Client side</b>	<b>148 - 160</b>
<b>Server side</b>	<b>161 - 162</b>
<b>Composite identity filed</b>	<b>163 - 166</b>
<b>PostgreSQL</b>	<b>167 - 170</b>

## **Annotation**

<b>Annotation Basics</b>	<b>171 - 174</b>
<b>Anno base Generators</b>	<b>175 - 178</b>
<b>Annotation driven JPA generators</b>	<b>179 - 185</b>
<b>Anno base Versioning &amp; Timestamp</b>	<b>186 - 196</b>
<b>Anno base Composite id</b>	<b>197 - 200</b>

## **HQL**

<b>HQL select Queries</b>	<b>201 - 207</b>
<b>HQL non- select Queries</b>	<b>208 - 213</b>
<b>HQL insert query</b>	<b>214 - 220</b>
<b>Named HQL queries</b>	<b>221 - 226</b>
<b>Anno base Named HQL Queries</b>	<b>227 - 230</b>

## **Native SQL**

<b>Native SQL select Queries</b>	<b>231 - 233</b>
<b>Named Native SQL Queries</b>	<b>234 - 239</b>
<b>Anno based Named Native SQL Queries</b>	<b>240 - 242</b>
<b>Calling PL/SQL procedures and functions</b>	<b>243 - 264</b>
<b>Anno based Calling PL/SQL procedures and fun</b>	<b>265 - 268</b>

## **Criteria API**

<b>Criteria API or (QBC: Query By Criteria)</b>	<b>269 - 276</b>
<b>Pagination</b>	<b>277 - 283</b>
<b>Hibernate Filters</b>	<b>284 - 287</b>
<b>Anno based hibernate filters</b>	<b>288 - 291</b>
<b>Component Mapping</b>	<b>292 - 296</b>
<b>Anno based Component Mapping</b>	<b>297 - 300</b>
<b>Inheritance mapping</b>	<b>301 - 325</b>
<b>Anno based Inheritance mapping</b>	<b>326 - 343</b>
<b>Working with Large Objects</b>	<b>344 - 351</b>

## **Association mapping**

<b>Introduction</b>	<b>352 - 362</b>
<b>One -to-Many uni-directional</b>	<b>363 - 493</b>
<b>Many-to-One uni-directional</b>	<b>394 - 404</b>
<b>One-to-Many/Many-to-One bi-directional</b>	<b>405 - 414</b>
<b>HQL JOINS</b>	<b>415 - 425</b>
<b>Many-to-Many association</b>	<b>426 - 443</b>
<b>One-to-One association</b>	<b>444 - 462</b>
<b>Anno based One -to-Many uni-directional</b>	<b>463 - 491</b>
<b>Anno based Many-to-One uni-directional</b>	<b>492 - 499</b>
<b>Anno based One-to-Many bi-directional</b>	<b>500 - 507</b>
<b>Anno based Many-to-Many association</b>	<b>508 - 521</b>
<b>Anno based One-to-One association</b>	<b>522 - 562</b>
<b>Caching</b>	<b>563 - 564</b>

<b>Transaction Management</b>	<b>565 - 574</b>
<b>Locking in Hibernate</b>	<b>575 - 582</b>
<b>ORM level</b>	<b>583</b>
<b>Hibernate statics</b>	<b>584 - 587</b>
<b>Hibernate Metadata</b>	<b>588</b>
<b>Soft delete</b>	<b>589-598</b>
<b>Hibernate FAQs</b>	<b>599-639</b>
<b>Hibernate 5 ORM Features</b>	<b>640-642</b>

# HIBERNATE

## Introduction:

### Important terminologies:

- a) Persistence Store.
- b) Persistent Data.
- c) Persistence Operations.
- d) Persistence Logic.
- e) Persistence Technologies.

### **Persistence:**

The process of saving and managing data for long time is called persistence.

#### **a) Persistence store:**

The data stored in objects, local variable allocate memory in heap, stack of RAM. Since RAM is temporary memory, the data will be vanished once the application execution is completed. To save that data permanently we need to write data to secondary memory devices like hard disk.

The place where data will be saved and managed for longtime is called **persistence store**.

eg: Files, Database software.

#### **b) Persistent Data:**

The place where data will be saved and managed for long time is called persistent data.

#### **c) Persistence operations:**

Insert, update, delete and select operations are called persistence operations. (also called as CURD/CRUD/SCUD operations).

- |                     |                |
|---------------------|----------------|
| 1) C-create /insert | S---> select   |
| 2) R-read/select    | C ----> create |
| 3) U-update/modify  | U---->update   |
| 4) D-delete/remove  | D---->delete   |

#### **d) Persistence logic:**

The logic that can be used to perform persistence operations is called persistence logic.

eg: I/O streams, jdbc code and etc.

#### **e) Persistence Technologies/ Frameworks:**

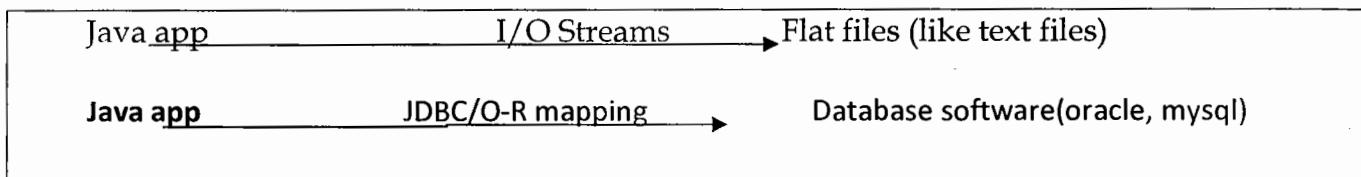
The technologies/ frameworks that are used to develop persistence logics are called persistence technologies/frameworks.

eg: JDBC (Technology), Hibernate (Framework).

#### **Note:**

Persistence technology/framework is given to develop persistence logic to perform persistence operations on the persistence data of persistence store.

- ⇒ Use files as persistence store in small applications because files are light weight and portable.  
eg: Desktop games, mobile games.
- ⇒ Use DATABASE software as persistence store in medium and large scale application because DATABASE software are heavy weight, but provides security and other features like applying constraints, SQL support, maintaining data relations and etc...  
eg: websites, Banking applications, ecommerce applications and etc...



- ⇒ Java application can talk with DATABASE software either by using JDBC or by using O-R mapping (object relations).

### Limitations of JDBC:

- Jdbc does not allow to use objects directly in persistence operations because SQL queries does not allow as objects as inputs. Here we need to convert object data to *simple* values (text data) to use them in persistence operations.
- Jdbc code/persistence logic is Database software dependent persistence logic because it uses Database software dependent SQL queries.
- Since Jdbc code is not portable across the multiple Database softwares, So changing Database software in the middle of project development or in the middle of project production is very complex.
- Jdbc code is having boiler plate code problem (writing some code in multiple applications either with no changes or with minor changes).

eg: The jdbc code placed in multiple applications remains same except SQL query execution and its result processing.

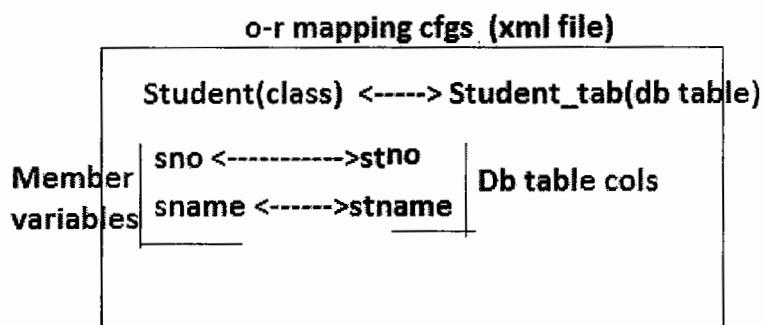
- Jdbc code throws checked exceptions, so programmers may tempt to catch and handle exception, this leads suppressing exceptions or eating exception in the layer where jdbc code is written. This kills the possibility propagating exceptions to other layers.
- The select SQL Query execution given result set object which is not serializable object to send over the network.
- We can't get object directly in persistence operation even though we get data in the form of object i.e. we need to convert object data into simple values to use in persistence operation.
- There is no direct support for services like caching, TRANSACTION management and etc...
- To overcome most of these problems use O-R mapping to develop the persistence logic. Due to above limitations we can't use JDBC to develop persistence in all kind of situations.

## Q. What is O-R mapping?

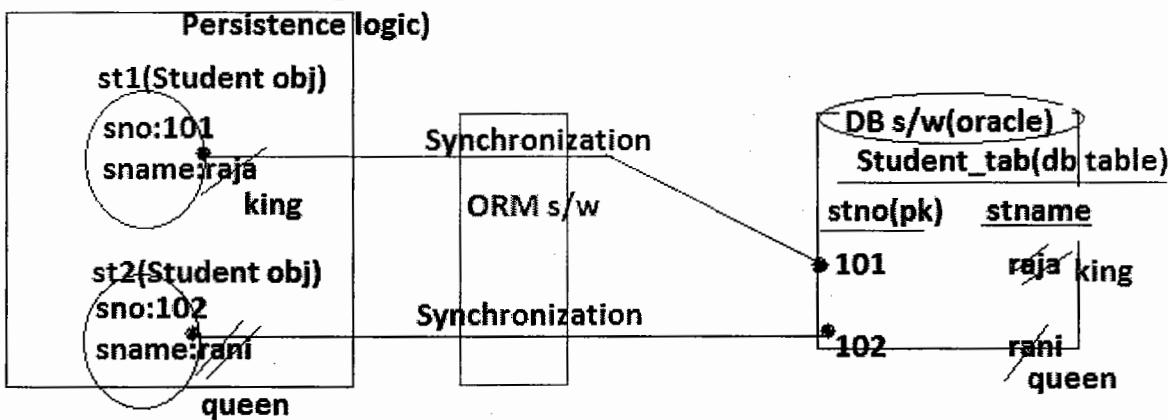
- The process of mapping java class with DATABASE table,java class member variables with Database table columns and making the objects of java class representing DATABASE table records having synchronization between them is called **O-R mapping**.
- The synchronization between object and DATABASE table row is nothing but the modifications done in object will reflect to DATABASE table row and **vice-versa**.
  - ORM software are required to develop o-r mapping persistence logic  
eg: Hibernate,iBatis.

### Java class

```
class Student{
    private int sno;
    private String sname;
    ....//setters and getters
    ...
}
```



### Java App (o-r mapping)



- While working/developing O-R mapping persistence logic we write all the logics in the form of the objects i.e. we can perform all the persistence operations directly through objects without any SQL queries support. Due to this O-R mapping persistence logic becomes DATABASE software independent persistence logic.
- In O-R mapping configuration, we use either xml file or annotations to specify which java class should be mapped with which DATABASE table of the project.
- In O-R mapping the ORM software takes objects as inputs and gives object as output so we need not to convert objects notation data to simple values and vice-versa.

List of ORM softwares (java based)

EJB: from Sun Ms (Oracle corporation)  
 Hibernate: from SoftTree (Redhat)  
 ibatis: from Apache  
 Toplink: from Oracle Corp.  
 JDO: from Adobe  
 JPA: from Sun ms (Oracle corp.)

- Every ORM software internally uses JDBC code and SQL queries to complete the persistence operations ,but it hides that process from programmer ,so programmer thinks that he is doing object based persistence logic development.This indicates that every ORM software provides abstraction layer on JDBC.
- ORM software also converts given object data into simple values as required for persistence operations and simple values into object as required for application.

**Q. Where can we use Jdbc and where can be use ORM software like hibernate to develop the persistence logic?**

- The application that is not having user interface and that deals with other application supplied data periodically is called **offline application**.  
 eg: Census data processing, survey data processing.

**The application that is having user interface and deals with end user supplied data is called online application.**

eg: Internet websites ([www.flipkart.com](http://www.flipkart.com), [www.gmail.com](http://www.gmail.com).)

→ While developing offline application that has to process huge amounts of data that comes batch by batch periodically use **JDBC** because it does not create much objects while processing that data if you use hibernate in this offline application system may crash while processing huge amount of data because hibernate or ORM software creates huge amounts of objects to represent huge amounts of records .

Teachers collects data ---> gives to Data Entry Operators ---> Data Entry operators writes data to Excel sheets--> provides excel sheets to offline application to perform

```

|---->validate the data
|---->Correct the data
|---->Process the data
|---->Stores into oracle DATABASE software
  
```

→While developing online application that deals with limited amount of data given by end user use **ORM software** like hibernate for persistence logic to take the advantage ORM software features like caching, DATABASE portability and etc... which are not there in JDBC.

Important points about ORM:

- ORM software like hibernate take care the mismatches between java language features and RDBMS data table designing.

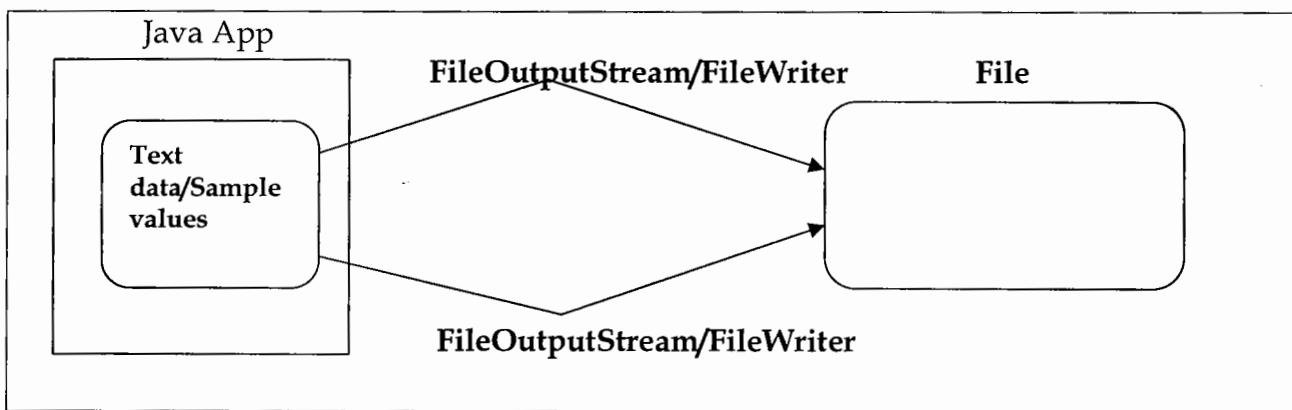
- We design java application based object oriented program principle and we designing DATABASE table based normalization rules in this process some mismatches may encounter between java classes and DATABASE tables as part O-R mapping the ORM software resolves these problems.

eg: Java supports inheritance between classes but the Database tables mapped by these classes do not support inheritance so we can use ORM framework software like hibernate to resolve this problem to perform persistence operation using the object of these java classes.

**There are multiple approaches to perform persistence operations from our java application:**

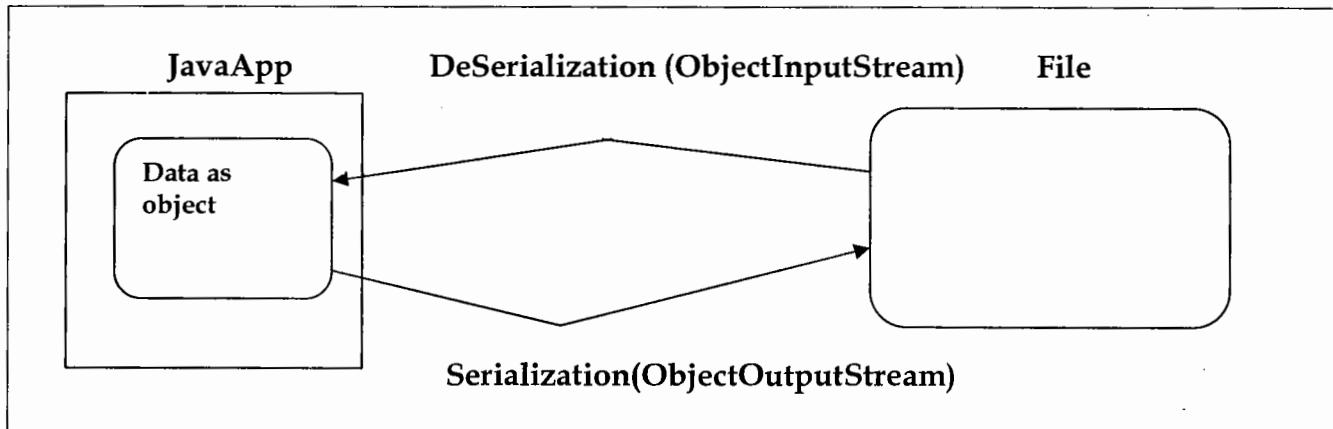
**Approach1:**

If application is small scale desktop application or mobile application generating text data/simple values then use files as persistence stores and use I/O streams for persistence operations.



**Approach2:**

If Application is small mobile/desktop application generating data in the form of objects then use file as persistence store and I/O streams (serialization and deserialization) for persistence operations.

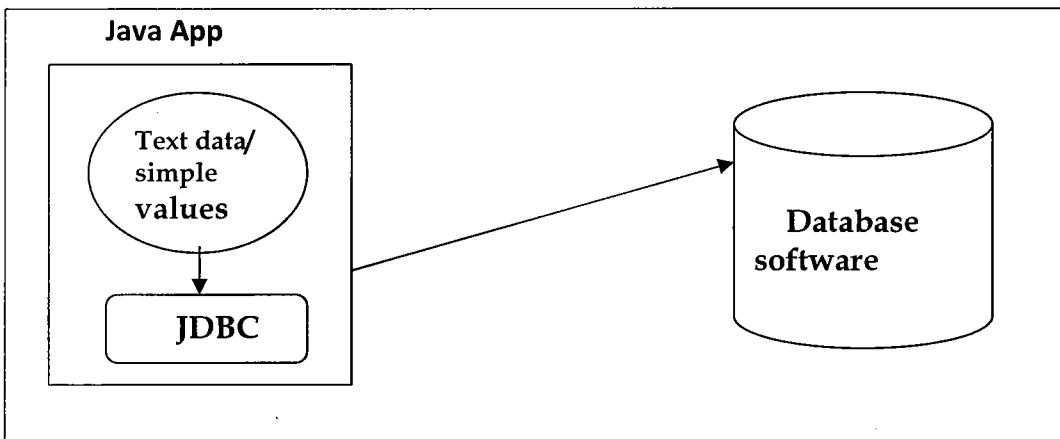


The process of converting object data into bits and bytes is called serialization. These bits and bytes can be written to file or DB s/w and etc...

The process of reading data from file and creating object having that data is called deserialization.

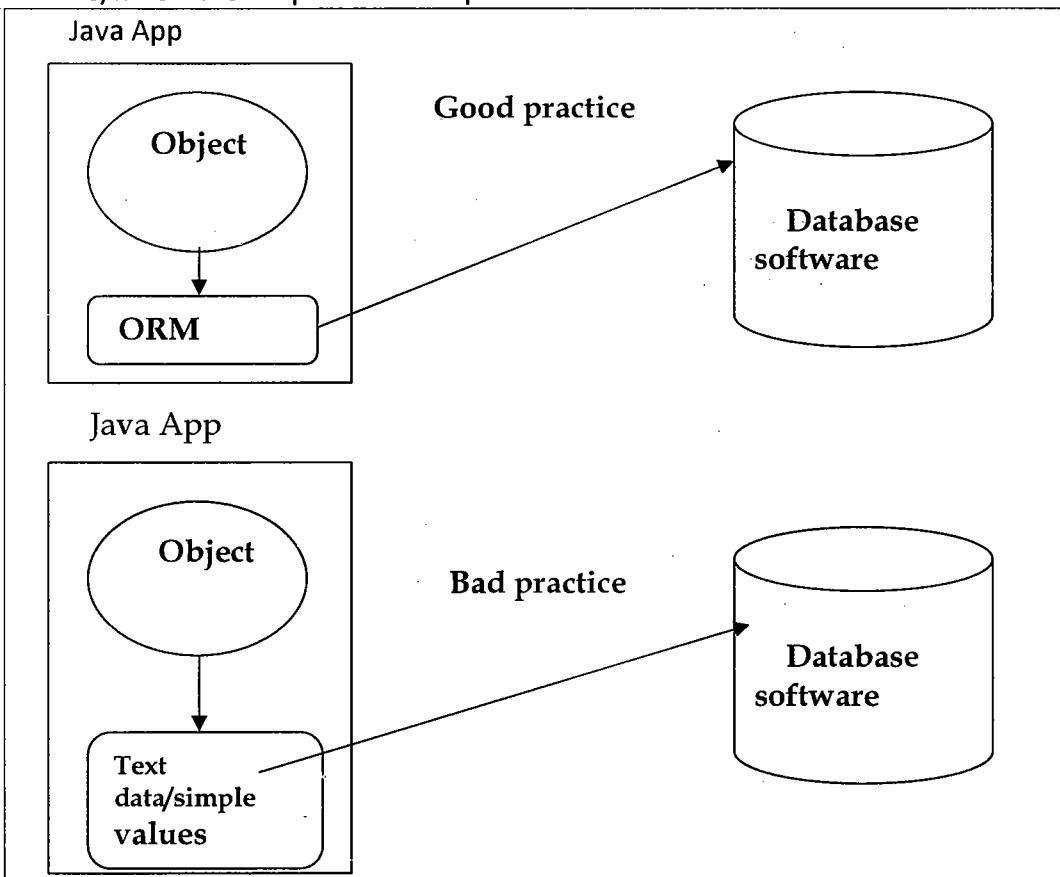
**Approach3:**

If Application is medium scale application generating less amount data in the form of text or simple values then Use DATABASE software as persistence store and Use JDBC code to persist data.

**Approach4:**

Application is medium /large scale application dealing with less amount of data

- Use DATABASE software as persistence store
- Use JDBC/hibernate for persistence operations.

**Approach5:**

Application is medium/large scale application dealing with huge amount object/text data (**comes batch by batch**)

- Use DATABASE software as persistence store
- Use JDBC to write persistence logic

**Q. What is difference between programming language, Software Technology and Framework?****Programming language:**

It is directly installable software as raw material having basic features of application development it defines the syntax a semantic of programming.

Syntax: rules of coding

Semantic: structure of coding

It is base for creating software technologies, operating systems, Database softwares, software tools and etc...

eg: c, c++, java

**Software Technology:**

It is a software specification defining set of rules and guidelines to create a software by using one or another programming language support .Technology is not installable but the technology based software is installable,Working with this software is nothing but working with technology.

eg: JDBC, Servlet, JSP, EJB etc...

**There are two types of technologies:****1. Open technologies:**

Here the rules and guidelines of technologies are open for all the vendors to create implementation softwares.

eg: JDBC, Servlet, JSP, EJB etc...

**2. Proprietary technologies:**

Here the rules and guidelines of each technology are specific to one vendor and only that vendor is allowed to develop implementation softwares

eg: all MicroSoft technologies.

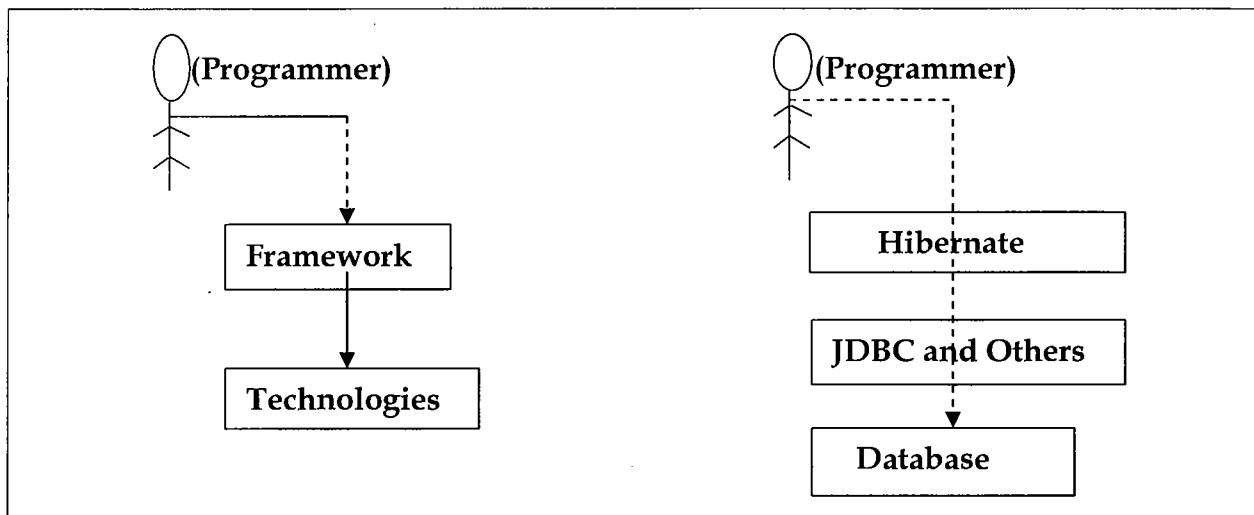
→All java technologies give rules in the form of interfaces and guidelines in the form of classes.

→Sun Ms provides rules and guidelines in the form of API packages and the vendor companies provide implementation softwares based on the rules and guidelines.

**Framework:**

- Framework is a special installable software that provides abstraction layer one or more existing technology to simplify to process of application development.
- Framework is special software that internally uses one or more technologies having the ability to generate common logics to the application dynamically and makes programmer to develop only application specific logics.
- Framework internally uses technology to generate common logics but never makes programmer to know/bother about that, This is nothing but providing abstraction layer on technologies.

- If we work with technology, we need take care of both common and application specific logics. If we work with framework we need to write only application specific logics.



### Advantages with framework:

- Provides abstraction layer on technologies
- Solve boiler-plate code problem
- Simplifies application development
- Provide faster application development and improves the productivity
- The exception handling is optional by converting checked exceptions into unchecked exception.
- Framework API designs by keeping real time use case in mind.

→Based on the kind of applications that we can develop three types of java framework:

#### 1.Web application Frameworks:

Provides abstraction layer on Servlet, Jsp technologies to develop MVC2 architecture based web applications

eg:

struts:	From Apache
Spring MVC:	From Interface21
Webwork:	From Open symphony
Tapstrey:	From Adobe
ADF:	From Oracle corp
JSF:	From Sun ms (oracle corp.)

#### 2.ORM Frameworks:

Provides abstraction layer on JDBC technology to develop Database portable and objects based persistence logic.

eg:

Hibernate :	SoftTree(redhat)
Toplink:	From Oracle corp.
JDO:	From Abobe
JPA:	From Sun ms(Oracle corp.)

**3.Application Frameworks:**

→Provides abstraction layer on multiple JSE, JEE technologies and simplifies all kind of application development.

eg: spring : From Interface21

**There are two types of Frameworks based on mode of Application development:**

**1. Invasive Framework:**

Here the resources/classes of the application that contains logic are tightly bound with framework apis. i.e. each resource of the class implements or extends the framework api interface or class. Due to this we cannot use the classes/resources in other framework environment.

eg: structs 1.x.

**2. Non invasive Framework:**

The resources/classes of the application that contains logic are loosely coupled with framework apis i.e. these classes need not to implement framework api interfaces or need not to extend from framework api classes. This indicates we can place these logics in ordinary java classes.

Due to this we can make these classes run in any framework environment by supplying that framework library (jar files).

eg: spring, hibernate, structs 2.x

## **Hibernate:**

Type:	java ORM framework/java ORM software or tool
Version:	5.x(compatible with jdk 1.8)
Vendor:	SoftTree(Redhat)
Creator:	Open source Mr. Gavin King

To download software: download as zip file from [www.hibernate.org](http://www.hibernate.org).website

To install software : extract zip file.

->To use this software arrange the jar files, that comes after extracting the zip as run time library of the application and use hibernate api in the development of persistence logic

Tutorials:	<a href="http://www.mkyong.com">www.mkyong.com</a> <a href="http://www.tutorials.com">www.tutorials.com</a> <a href="http://www.javagreeks.com">www.javagreeks.com</a>
------------	--

Books:	hibernate live hibernate in action/persistence with hibernate.
--------	---

->Since programming languages,frameworks are installable softwares they provide interfaces and their implementation classes directly. When it comes to technologies since they are not installable softwares so they provide only interfaces but the implementation classes will be provided by third party vendors.

<hibernate 5.1-home>\changelog.txt Transaction file various version of hibernate and their release dates.

<hibernate 5.1-home>\documentation folder contains api, docs, tutorials, developer guidance docs.

<hibernate 5.1-home>\lib contains jar files respecting hibernate library(both main and dependent jar files)

<hibernate 5.1-home>\project contains sample utility files,, source code, apis ,projects.

<hibernate 5.1-home>\lib\hibernate-core-<ver>- final.jar represent hibernate api and its having multiple dependent jar files.

→ If application uses third party api then we need to add the hibernate api related both dependent and main jar files to class path/build path

→ If the classes of a.jar are using the classes of b.jar file then when can say b.jar file is dependent jar file to a.jar file.

### **Hibernate Framework:**

- Hibernate is open source, non invasive , light weight java ORM framework given by SoftTree(Redhat) to develop object based on DATABASE software independent O-R mapping persistence logics in all kind of java applications.
- 
- Along with hibernate installation we get its source code in <Hibernate\_home> project\hibernate-core folder, this makes hibernate as open source.
- Hibernate is light weight and this software comes in small size.
- To execute hibernate code we don't need heavy weigh container like tomcat and etc... Just having JDK and hibernate libraries we can execute hibernate code.
- The classes whose objects will be used O-R mapping persistence logic development are ordinary classes or light weight classes.
- Hibernate persistence logic is intigretable with any java technology/code java framework based application
- EJB Entity bean component allows as developing object based O-R mapping ,but these components are heavy weight components.
- To execute EJB components we need EJB container which is part of heavy weight Application server like weblogic, the design are
  - a) The classes of EJB component we need EJB container which is part of heavy weight application server like **weblogic**.
  - b) The classes of EJB component development are not ordinary classes they must implement EJB API interfaces and must extend from EJB API classes.

→ Based on These reasons we can say EJB components of heavy weight.

→ In real-time most of applications will be developd based on MVC architecture

**M-** for model (data) to represent for persistence logic and business logic that gives data (like Account Officer)

**V-** for view to represent for presentation logic (like beautician)

**C-** Controller to represent for integration logic. (Like super visor)

Jsp--->Servlet---->java class with java bean --->Database software

(v)           (c)                   (M)

### **Note:**

Controller monitors all the activities of application execution. It takes the request passes to model components, gathers the result from model components and passes the result to view components.

Jsp--->Servlet---->service class ---> DAO class--->Database software

(v) (c) (m-->b. logic) (m-->persistence logic)

Jsp--->Servlet---->EJB session bean ---> EJB EntityBean

(v) (c) (m-->b. logic) (m-->persistence logic)

struts--->EJB session bean---->DAO class ---> Database software

(v)(c) (m-->b. logic) (m-->persistence logic)

struts--->EJB session bean---->Hibernate ---> Database software

(v)(c) (m-->b. logic) (m-->persistence logic)

struts--->Springcore/jee---->Hibernate ---> Database software

(v)(c) (m-->b. logic) (m-->persistence logic)

SpringMVC--->Springcore---->SpringORM/SpringDAO---> Database software

(v)(c) (m-->b. logic) (m-->persistence logic)

SpringMVC--->web services---->DAO(JDBC/hibernate)---> Database software

(v)(c) (m-->b. logic) (m-->persistence logic)

SpringMVC--->web services----> Database software

(v)(c) (m-->b. logic+ persistence logic)

#### **List of view layer Technologies:**

Html, javascript, velocity, freemarker, jsp, ajax, dojo, jquery, angularjs and etc....

#### **List of controller Technologies:**

Servlet, Servlet filter

#### **List of model layers Technologies/framework to develop business logic)**

Rmi,EJB, corba,webservices, java class with java bean, spring core/jee and etc...

#### **List of model layer Technologies/framework to develop persistence logic**

JDBC,hibernate,ejb entity,jpa, toplink,jdo and etc...

#### **List of web application frameworks to develop view and controller layer logics:**

Struts,weblogic, wicket, tapstey, springMVC, adf, cocoon, and etc...

In real world Hibernate is just used for persistence logic i.e. it can't be used to develope other layer logics ,more ever it will used in combination with other technology/ framework like spring.

#### **Understating various terminologies related to classes:**

##### **Utility class:**

The java class that contains logic to manipulate certain data or to perform all the operation certain requirement in multiple angles is called utility class.

eg: String utility classes, Data utility, DATABASE connection utility classes etc...

**Java Bean:**

The java class that is developed with standards is called java bean class.

**Standards are:**

- Must be public class.
- Suggested to implement `java.io.Serializable` (I).
- All members variables are must be private, Non-static (these are called bean properties).
- Every bean property must have one setter and getter method.
- setters/getters are useful to set/get data to/from bean properties.
- Must have direct (given by programmer) or indirect (generated by java compiler) 0-param compiler

**eg:**

```
public class Student{  
    //bean properties  
    private int sno;  
    private String sname;  
    //setters & getters  
    public int getSno(){  
        return sno;  
    }  
    public void setSno(int sno){  
        this.sno=sno;  
    }  
    public String getSname(){  
        return sname;  
    }  
    public void setSname(String sname){  
        this.sname=sname;  
    }  
}
```

**In real world java bean will used in modes they are:**

- **BO class(Business Object)**
- **DTO class(Data Transfer Object)**
- **VO class(Value Object)**

**Note:** Every Java Bean is a POJO class, but not vice-versa.

- ⇒ The Java Bean class whose objects hold the input values given by enduser (like form data) or output values given by the application is called **VO class (Value Object class)**. This class generally contains all String properties.
- ⇒ The Java Bean class whose object can be sent over the network from one resource to another resource is called **DTO class (Data Transfer Object class)** .DTO class implements `Serializable` interface and contains different types of bean properties.

- ⇒ The Java Bean class, whose object holds data as required for persistence operations is called BO class (Business Object class). This class bean properties types and DATABASE table column types must be compatible with each other.

→ BO class is also called as Domain class/Model class /Entity class

→ VO/BO/DTO classes are basically POJO classes.

#### Note:

=> In real time java beans are used as helper classes to pass data between one component class to other components classes (1 layer to another layer). In real time designing method having more than 3 param is bad practice because while calling that method confusion will come towards passing argument values, To overcome this problem, Design method having java bean as parameter type so that multiple values can be received as single object value.

#### VO class:

The java bean that contains only String properties and whose object holds input values given by end user and output values as required for end-user for displaying result is called VO class.

```
public class StudentVO{

    private String sno;
    private String sname;

    public String getSno(){
        return sno;
    }
    public void setSno(String sno){
        this.sno=sno;
    }
    public String getSname(){
        return sname;
    }
    public void setSname(String sname){
        this.sname=sname;
    }
}
```

#### DTO class:

The java bean that holds the data to send from one component to another component class of same layer or different layers with or without network is called DTO class this class implements serializable interface.

#### **DTO class**

```
public class StudentDTO implements Serializable{
    private int sno;
    private String sname;
```

```

public int getSno(){
    return sno;
}
public void setSno(int sno){
    this.sno=sno;
}
public String getName(){
    return sname;
}
public void setName(String sname){
    this.sname=sname;
}
}

```

**BO class:**

The java bean whose object holds data as persistable data or whose data is given by persistence logic is called BO class or Entity class or domain class or model class or persistence class we generally take this class 1 per table basis.

→ This class properties type and DATABASE table column types will be compatible with each other.

```

public class StudentBO{
    private int sno;
    private String sname;

    public int getSno(){
        return sno;
    }
    public void setSno(int sno){
        this.sno=sno;
    }
    public String getName(){
        return sname;
    }
    public void setName(String sname){
        this.sname=sname;
    }
}

```

**Note:**

In hibernate programming the java class object that represents records in O-R mapping should be designed based on Java Bean i.e, acting as BO class. Technically it is called as domain class in hibernate programming.

**Component class:**

The java class that contains member variables and methods is called component class. A perfectly designed java class with business logic in methods is called component class.

```
public class StudentService {
    private int sno;
    private String sname;
    public void processData(int sno, String sname) {
        -----
        -----
    }
}
```

**Questions:**

- 1) Every java bean is a POJO class? (yes)
- 2) Every POJO class is a java bean ?(no)
- 3) Every component class is a POJO class? (no)
- 4) Every POJO class is a component class? (no)
- 5) Component class can be developed as POJO class ?(yes)
- 6) POJO class can be developed by component class? (yes)
- 7) Every component is a java bean ?(no)
- 8) Every java bean is a component class? (no)

**POJO class :( Plain Old Java Object):**

- It is ordinary classes without any specialties.
- It is normal java class not bound to any technology/framework.
- Java class that is not implementing technology/framework API interface and not extending from technology/framework API class is called **POJO class**.

eg: The java class of hibernate application that is not implementing hibernate API interfaces and not extending from hibernate API class is called POJO class.

```
class Demo{
    .....
    .....
}

“Demo” is POJO class

class Test implements Serializable{
    .....
    .....
}
```

Test is POJO class because Serializable (i) is part of java/jdk apis.

```
class Test extends Thread{
    .....
    .....
}
```

Test is POJO class because Thread is part of java apis.

```
class Test implements java.sql.Connection{  
.....  
.....  
}  
Test is not POJO class because java.sql.Connection(i) is part of JDBC technology api.
```

```
class Test extends HttpServlet{  
.....  
.....  
}  
Test is not POJO class because HttpServlet is part of Servlet api
```

```
class Test extends Demo{  
.....  
.....  
}  
class Demo extends HttpServlet{  
.....  
.....  
}  
Demo is not POJO class  
Test is not POJO class
```

```
class Test {  
  
public void m1(){  
.....  
....//hibernate api code  
  
}  
}  
Test is POJO class
```

```
@Entity  
class Test {  
.....  
.....  
}  
Test is a POJO class
```

#### Important points about POJO class

- While compiling and executing we may or we may not require third party library.
- While developing POJO class we can use technology or framework API inside the class or we may use annotation of same API.
- POJO class is not against of inheritance but it is against of extending from technology/framework api classes or implementing technology / framework api interfaces.
- Every java bean is POJO class but every POJO class not to be java bean.
- In hibernate application the classes that are mapped with Database table will be taken as BO class/Domain class which is a POJO class.

**POJI (plain old java interface)**

- It is ordinary interface not a special interface.
- It is normal interface not bound to any technology/ framework
- The java interface that is not extending from technology/framework specific interfaces is called POJI.

```
interface Test{
```

```
.....
```

```
.....
```

```
}
```

**Test is POJI**

```
interface Test extends java.rmi.Remote{
```

```
.....
```

```
.....
```

```
}
```

**Test is not POJI because java.rmi.Remote part of rmi technology.**

Hibernate supports POJO/POJI model programming so it is called light weight

**Hibernate Features:**

- Supports **POJO/POJI** model programming.
- Hibernate is lightweight.
- Hibernate persistence logic is Database software independent i.e. it is portable across the multiple DATABASE softwares.
- Gives DATABASE software independent query language called **HQL(Hibernate Query Language)**
- Gives built-in JDBC connection pool and also allows to use third party JDBC connection pool like **c3p0, proxool** and Server managed **JDBC** connection pool.
- Provides caching/ buffering support which holds DATABASE table records across the multiple same requests and reduces network round trips between java application and DATABASE software.
- **Supports Lazy loading:**  
Hibernate can load record from table into application not when code is executed but when record utilization started in the application, this is called loading the record from Database table on demand basis(Lazy loading).
- **Supports Versioning:**  
Allows to keep the track of when the record is inserted/updated through hibernate persistence logic (maintains version number)
- **Supports Time stamping:**  
Allows to keep the track of when the record is inserted/updated through hibernate persistence logic (maintains date and time values)
- Allows to create dynamic schema (DATABASE table creation dynamically).

- Hibernate is capable of creating DATABASE table dynamically in new DATABASE software when the DATABASE software of the change dynamically.
- Gives Criteria API to generate optimized SQL queries. Hibernate itself takes the responsibility to generate best SQL query which give good performance.
- Allows to develop object based persistence logic
- Support object based relationships like **one to one, many to many, many to one, many to one and etc...**
- Easy to integrate with other java technologies and java frameworks  
eg: struts with hibernate, spring with hibernate.
- Exception handling is optional because hibernate exceptions are unchecked exceptions.
- Allows to call PL/SQL procedure and functions
- Supports both xml based configurations and annotation based configurations.
- Easy to learn and easy to use.

**Note:**

To utilize all above features of hibernate programmers are preferring to use hibernate to develop persistence logic even though same logic can be developed using JDBC.

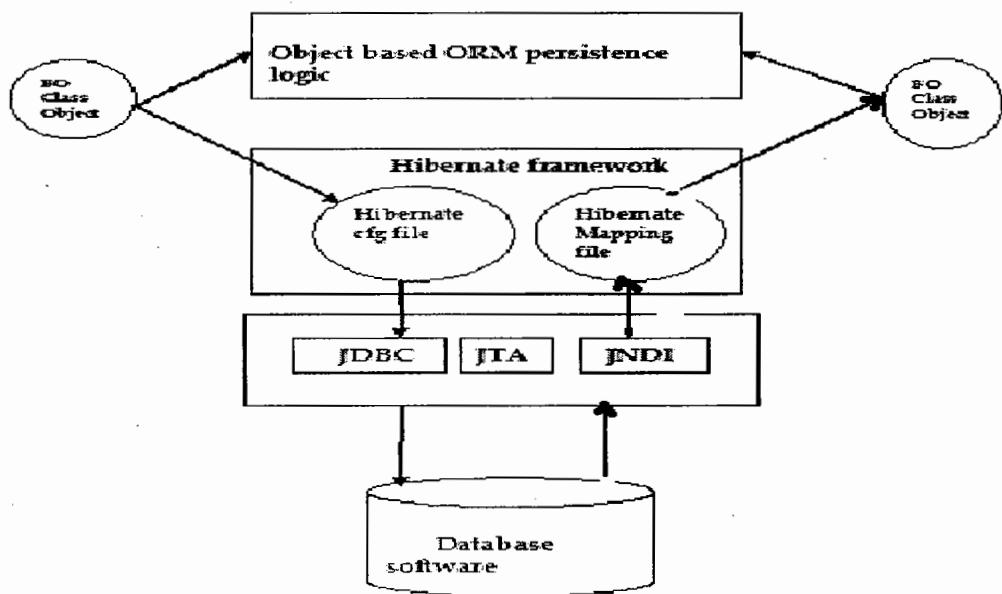
**Hibernate internally uses 3 technologies:**

- 1) JDBC
- 2) JTA
- 3) JNDI

**JDBC:** Uses this to develop SQL queries based persistence logic.

**JTA:** Uses this for TRANSACTION management to execute sensitive logics by applying do everything or nothing principle.

**JNDI:** Uses this to utilize the JDBC connection objects that are collected from JDBC con pool.

**Hibernate high level diagram architecture:**

**To save object from application:**

Application gives object to hibernate framework, framework decides destination Database software based on configuration file(xml) and destination table based on mapping file(xml) .Hibernate framework generates JDBC code with SQL insert query to insert record into Database table.

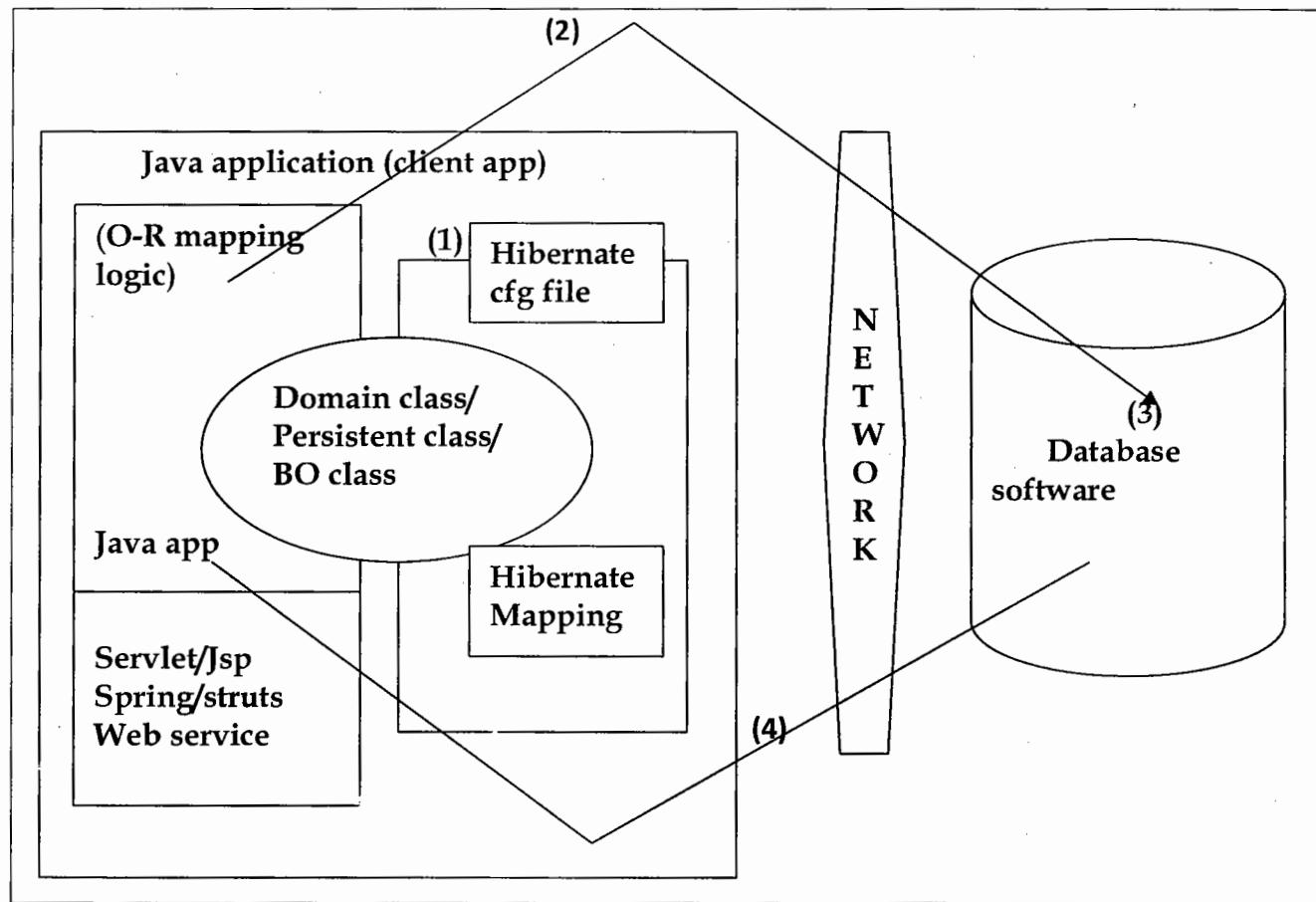
**To retrieve object from application:**

Hibernate internally executes SQL select query with JDBC code support and gets ResultSet object and keeps that record in BO/Domain class object and gives that object to application

**In Hibernate application development the important terminologies are:**

- **Saving object** means saving the data of the object as record in DATABASE table.
- **Updating object** means updating the record of Database table that is represented by object.
- **Deleting object** means deleting the record of DATABASE table that is representing by object.
- **Loading object** means selecting the record from the Database table in to object.

Here objects means the BO/Domain class objects, these objects are actors of hibernate programming who can play the persistence operation roles like inserting record, updating record, deleting record and selecting records and etc...operations..

**Typical Hibernate Application:**

1. O-R mapping persistence logic will be developed in java application using hibernate resources and hibernate api.

2. This O-R mapping persistence logic passes the objects based persistence instructions to hibernate framework .
3. Hibernate framework internally uses JDBC code and SQL queries to complete persistence operations.
4. Hibernate framework gathers the results from DATABASE software given by JDBC code and passes those results back to the application in the form of objects.

#### **Hibernate configuration file:**

- Any <file name>.xml can be taken as hibernate configuration file. If no file name is given hibernate.cfg.xml name will be taken as default file name.
- This file content will be used by hibernate framework to locate destination DATABASE software by using the given JDBC driver details.
- This file contains property names and values where property name are fixed and values will be changed based on the DATABASE software and JDBC software we use.

#### **→ This file contains 3 types of details:**

1. Connection properties (for DATABASE connectivity)
2. Hibernate properties(for giving instruction to hibernate framework)
3. Mapping files names
  - Generally we take this file on 1 per database basis.  
→*Hibernate Properties are like this  
connection.driver\_class, connection.url and etc...*

#### **Hibernate Mapping file:**

- This xml file contains of O-R mapping configurations like mapping classes with Database tables and class properties with DATABASE table columns.
- Hibernate framework decides the Database table to perform persistence operations based on this mapping file configuration.
- There is no default file name for this file but we can take any <filename>.xml as the hibernate mapping file name. The standard notation of this file name is <filename>.hbm.xml.
- We generally take mapping file on 1 per Database table basis and we need to configure/specify all the mapping file names in hibernate configuration file.

#### **Client Application:**

- It is not client to Hibernate framework, it is client to Database software. This can be any standalone java application or java technology based application or java framework based application.
- This application uses Hibernate libraries(jars), hibernate API to activate hibernate framework and to develop objects based O-R mapping persistence logic ,to provide persistence instructions to framework in the form of domain class objects and to get results in the form domain class objects.
- Resource configuration means specifying the details of the class or interface in the xml file to make the underlying framework or server recognizing that resource is called resource configuration.
- By configuring java bean class in Hibernate mapping file we will make that class as domain class of hibernate for developing objects based persistence operations.

**Q. What is synchronization between object and table rows and who will take care of this synchronization in which manner?**

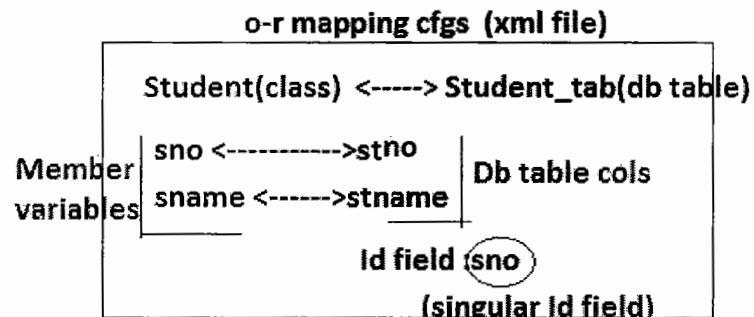
**Ans:**

- If the modification done in objects are reflecting to table rows and vice-versa then it is called synchronization. Hibernate framework takes care of this synchronization by taking the support of JDBC code and SQL queries internally.
- Every java object is identified by jvm through hashcode, similarly every object of domain class will be identified by the Hibernate through **identity value** or identifier value this value will be generated based on identifier filed/property configuration done in the mapping file. We configure one or other property of domain class as identifier filed/identity field.
- Hibernate framework generates update query having domain class object identifier value as the criteria value to synchronize the changes done in object to db table row ,Similarly ,it generates select query having identifier value as criteria value to synchronize changes done in Database table row to object as shown in the diagram.

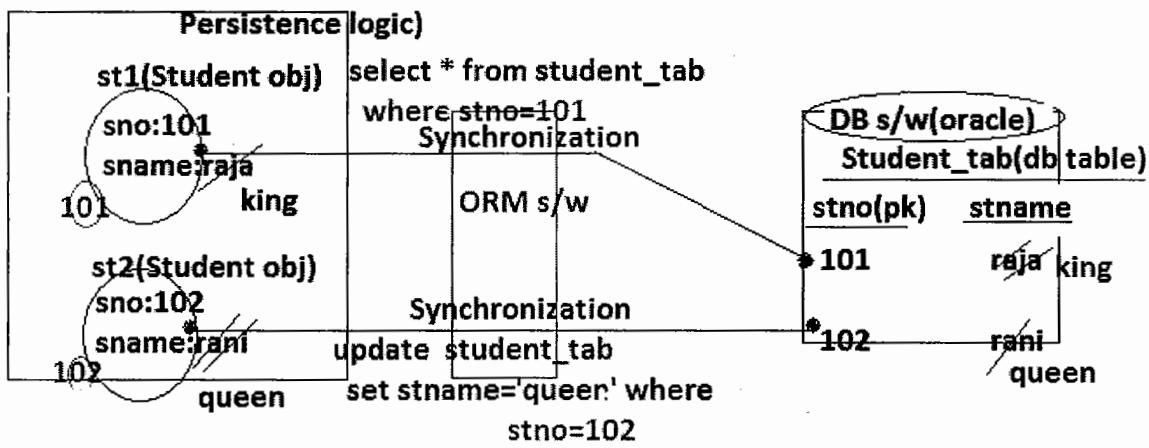
=> Using o-r mapping to map more java classes of Java App with more Db tables of Db s/w and we can go for objs based DB s/w independent persistence logic development with out using any SQL Queries . This also makes the o-r mapping persitence logic as portal persitence logic across the multiple Db s/ws.

**Java class**

```
class Student{
    private int sno;
    private String sname;
    ....//setters and getters
    ...
}
```



**Java App (o-r mapping)**



**Q. What is strategy that we need to follow while configuring identity filed in Hibernate mapping file?**

**Ans:**

- If pk(primary key) constraint is applied one column of Database table then it is called **singular pk constraint**. if pk constraint applied on multiple columns of Database table then it is called **composite pk constraint**.
- If only one property of domain class is configured as identity filed then it is called singular identity filed/singular identifier.
- If more than one property of domain class is configured as identity filed then it is called as composite identity filed/composite identifier.

**We can use the following strategies to configure identity filed:**

**Strategy1:**

If Database team gives Database table having singular pk constraint then configure that column related property of domain class as singular identity filed in mapping file using **<id>** tag.

**Strategy2:**

If Database team gives Database table having composite pk constraint then configure those multiple columns related myliple properties of domain class togather as composite identity filed in mapping file using **<composite-id>** tag.

**eg: programmers\_project(composite pk constraint on pid, projid)**

**pid pname projid projname**

101 raja 1001 proj1

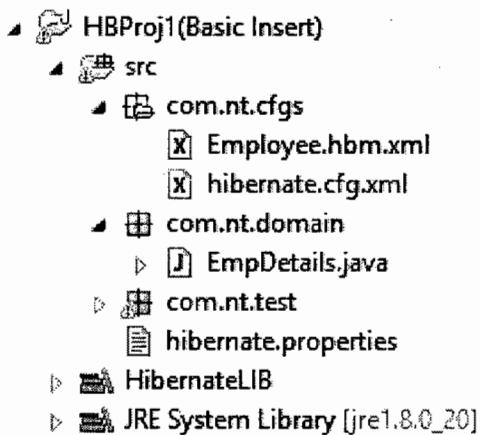
101 raja 1002 proj2

102 rani 1001 proj1

**Note:**

If Database team gives DATABASE table without pk constraint then reject that Database table to use hibernate application.

**First application development to save object (inserting record into Database table)**



Jar file in build path : <HB5.1\_home>lib\required folder all jar file + ojdbc14/ojdbc6.jar

**DATABASE table in oracle:**

Employee

|---->Eid (n) (pk)

```
|---->Firstname (vc2)
|---->Lastname (vc2)
|---->Email (vc2)
```

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
    <id name="no" column="EID"> <!-- Singular Id field -->
      <generator class="native"/>
    </id>
    <property name="fname" >
      <column name="FIRSTNAME"/>
    </property>
    <property name="lname" >
      <column name="LASTNAME"/>
    </property>
    <property name="mail" >
      <column name="EMAIL"/>
    </property>
  </class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private int no;
```

```
private String fname;
private String lname;
private String mail;

public EmpDetails(){
System.out.println("EmpDetails:0-param constructor");
}

public int getNo() {

return no;
}
public void setNo(int no) {

this.no = no;
}
public String getFname() {

return fname;
}
public void setFname(String fname) {

this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}
```

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

import com.nt.domain.EmpDetails;

public class ClientApp {

    public static void main(String[] args) {
        //Activate Hibernate f/w
        Configuration cfg=new Configuration();
        //Read and store HB cfg file , mapping file data into Configuration obj
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        //create SessionFactory obj
        SessionFactory factory=cfg.buildSessionFactory();

        // get Session obj
        Session ses=factory.openSession();

        // create Domain class obj with data
        EmpDetails ed=new EmpDetails();
        ed.setNo(1021); ed.setFname("raja");
        ed.setLname("rao"); ed.setMail("raja@x.com");

        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(ed); //saves obj (inserts the record)
            tx.commit();
            System.out.println("Object Saved /Record inserted");
        }
        catch(Exception e){
            tx.rollback();
        }

        //close objs
        ses.close();
        factory.close();
    }//main
}//class
```

**Note:**

- Always develop Hibernate configuration file based on reference document available in <hibernate\_home>\project folder.
- To make programmer of hibernate to create xml document by using set of tags and attributes. The hibernate framework is supplying rules in the form DTD(Document Type

Definition). programmers should create hibernate configuration based on the rules that are available in hibernate-configuration-3.0.dtd file.

- Similarly programmer must develop hibernate mapping file based on hibernate-mapping-3.0.dtd file
- In hibernate configuration file the property names are fixed but the value will change based on the JDBC driver we use and the Database software we use.
- Transaction is a unit of work which can be committed or rollback. In hibernate persistence logic, all non select operations must take place in transactional environment.

#### **In hibernate configuration file we can place 3 types of properties in configuration file:**

1. Connection properties: like connection.driver\_className, connection.url and etc..
2. Hibernate properties: like show\_sql, dialect and etc....
3. Mapping file names

#### **Note:**

We can make hibernate framework showing generated SQL query as the log message by suing "show\_sql" hibernate property that is placed in configuration file.(hibernate.cfg.xml)

```
<property name="hibernate.show_sql">true</property>
```

#### **Important Observations**

- Hibernate framework internally uses PreparedStatement to execute the generated Sql queries.
- The method called on session object like save(-) method gives persistence instructions to hibernate framework ,but hibernate framework completes that persistence instruction on Database software only when tx.commit() method is called.
- Hibernate internally reads the given domain class object data by calling getter methods .
- In Hibernate application we must develop domain class as perfect java bean having getter and setter methods, because it uses getter methods to read domain class object data and to write that data to Database table, it also uses setter methods to write data to Domain class object by retrieving data from Database table .
- In hibernate 5.x SessionFactory object is the object of org.hibernate.internal.SessionFactoryImpl(C) that implements org.hibernate.SessionFactory(I).
- In hibernate 5.x Session object is the object of org.hibernate.internal.SessionImpl(C) that implements org.hibernate.Session(I).

→If the xml document satisfies just syntax rules then it is called well formed xml document. If xml document satisfies both dtd or xsd rules then it is called Valid xml document. Xml parser is a software application that can check whether xml document is well formed document or not, valid document or not and can also read and process xml document.

Example: SAX parser, DOM parser, DOM4J parser and etc...

- Hibernate internally uses sax parser/DOM4J parser to read and process both hibernate configuration file and hibernate mapping files.

#### **Internal flow of Execution of Hibernate application:**

*Configuration cfg = new Configuration();*

- Activates hibernate framework based on the libraries(jars) that are added to buildpath/ classpath and also tries to locate hibernate.properties file to get hibernate configuration properties.  
`cfg = cfg.configure(".....");`
- Takes the given hibernate configuration file and also get name of mapping file from configuration file.
- Locate and loads both hibernate configuration file.
- Checks whether both xml files are well formed and valid or not using xml parser, If not exception will be thrown.
- Reads hibernate configuration file data and collect other information about current executing environment and creates **in-memory Meta data**( in the memory of the RAM that is used by jvm) and returns object of Configuration class representing that in memory metadata.

**Note:**

- The in-memory meta data of xml file that represents xml data will be used by hibernate framework for multiple times during the execution of the application, this avoids unnecessary loading of both xml files for multiple times.

***SessionFactory factory = cfg.buildSessionFactory();***

- SessionFactory* object having in memory meta data of both xml files that is generated through Configuration object. This SessionFactory object represents the JDBC connection pool that is created based on the hibernate configuration properties of in memory meta data.SessionFactory object is factory to create Session objects

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

- Makes SessionFactory object to create Hibernate session objects by encapsulating the JDBC connection object. This session object is useful to maintain domain class object and to provide persistence instructions to hibernate software.

***tx = ses.beginTransaction() -> begins transaction***

***ses.save(obj)*** → gives persistence instruction to hibernate framework to save the object

***tx.commit()*** → completes the insert operation

**Note:**

***tx.commit() → internally uses con.commit() and tx.rollback() uses con.rollback() methods***

***ses.close()***

- Closes hibernate session object by releasing and cleaning up the JDBC connection object that is associated with Session object and also makes the domain class object as detached object from Session i.e. it stop representing DATABASE table record.

***factory.close();***

- Closes hibernate SessionFactory object by releasing JDBC connection pool that is representing by SessionFactory object.
- It is recommended to close the all session objects before closing SessionFactory object.

**session.save(-) method**

- This method returns hibernate framework generated id value/identifier value for the given domain class object as serializable object, hibernate framework generates this **id value** based on that identifier filed configuration that is done in mapping file through<id> tag.

```
int idVal = (Integer)ses.save(ed); // unboxing
```

**Note:**

- In our example “no” property value will become id value because it is configured in mapping file as identifier using <id> tag.

**Prototype of save():**

```
public serializable save(Object obj)
```

**Q. What is use of hibernate.dialect property in hibernate configuration file?**

**Ans:** It is hibernate property that expects hibernate API supplied sub class name of **org.hibernate.dialect.Dialect class as value**. This value will change based on the database software and its version we use .

Oracle8i ----- org.hibernate.dialect.Oracle8iDialect

Oracle9i----- org.hibernate.dialect.Oracle9iDialect

Oracle10g/11g----- org.hibernate.dialect.Oracle10gDialect

→For more detail go to pdf documentation of hibernate,Most of the times hibernate framework automatically detects and uses dialect but it is recommended to specify explicitly.

**Hibernate framework uses hibernate.dialect property in two ways:**

- To generate optimize(best)SQL query as required for underlying DATABASE software.
- To assign sensible, intelligent default values for the properties that are not specified in hibernate configuration file.

**eg: In hibernate.cfg.xml**

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

**Saving Object using ses.persist(-) method:**

```
//Saving obj using ses.persist(-) method
```

```
EmpDetails ed=new EmpDetails();
ed.setNo(1114); ed.setFname("ravi");
ed.setLname("rao"); ed.setMail("rao2@x.com");
```

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.persist(ed);
    tx.commit();
}
catch(Exception e ){
    tx.rollback();
}
```

**Difference between ses.save(-) and ses.persist(-) method:**

ses.save(-) gives instruction to hibernate to save the object and also returns the generated identity values back to java application where as ses.persist(-) does not return the identity value.

*ses.save(-) method return type is Serializable(I) and ses.persist(-) method return type is void.*

**Note:** ses.save(-) method can use generators to generate the identity/identifier values whereas ses.persist(-) can not use the same generators.

**Prototypes:**

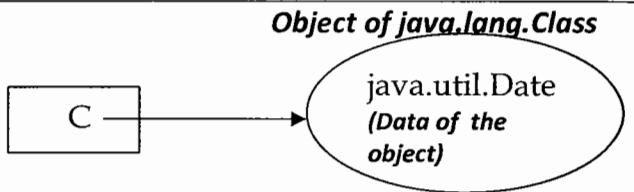
```
public void persist(Object obj);
public Serializable save(Object obj);
```

**Understanding java.lang.Class**

→ In a running java application if you want to represent java class we can use object of java.lang.Class and we can use this object to supply class/interface as the input value to the application. The object of java.lang.Class holds class name and allows getting more details about that class name. There are multiple ways to create object of java.lang.Class

**1. Class.forName(-)**

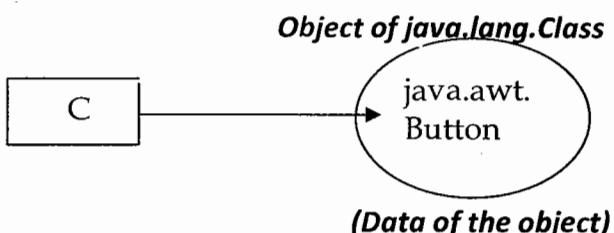
```
Class c = Class.forName("java.util.Date");
```

**2. getClass() of java.lang.Object class**

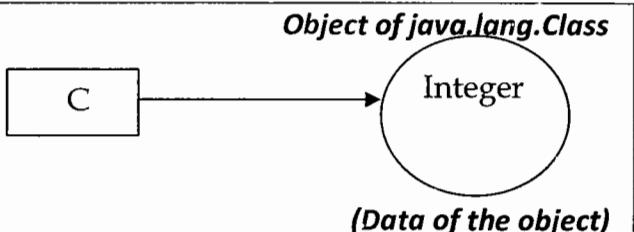
```
getClass() of java.lang.Object class
```

```
Button b = new Button();
```

```
Class c = b.getClass();
```

**3. using class property**

```
Class c = Integer.class
```



In every .class file generated for java class the compiler adds 1 built-in property called **class**, This is a public static property of type java.lang.Class holding the details of the current class.

**Loading object in hibernate:**

We can use

- ses.get(-,-) // (eager/early loading)
- ses.load(-,-)//lazy loading/late loading

**Important points:**

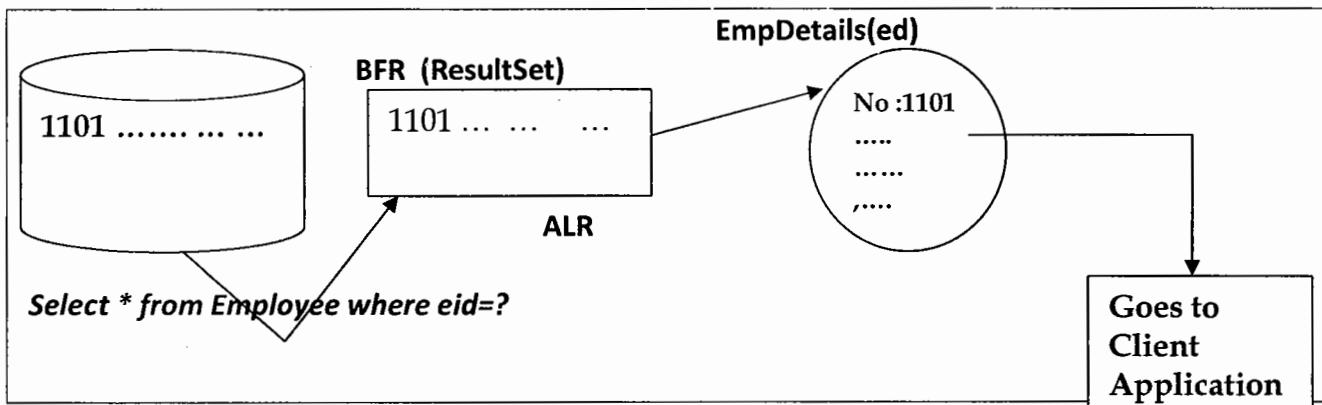
- In hibernate we can perform select persistence operation in non transactional environment .
- All single row persistence operations will be performed by taking identity value as the criteria value.
- Both ses.get(-,-) and ses.load(-,-) methods take domain class name as input value (in the form of java.lang.Class) to create the object of domain class and to store retrieved records into domain class object.

**Prototype:**

```
public object get(Class clazz, Serializable Id);
public object load(Class clazz,Serializable Id);
public <T> get(Class<T> clazz, Serializable Id);
public <T> load(Class<T> clazz, Serializable Id);
```

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

```
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,1101);
if(ed!=null){
    System.out.println(ed.getNo()+" "+ed.getFname() +" "+ed.getLname()+" "+ed.getMail());
}
else{
    System.out.println("record not found");
//close objs
ses.close();
factory.close();
```

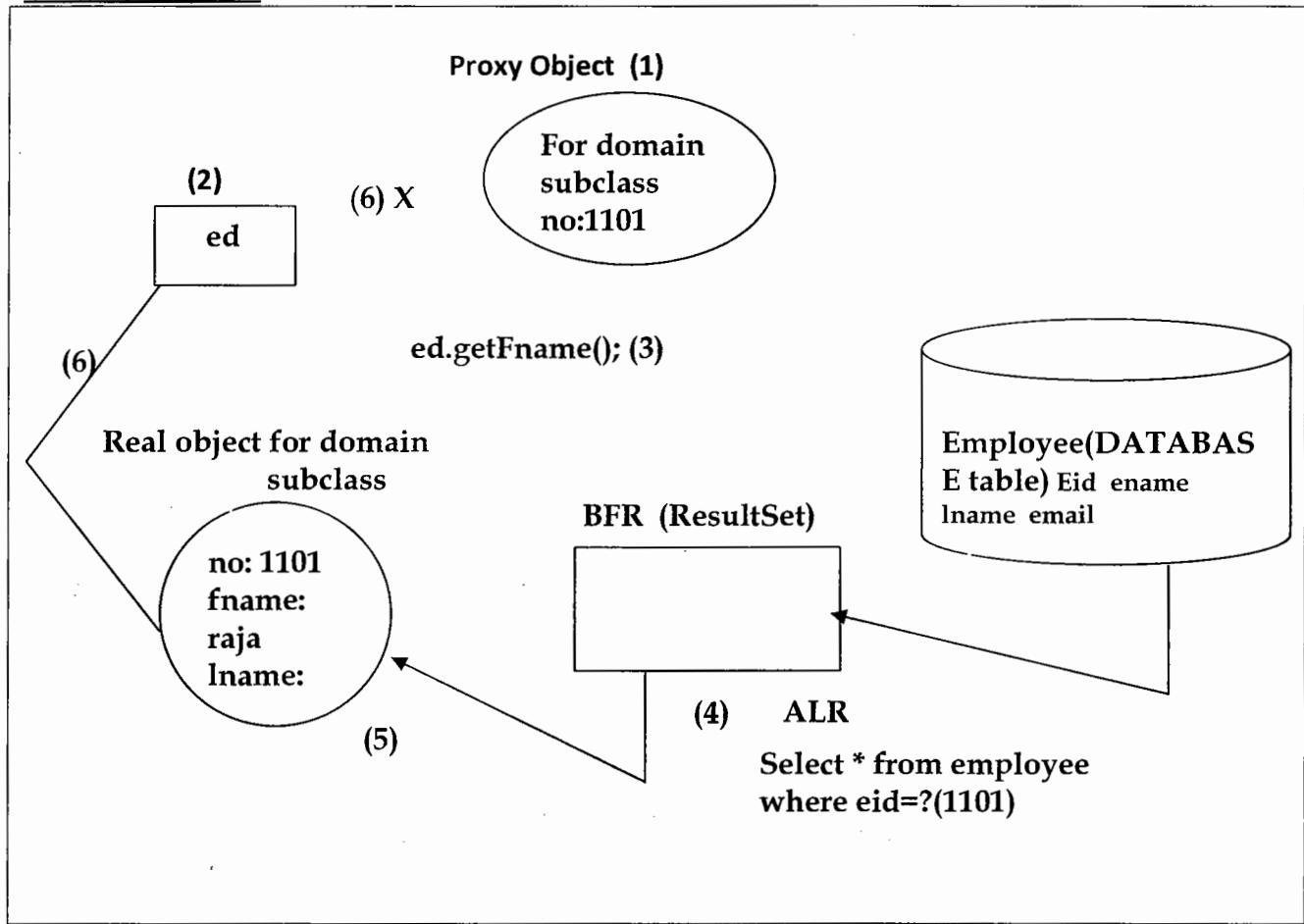
**Memory Diagram:****Loading object using ses.load(-,-) method**

```
EmpDetails ed=(EmpDetails)ses.load(EmpDetails.class,1101);
System.out.println(ed.getNo()+" "+ed.getFname() +" "+ed.getLname()+" "+ed.getMail());
```

- ses.load(-,-) method involves 1 proxy object and 1 real object while loading the record into object.
- The dummy object that represents the presence of real object until real object comes is called as proxy object.
- ses.get(-,-) method performs early/eager loading of the record that means it loads record from Database table to object irrespective of whether that object is used in further part of the application or not.
- ses.load(-,-) method performs lazy loading i.e. hibernate framework retrieves the record from DB table to domain class object only when the utilization of the object is started, till that it gives proxy object to application as shown below.

*EmpDetails ed=(EmpDetails)ses.load(EmpDetails.class,1101);*

#### Memory Diagram:



#### With respect diagram

- Application calls ses.load(-,-) -->hibernate framework creates proxy object for domain subclass having identity value -->That object is referred by the reference variable “ed”.
- Application calls non identifier method on proxy object using that reference variable ed--> But hibernate doesn't complete that method invocation
- Hibernate internally uses JDBC code to execute select query with identity value and gets ResultSet object.
- Hibernate creates real object for domain subclass and copies the record into that object

form ResultSet.

- The reference variable “ed” stops referring proxy object and starts referring to real object
- The non identifier method that invoke earlier Now will be executed on real object

#### Different between ses.get (-,-) and ses.load (-,-) method:

->while working with ses.load (-,-) method the proxy, real objects will not be created from domain class they will be created for domain subclass(proxy class)i.e generated as inmemory class dynamically at run time .

ses.get(-,-)	ses.load(-,-)
<ol style="list-style-type: none"> <li>1. Performs eager loading of object</li> <li>2. Does not generate proxy object</li> <li>3. Involve 1 object of domain class in the loading of record</li> <li>4. Suitable to check whether record is available or not</li> <li>5. Return null value when record is not available</li> <li>6. Useful for immediate and guaranteed utilization of record once the record is loaded</li> <li>7. Useful in single layer environment</li> <li>8. Use case: <ul style="list-style-type: none"> <li>• To show offers of the day without user permission</li> <li>• To show job posting</li> </ul> </li> </ol>	<ol style="list-style-type: none"> <li>1. Performs lazy loading of object</li> <li>2. Generates proxy object</li> <li>3. Involve total 2 object of domain class</li> <li>4. (1 real object + 1 proxy object in the loading of record)</li> <li>5. We should use this method by assuming record is available otherwise it throws ObjectNotFoundException when record is not available</li> <li>6. Useful for delayed or non guaranteed utilization of the record once the record is marked</li> <li>7. Useful in multiple layer environment where record marked in one layer will be utilized in another layer</li> <li>8. Use case: <ul style="list-style-type: none"> <li>• To show personalized offer based on user interests</li> <li>• To show notification related details in FB in which are interested.</li> </ul> </li> </ol>

#### Deleting object (deleting record from Database table that is represented by domain class object)

//Version1: Delete obj using ses.delete(-)

```
EmpDetails ed=new EmpDetails();
ed.setNo(1101);
Transaction tx=null;
```

```

try{
tx=ses.beginTransaction();
ses.delete(ed);
tx.commit();
System.out.println("object is deleted(record deleted)");
}
catch(Exception e){
tx.rollback();
}

```

->Delete object operation does not actually delete the object it just delete the record represented by the object.

**Note:**

ses.delete(-) checks the availability of record and detecting of the record by taking identity value of the given domain object.

```

//Version2: Deleting obj by loading obj
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class, 1001);
if(ed!=null){
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.delete(ed);
tx.commit();
System.out.println("record deleted");
}
catch(Exception e){
tx.rollback();
}
}//if
else{
System.out.println("Record not found");
}

```

->While working with single row operations hibernate perform all its persistence operations by taking identity value as the criteria value.

**Prototype of delete(-)**

public void delete(Object obj);

**Updating object(updating the record that is represented by domain class object)**

**Approach 1:**

```

//Approach1: Update obj
EmpDetails ed=new EmpDetails();
ed.setNo(1002); //existing identifier/id value
ed.setFname("new Raja");
ed.setLname("rao");

```

```

ed.setMail("newRaja@x.com");
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.update(ed);
tx.commit();
}
catch(Exception e){
tx.rollback();
}

```

->In this approach1 the following limitations are there

- a) There is no provision to check whether record is available or not for updation.
- b) We must pass existing identity value and we must check old data to properties which we do not want to modify.

#### **Note:**

The limitation of above approach is we must remember and pass old identity value and other values to the object which we don't want to change. we should pass these value along with the new values.

#### **Approach2:**

```

//Approach2: Load and update the obj
EmpDetails ed=(EmpDetails)ses.load(EmpDetails.class,1002);
if(ed!=null){
ed.setLname("rao1");
ed.setMail("rao_new@x.com");
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.update(ed);
tx.commit();
}
catch(Exception e){
tx.rollback();
}
}

```

In this approach we just need to set new values to the loaded object i.e. there is no need of remembering and assigning old values to the object which we don't want to modify ,so approach2 is good.

#### **Approach3:**

```

//Approach3:updating object
//load object
Details=ses.get(EmpDetails.class,1002);
If(ed!=null){
try{
tx=ses.beginTransaction();
ed.setMail("x@z1.com");
}

```

```

tx.commit();
sopln("object is updated");
}
Catch(Exception e){
Tx.rollback();
}
}//if
else{
sopln("object not found to update");
}

```

In this approach if you modify loaded object in a transaction then the modification done on the object will reflect to database table row through synchronization when transaction is committed.

#### Prototype of update(-) method:

```
public void update(Object obj);
```

#### Saving or updating object: (use ses.saveOrUpdate(-))

```

EmpDetails ed=new EmpDetails();
ed.setNo(3454);
ed.setFname("ramesh");
ed.setLname("rao");
ed.setMail("rao21@1234.com");
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.saveOrUpdate(ed);
tx.commit();
}
catch(Exception e){
tx.rollback();
}

```

This method takes given object identity value as the criteria value and check the availability of record in Database table by generating select query. if available, it will update the record, otherwise it will insert the record.

#### Use case:

Collecting information from customers along with phone numbers, if record is already available then it will update the details otherwise it will insert the record.

#### Prototype:

```
public void saveOrUpdate(Object obj);
```

#### Saving or updating object using ses.merge(-):

```

EmpDetails ed=new EmpDetails();
ed.setNo(3454);
ed.setFname("ramesh");
ed.setLname("rao");
ed.setMail("rao21@1234.com");
Transaction tx=null;
try{
tx=ses.beginTransaction();
EmpDetails ed1=(EmpDetails)ses.merge(ed);
System.out.println(ed1);
tx.commit();
}
catch(Exception e){
tx.rollback();
}

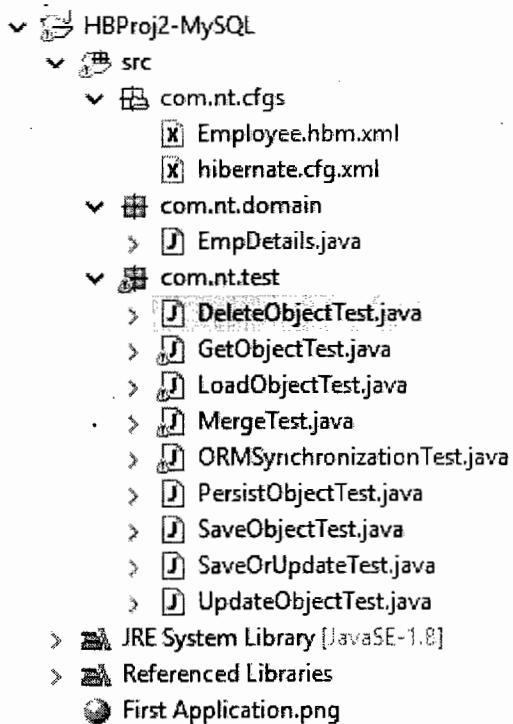
```

**Note:**

merge(-) method returns domain class object representing the record that has to be inserted or updated.

**Prototype:**

```
public Object merge(Object obj);
```

**Employee.hbm.xml:-**

```

<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
```

```
<!-- O-R mapping cfgs -->
<class name="com.nt.domain.EmpDetails" table="Employee">
  <id name="no" column="EID"/>

  <property name="fname">
    <column name="FIRSTNAME"/>
  </property>
  <property name="lname" column="LASTNAME"/>
  <property name="mail" column="EMAIL"/>
</class>
</hibernate-mapping>
```

***hibernate.cfg.xml:-***

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <!-- connection properties -->
    <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql://nths98db</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>
    <!-- Dialect cfg -->
    <property
name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
    <!-- mapping file -->
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

***EmpDetails.java:-***

```
package com.nt.domain;

public class EmpDetails {
  private int no;
  private String fname;
  private String lname;
  private String mail;

  public EmpDetails() {
    System.out.println("EmpDetails:0-param constructor");
  }

  public int getNo() {
    return no;
  }
}
```

```
}

public void setNo(int no) {
    this.no = no;
}

public String getFname() {
    return fname;
}

public void setFname(String fname) {
    this.fname = fname;
}

public String getLname() {
    return lname;
}

public void setLname(String lname) {
    this.lname = lname;
}

public String getMail() {
    return mail;
}

public void setMail(String mail) {
    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ",
    lname=" + lname + ", mail=" + mail + "]";
}

}
```

**DeleteObjectTest.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.domain.EmpDetails;

public class DeleteObjectTest {
    public static void main(String[] args) {
        Session ses=null;
        SessionFactory factory=null;
        EmpDetails details=null;
        Transaction tx=null;
        //create Session object
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
        buildSessionFactory();
```

```
ses=factory.openSession();
    /**delete Object (Version1)
details=new EmpDetails();
    details.setNo(1001);
try{
    tx=ses.beginTransaction();
    ses.delete(details);
    tx.commit();
    System.out.println("Object deleted");
}
catch(Exception e){
    tx.rollback();
}*/



//delete object (version2)
//load object
details=ses.get(EmpDetails.class,1002);
if(details!=null){
try{
    tx=ses.beginTransaction();
    ses.delete(details);
    tx.commit();
    System.out.println("Object deleted");
}
catch(Exception e){
    tx.rollback();
}
}
else{
    System.out.println("Object is not deleted");
}

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

**GetObjectTest.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.domain.EmpDetails;

public class GetObjectTest {

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

```
Configuration cfg=null;
SessionFactory factory=null;
Session ses=null;
EmpDetails details=null;
Transaction tx=null;
//Activate HB framework based on jars added to build path
cfg=new Configuration();
//read both xml files
cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
//build SessionFactory
factory=cfg.buildSessionFactory();
//create Session
ses=factory.openSession();

//Load object
details=ses.get(EmpDetails.class,1071);
//details.getFname();
if(details!=null)
    System.out.println(details.getNo()+
"+details.getFname()+" "+details.getLname()+" "+details.getMail());
else
    System.out.println("Record not found");

//close objs
ses.close();
factory.close();
}//main
}//class
```

***LoadObjectTest.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.domain.EmpDetails;

public class LoadObjectTest {

    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //read both xml files
```

```

cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
//build SessionFactory
factory=cfg.buildSessionFactory();
//create Session
ses=factory.openSession();

//Load object
details=ses.load(EmpDetails.class,121);
if(details!=null)
    System.out.println(details.getNo()+""
"+details.getFname()+" "+details.getLname()+" "+details.getMail());
else
    System.out.println("Record not found");

//close objs
ses.close();
factory.close();
}//main
}//class

```

**MergeTest.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.domain.EmpDetails;

public class MergeTest {

    public static void main(String[] args) {
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null,details1=null;
        Transaction tx=null;
        //create SessionFactory object
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        //create Session
        ses=factory.openSession();
        //prepare object
        details=new EmpDetails();
        details.setNo(9867);
        details.setFname("ramesh3");
        details.setLname("rao1");
        details.setMail("rao@gmail.com");
        try{
            tx=ses.beginTransaction();

```

```
details1=(EmpDetails)ses.merge(details);
tx.commit();
System.out.println("Object is saved/updated");
}//try
catch(Exception e){
    tx.rollback();
}
//close objs
ses.close();
factory.close();
}//main
}//class
```

***ORMSynchronizationTest.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.domain.EmpDetails;

public class ORMSynchronizationTest {
    public static void main(String[] args) {
        Session ses=null;
        SessionFactory factory=null;
        EmpDetails details=null;
        Transaction tx=null;
        //create Session object
        factory=new Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        ses=factory.openSession();
        /* //Syncrhronization between Object to Db table row
           //load object
        details=ses.get(EmpDetails.class,9867);
        //modify the object in a Transaction
        try{
            tx=ses.beginTransaction();
            details.setMail("x@y1.com");
            tx.commit();
            System.out.println("Object is modified");
        }//try
        catch(Exception e){
            tx.rollback();
        }*/
        //Syncrhronization between Db table row to object
        //load object
```

```

        details=ses.get(EmpDetails.class,9867);
        System.out.println(details);
        try{
            Thread.sleep(40000); //Modify DB table data from SQL prompt
        }
        catch(Exception e){
            e.printStackTrace();
        }
        ses.refresh(details); //reloading object
        System.out.println(details);

        //close objs
        ses.close();
        factory.close();
    } //main
} //class

```

**PersistObjectTest.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.domain.EmpDetails;

public class PersistObjectTest {

    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //read both xml files
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");

        //build SessionFactory
        factory=cfg.buildSessionFactory();
        System.out.println("SessionFactory obj class
                           "+factory.getClass());

        //create Session
        ses=factory.openSession();
        System.out.println("session object class::"+ses.getClass());
        //create Domain class object
        details=new EmpDetails();
    }
}

```

```
details.setNo(1060); details.setFname("raja2");
details.setLname("rao2"); details.setMail("rao2@gmail.com");

//Persist object..
try{
    tx=ses.beginTransaction(); //begin Tx
    ses.persist(details);
    tx.commit(); //commit Tx
    System.out.println("Object is Persisted");
} //try
catch(Exception e){
    tx.rollback(); //rollback Tx
    e.printStackTrace();
}
//close objs
ses.close();
factory.close();
}//main
}//class
```

**SaveObjectTest.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.domain.EmpDetails;

public class SaveObjectTest {

    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //read both xml files
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");

        //build SessionFactory
        factory=cfg.buildSessionFactory();
        System.out.println("SessionFactory obj class
                           "+factory.getClass());
        //create Session
        ses=factory.openSession();
        System.out.println("session object class::"+ses.getClass());
        //create Domain class object
```

```
details=new EmpDetails();
details.setNo(1071); details.setFname("raja2");
details.setLname("rao2"); details.setMail("rao2@gmail.com");

//Save object..
try{
    tx=ses.beginTransaction(); //begin Tx
    int idval=(Integer)ses.save(details);
    System.out.println("Generated id value:::"+idval);
    tx.commit(); //commit Tx
    System.out.println("Object is saved");
} //try
catch(Exception e){
    tx.rollback(); //rollback Tx
}
//close objs
ses.close();
factory.close();
}//main
}//class
```

**SaveOrUpdateTest.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.domain.EmpDetails;

public class SaveOrUpdateTest {

    public static void main(String[] args) {
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        //create SessionFactory object
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
                buildSessionFactory();

        //create Session
        ses=factory.openSession();
        //prepare object
        details=new EmpDetails();
        details.setNo(9887);
        details.setFname("ramesh2");
        details.setLname("rao1");
        details.setMail("rao@gmail.com");
```

```
try{
    tx=ses.beginTransaction();
    ses.saveOrUpdate(details);
    tx.commit();
    System.out.println("Object is saved/updated");
} //try
catch(Exception e){
    tx.rollback();
}
//close objs
ses.close();
factory.close();
}//main
}//class
```

***UpdateObjectTest.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.domain.EmpDetails;

public class UpdateObjectTest {

    public static void main(String[] args) {
        Session ses=null;
        SessionFactory factory=null;
        EmpDetails details=null;
        Transaction tx=null;
        //create Session object
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
        buildSessionFactory();

        ses=factory.openSession();
        //Update object (version1)-->Bad approacj
        details=new EmpDetails();
        details.setNo(56778);
        details.setFname("raja2");
        details.setLname("rao2");
        details.setMail("raja2@coolMail.com");
        try{
            tx=ses.beginTransaction();
            ses.update(details);
            tx.commit();
            System.out.println("Object updated");
        }
        catch(Exception e){
```

```
        tx.rollback();
    }

/* //Updating object (version2)-->Good pratice
   //load object
details=ses.get(EmpDetails.class,9877);
if(details!=null){
    try{
        tx=ses.beginTransaction();
        details.setFname("raja");
        details.setLname("rao");
        details.setMail("x@z.com");
        ses.update(details);
        tx.commit();
        System.out.println("Object is updated");
    } //try
    catch(Exception e){
        tx.rollback();
    }
} //if
else{
    System.out.println("Object not found to update");
}
*/
/* //Updating object (version3)-->Good pratice
//load object
details=ses.get(EmpDetails.class,1003);
if(details!=null){
    try{
        tx=ses.beginTransaction();
        details.setMail("x@z1.com");
        tx.commit();
        System.out.println("Object is updated");
    } //try
    catch(Exception e){
        tx.rollback();
    }
} //if
else{
    System.out.println("Object not found to update");
} */

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

Difference between ses.update(-), ses.saveOrUpdate(-) and ses.merge(-):

ses.update(-)	ses.saveOrUpdate(-)	ses.merge(-)
<b>Prototype:</b> <b>public void update(Object obj);</b>  1. Performs object updation/record updation. 1. Updates the record without checking the availability of record  2. Generates only update query  3. Generates update query even through given object data matching with the record available in the table  4. Does not return any object representing the record that has to be updated	<b>Prototype:</b> <b>public void save or update(Object obj);</b>  1. Performs save/insert or update operation on record 2. Inserts or updates the record by checking the availability of the record.  3. Generates select query and insert/update query 4. Does not generate update query  5. Same as ses.update(-)	<b>Prototype:</b> <b>public Object merge(Object obj);</b>  1. Same as ses.saveOrUpdate(-)  2. Same as ses.saveOrUpdate(-)  3. Same as ses.saveOrUpdate(-)  4. Same as saveOrUpdate(-)  5. Returns object of the record that is representing to be updated or inserted

**Q. How can we perform synchronization from object to Database table row?**

**Or**

**How can we update the object/record without ses.update(-) or ses.saveOrUpdate(-) or ses.merge(-)?**

**Ans:** Load the object from Database table and modify the object in a Transaction mode. when Transaction is committed all the modifications done in the object will be synchronized to DATABASE table record by generating update query.

```

EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,1002);
if(ed!=null){
  Transaction tx=null;
  try{
    tx=ses.beginTransaction();
    ed.setMail("rao_new@x.com");
    tx.commit();
  }
  catch(Exception e){
    tx.rollback();
  }
  else{
    System.out.println("Record not found");
  }
}

```

**Showing Database table record to object synchronizations:**

```
// Showing DATABASE table record to Object Synchronization
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class, 1002);
System.out.println("Before Refresh"+ed);
Thread.sleep(30000); //During this perio modify DATABASE table record
ses.refresh(ed);
System.out.println("After Refresh"+ed);
```

**Note1:**

For object to Database table record synchronization, modify loaded object in a Transaction and commit that Transcation or modify loaded object and flush the session using ses.refresh() method.

**Note2:**

Flushing is the process of writing pending changes that are there in domain object of ses to underlying Database table record.

The synchronized DATABASE table record with already loaded object of Hibernate session call ses.refresh() method.

**Q. can we develop hibernate application without hibernate configuration file (xml file)?**

Ans: yes

- It is possible by using hibernate.properties file but not recommended to use.
- By setting hibernate property values programmatically to the application.(not recommended to use)

**Using hibernate.properties file:**

When we activate hibernate framework by using configuration object it looks for hibernate.properties file having configuration properties.

eg:

- remove hibernate.cfg.xml from the project
- Add hibernate.properties to the src folder

**hibernate.properties:**

```
#hibernate properties
connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=scott1
hibernate.connection.password=tiger
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
```

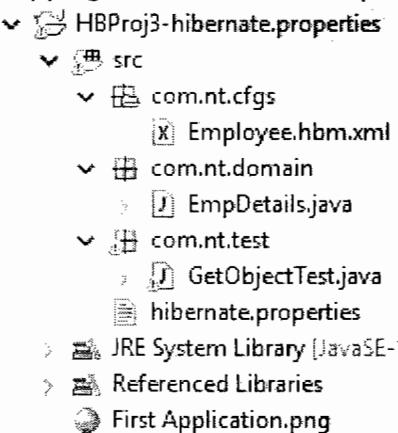
**Add following code in client application to create Hibernate Session object**

```
Configuration cfg=new Configuration();
cfg.addResource("com/nt/cfgs/Employee.hbm.xml");
SessionFactory factory=cfg.buildSessionFactory();
Session ses=factory.openSession();
```

**Limitations:**

- It is not a industry standard
- It can't be placed in required package

- The word hibernate is not optional in the property name
- Mapping file name must be specified programmatically.

**Employee.hbm.xml**

```

<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <!-- O-R mapping cfgs -->
  <class name="com.nt.domain.EmpDetails" table="Employee">
    <id name="no" column="EID"/>

    <property name="fname">
      <column name="FIRSTNAME"/>
    </property>
    <property name="lname" column="LASTNAME"/>
    <property name="mail" column="EMAIL"/>
  </class>
</hibernate-mapping>
  
```

**EmpDetails.java:-**

```

package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
}
  
```

```

    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }

    @Override
    public String toString() {
        return "EmpDetails [no=" + no + ", fname=" + fname + ",
lname=" + lname + ", mail=" + mail + "]";
    }
}

```

- GetObjectTest.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.domain.EmpDetails;

public class GetObjectTest {
    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //add mapping file
        cfg.addFile("src/com/nt/cfgs/Employee.hbm.xml");
        //build SessionFactory
        factory=cfg.buildSessionFactory();
        //create Session
    }
}

```

```

ses=factory.openSession();

//Load object
details=ses.get(EmpDetails.class,9867);
if(details!=null)
    System.out.println(details.getNo()+""
"+details.getFname()+" "+details.getLname()+" "+details.getMail());
else
    System.out.println("Record not found");

//close objs
ses.close();
factory.close();
}//main
}//class

```

**hibernate.properties:-**

```

#Connection properties
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=system
hibernate.connection.password=manager
#hibernate properties
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect

```

**Programmatic approach:**

Set hibernate configuration property names and values to configuration object directly in the client application by calling set property method.

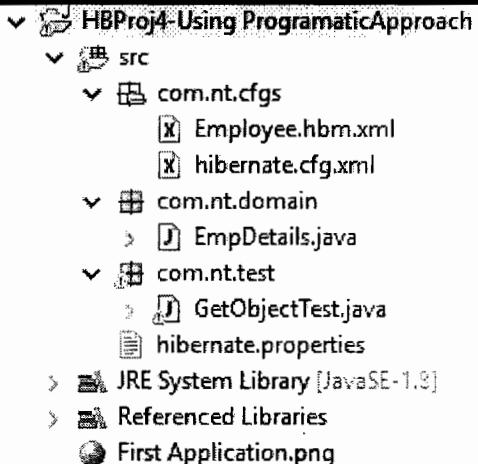
eg:

- Remove hibernate.cfg.xml ,hibernate.properties from the application in ClientApp
- Add following code in ClientApp

```

//Activate HB F/w
Configuration cfg=new Configuration();
//set Cfg properties
cfg.setProperty("hibernate.connection.driver_class","oracle.jdbc.driver.OracleDriver");
cfg.setProperty("hibernate.connection.url","jdbc:oracle:thin:@localhost:1521:xe");
cfg.setProperty("hibernate.connection.username","scott");
cfg.setProperty("hibernate.connection.password","tiger");
//set mapping file
cfg.addResource("com/nt/cfgs/Employee.hbm.xml");
//Create SessionFactory obj
SessionFactory factory=cfg.buildSessionFactory();
//Open Session
Session ses=factory.openSession();

```

**Employee.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- O-R mapping cfgs -->
    <class name="com.nt.domain.EmpDetails" table="Employee">
        <id name="no" column="EID"/>

        <property name="fname">
            <column name="FIRSTNAME"/>
        </property>
        <property name="Lname" column="LASTNAME"/>
        <property name="mail" column="EMAIL"/>
    </class>
</hibernate-mapping>
```

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- connection 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>
        <!-- Dialect cfg -->
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <!-- mapping file -->
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**EmpDetails.java:-**

```
package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
}
```

```
public String getLname() {
    return lname;
}
public void setLname(String lname) {
    this.lname = lname;
}
public String getMail() {
    return mail;
}
public void setMail(String mail) {
    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ",
lname=" + lname + ", mail=" + mail + "]";
}

}
```

**GetObjectTest.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.domain.EmpDetails;

public class GetObjectTest {

    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;

        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //use hibernate.cfg.xml file
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");

        //set hibernate connection,mapping file info
        cfg.setProperty("hibernate.connection.driver_class",
                        "oracle.jdbc.driver.OracleDriver");
        cfg.setProperty("hibernate.connection.url",
                        "jdbc:oracle:thin:@localhost:1521:xe");
        cfg.setProperty("hibernate.connection.username","system");
    }
}
```

```

cfg.setProperty("hibernate.connection.password","manager1");
cfg.setProperty("hibernate.dialect",
                "org.hibernate.dialect.Oracle10gDialect");
cfg.setProperty("hibernate.show_sql","true");
cfg.setProperty("hibernate.format_sql","true");
//add mapping file
cfg.addFile("src/com/nt/cfgs/Employee.hbm.xml");
//build SessionFactory
factory=cfg.buildSessionFactory();
//create Session
ses=factory.openSession();
//get Object
details=ses.get(EmpDetails.class,9867);
if(details!=null)

System.out.println(details);

//close objs
ses.close();
factory.close();
}//main

```

**hibernate.properties:-**

```

#Connection properties
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=system
hibernate.connection.password=manager1
#hibernate properties
hibernate.show_sql=true
hibernate.format_sql=true
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect

```

**Limitations:**

- Kills the flexibility of modification.
- Hard coding of **configuration properties** is bad practice.
- The word **hibernate** is not optional in some property names.
- Mapping file name is **hardcoded** here.
- It is not industry standard.

**Conclusion:**

Working with hibernate configuration file (xml) is the best practice.

**Q. if you use all the 3 approaches to supplied hibernate configuration properties to the application can you tell what will be applied finally?**

**Ans:** If `cfg.setProperty(-,-)` methods are called after `cfg.configure()` then programmatic approach settings will take place otherwise the settings done in `hibernate.cfg.xml` file will take place.

**Q. can we develop hibernate application without mapping file?**

**Ans:** Possible by using mapping annotations. We need to add mapping annotations in the domain class without taking hibernate mapping file.

### Understanding the hibernate.hbm2ddl.auto property:

This is hibernate configuration property of hibernate configuration file ,this is useful to create/alter/validate DATABASE table according to hibernate mapping file configuration.

This property allows 4 types of possible values:

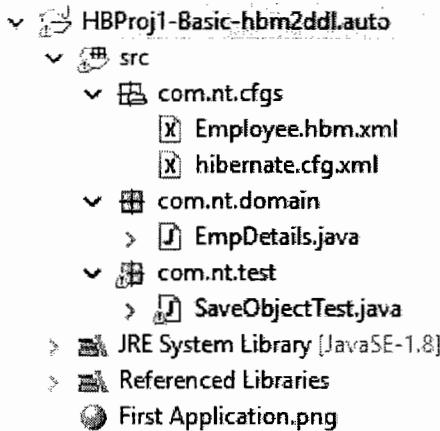
1. create
2. update
3. validate (default)
4. create-drop

#### create:

- Always crate new DATABASE tables based on mapping file configuration. If needed drops the existing Database tables it collect Database table names to be created from the hibernate mapping file
- We can control column size , constraint by using not-null, unique, length attribute of <property> tag
- It internally uses **SchemaExport** tool to perform all these operations.

#### Example:

- Keep any hibernate application ready.
  - Add the following property in hibernate configuration file.
- ```
<property name="hbm2ddl.auto">create</property>
```
- Add length, not-null, unique and etc attributes in <property> tag of mapping file.
- ```
<property name="mail" column="EMAIL" length="20" unique="true" not-null="true"/>
```
- Run the application.



#### Employee.hbm.xml:-

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- O-R mapping cfgs -->
    <class name="com.nt.domain.EmpDetails" table="Employee">
        <id name="no" column="EID"/>
        <property name="fname" type="string">
            <column name="FIRSTNAME" length="20" not-null="true" />
        </property>
    </class>
</hibernate-mapping>
```

```

        </property>
    <property name="Lname" column="LASTNAME" type="string" length="20"
not-null="true"/>
    <property name="mail" column="EMAIL" type="string" length="20"
unique="true"/>
</class>
</hibernate-mapping>

```

**hibernate.cfg.xml:-**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- connection 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>
        <!-- <property name="hibernate.hbm2ddl.auto">create</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property name="hibernate.hbm2ddl.auto">validate</property>
-->
        <property name="hibernate.hbm2ddl.auto">create-drop</property>
        <!-- Dialect cfg -->
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <!-- mapping file -->
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

**EmpDetails.java:-**

```

//EmpDetails.java (BO class/Domain class/Entity class/Persistence
class/Model class)
package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;
}

```

```
public EmpDetails() {
    System.out.println("EmpDetails:0-param constructor");
}

public int getNo() {
    return no;
}
public void setNo(int no) {
    this.no = no;
}
public String getFname() {
    return fname;
}
public void setFname(String fname) {
    this.fname = fname;
}
public String getLname() {
    return lname;
}
public void setLname(String lname) {
    this.lname = lname;
}
public String getMail() {
    return mail;
}
public void setMail(String mail) {
    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ",
    lname=" + lname + ", mail=" + mail + "]";
}

}
```

**SaveObjectTest.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.domain.EmpDetails;

public class SaveObjectTest {

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

```

Configuration cfg=null;
SessionFactory factory=null;
Session ses=null;
EmpDetails details=null;
Transaction tx=null;
//Activate HB framework based on jars added to build path
cfg=new Configuration();
//read both xml files
cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");

//build SessionFactory
factory=cfg.buildSessionFactory();

//create Session
ses=factory.openSession();
//create Domain class object
details=new EmpDetails();
details.setNo(1079); details.setFname("raja3");
details.setLname("rao3"); details.setMail("rao9@gmail.com");
//Save object..
try{
    Thread.sleep(30000);
    tx=ses.beginTransaction(); //begin Tx
    int idval=(Integer)ses.save(details);
    tx.commit(); //commit Tx
    System.out.println("Object is saved");
} //try
catch(Exception e){
    tx.rollback(); //rollback Tx
}
//close objs
ses.close();
factory.close();
}//main
}//class

```

**Note**

Do not use this property to create DATABASE table of the project as part of project development. But this property can be used when DATABASE software of the project changed to new DATABASE software.

**Use case:**

In Indian railway, the details of passenger whose tickets are booked will be deleted after every 3 years, in this situation to empty the tables data we can run the application having **create** value for **hbm2ddl.auto** property.

**update:**

- This value of **hbm2ddl.auto** property can use the existing table or can create the new DATABASE table or can alter the existing DATABASE table by adding new columns to

Database table as additional properties are configure in the mapping file.

- This internally uses **SchemaUpdate** tool. This is most recommended value to use in real-time.
- To make hibernate framework creating new column while working with this value, we need add new property in domain class and configure that property in the hibernate mapping file.

#### Example:

- **In hibernate.cfg.xml**

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

- Add additional property in domain class EmpDetails.java having setter and getter methods

```
private String addrs;
public String getAddrs() {
    return addrs;
}
```

```
public void setAddrs(String addrs) {
    this.addrs = addrs;
}
```

- Configure the above property in mapping file

```
<property name="addrs" column="addrs" length="20" />
```

#### validate:

it is the default value of **hbm2.ddl.auto** property, if no value is specified. it makes hibernate framework to verify/validate whether Database table is available according to hibernate mapping file configuration or not, if not available then Exception will be raised.

#### Example:

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

#### Note:

→This value internally uses **SchemaValidator** tool.

#### create-drop:

This value creates the Database table at beginning of the application when hibernate SessionFactory object is created and uses the Database table during the application execution and drops the Database table at the end of application execution once factory.close() method is called. This value is very useful to create and drop table.

#### Use case:

→To create and drop DATABASE tables in the UAT (User Acceptance Test).

The test that happens in front of client after installing the project at the client organization is called UAT.

→One the UAT is over the project goes for production, during UAT Database tables should created temporarily and should be dropped at the end of UAT, so for this create-drop option is good.

This value internally uses **SchemaExport** tool.

**Example:**

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

**Understating ServiceRegistry in Hibernate Framework:****ServiceRegistry:-**

Service is a plugable unit to provide extra functionality to the existing code. Service registry is responsible to hold, manage, providing access to services:

Service registries are hierarchical that means one service of service registry can use another service of same registry (or) parent registry.

To create service registry we need service registry builder.

→ In hibernate 4.x ServiceRegistry is introduced which is a registry that can hold manage and provide access to services both built-in and custom services.

→ A **ServiceRegistry** can manage the following services like Batchbuilder service, Configuration service, Connection Provider Service, Dialect Resolver service and jdbc, Transaction, JNDI services.

→ Every ServicesRegistry is an object of a class that implements org.hibernate.service **ServiceRegistry** interface.

eg: BootStrapServiceRegistry, StandardServiceRegistry and etc...

→ To create ServiceRegistry we need ServiceRegistryBuilder class object, There are multiple ServiceRegistryBuilder in hibernate 4.x thet are:

- StandardServiceRegistryBuilder(to build StandardServiceRegistry)
- BootStrapServiceRegistryBuilder(to build BootStrapServiceRegistry)

→ In hibernate 4.x cfg.buildSessionFactory(-) method is deprecated and alternate method is cfg.buildSessionFactroy(ServiceRegistry) method.

→ It recommended to use following code in hibernate 4.x to SessionFactory Session objects.

```
//activate hibernate framework
Configuration cfg= cfg=new Configuration();
//configure hibernate cfg and mapping file
cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
//create ServiceRegistryBuilder
```

```
StandardServiceRegistryBuilder builder = new StandardServiceRegistryBuilder();
//create ServiceRegistry
ServiceRegistry registry= registry=builder.applySettings(cfg.getProperties()).build();
//Create SessionFactory obj
SessionFactory factory=cfg.buildSessionFactory(registry);
//create session obj
Session ses = factory.openSession();
```

**DAO (Data access object):**

- It is a design pattern having the ability to separate persistence logic from other logics of the application. This pattern makes the persistence logic as the reusable logic and flexible logic to modify.
- Design patterns are the set of the rules that come as best solutions for reoccurring problems of application development.
- Design patterns are the best practices to work with software technologies, frameworks effectively.  
eg: singleton design pattern, dao pattern and etc...

**Every DAO pattern contains 3 resources:**

1. DAO interface(contains the declaration of persistence methods)
2. DAOImpl class(contains persistence logic)
3. DAOFactory(returns one of the several DAOImpl class object)

**Example:**

EmployeeDAO interface  
EmployeeDAOImplementation  
EmployeeDAOFactory class

=>DAO pattern method take BO class object as the params(if inputs are more than 3) and returns BO class object(if outputs are more than3) as the return value.

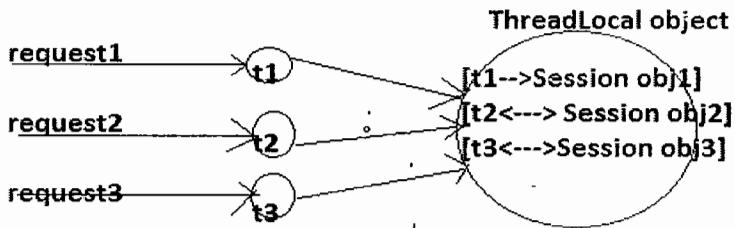
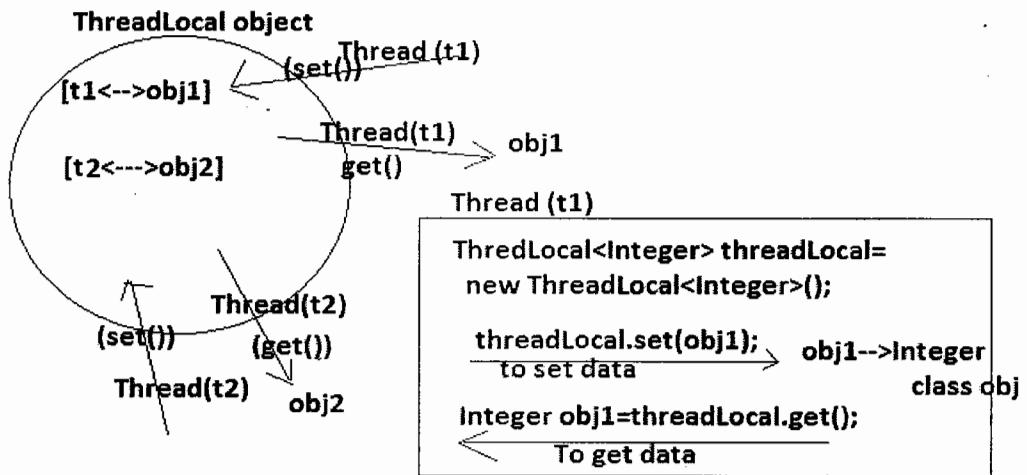
**DAO impl class contains 2 parts:**

1. Query part
2. Code part

**In real time 2 strategies will be taken while creating DAO classes:**

1. 1 DAO per DATABASE table (if project contains <100 table)
  2. 1 DAO per related DATABASE tables(if project contains>100 table)
- All immutable objects are thread safe objects by default, because even though multiple threads are started on that objects the data of the objects can't be manipulated.
  - SessionFactory object is thread safe object by default because it is thread safe object we generally take sessionFactory object in the project one per project and database basis.
  - Hibernate Session object is not thread safe because it is mutable object. We generally take session object one per client or one per request and we keep that object in ThreadLocal to make them specific to each Thread.
  - ThreadLocal is a predefined class that is introduced from jdk1.4 version to maintain one object per thread(internally uses HashMap having thread names as the keys and objects to be maintained as the values)
  - It does not allow the object set by one thread getting accessed by another thread.

- ThreadLocal is very useful to make the object as Thread Safe even though object is not immutable and without using Synchronization



In real time the logic of creating SessionFactory , Session object will be placed in separate class called HibernateUtility class to make that logic visible to multiple DAO classes or client Applications.

#### In java we can create variable in 4 scopes:

1. Global scope/instance scope
  2. Class scope(static)
  3. Local scope
  4. Thread scope(ThreadLocal)
- Thread scope variables are placed in `ThreadLocal` on one per thread basis. object that is specific to one thread can't be accessed by another thread. In `ThreadLocal` we can place only one object per thread at a time.

DAO class contains persistence logic, Service class contains business logic and controller contains Integration logic to receive the requests from client and to delegate the requests to service class. In realtime sessionfactory object will be created on one per database and one per application where as session objects will be created on one per client application/thread basis by taking support of `ThreadLocal`.

**Hibernate Application performing all CURD operation using DAO,HibernateUtil class:**

HBProj2(DAO+HibernateUtil.java+CURD Operations)

- src
  - com.nt.cfgs
    - Employee.hbm.xml
    - hibernate.cfg.xml
  - com.nt.dao
    - EmployeeDAO.java
    - EmployeeDAOFactory.java
    - EmployeeDAOImpl.java
  - com.nt.domain
    - EmpDetails.java
  - com.nt.test
    - ClientApp.java
  - com.nt.utility
    - HibernateUtil.java
- JRE System Library [JavaSE-1.8]
- ojdbc14.jar - C:\oracle\app\oracle\product\10.2.0\
- HibernateLIB

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<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">scott</property>
<property name="connection.password">tiger</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
<id name="no" column="EID"/><!-- Singular Id field -->
<property name="fname" column="FIRSTNAME" length="20" not-null="true"/>
<property name="lname" column="LASTNAME" length="20"/>
<property name="mail" column="EMAIL" length="20" unique="true"/>
</class>
</hibernate-mapping>
```

EmpDetails.java

```
package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }
}
```

```
@Override  
public String toString(){  
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

### HibernateUtil.java

```
package com.nt.utility;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
public class HibernateUtil {  
private static ThreadLocal<Session> threadLocal=new ThreadLocal<Session>();  
private static SessionFactory factory;  
  
static{  
Configuration cfg=null;  
//Create SessionFactory obj  
cfg=new Configuration();  
cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");  
factory=cfg.buildSessionFactory();  
}  
  
public static Session getSession(){  
  
Session session=null;  
if(threadLocal.get()==null){  
    session=factory.openSession();  
    threadLocal.set(session);  
}  
session=threadLocal.get();  
return session;  
}//getSession()  
  
public static void closeSession(){  
Session session=null;  
session=threadLocal.get();  
session.close();  
threadLocal.remove();  
}  
public static void closeSessionFactory(){  
factory.close();  
}
```

**EmployeeDAO.java**

```
package com.nt.dao;

import com.nt.domain.EmpDetails;

public interface EmployeeDAO {
    public int register(int no, String fname, String lname, String mail);
    public void modifyEmail(int no, String newEmail);
    public EmpDetails search(int no);
    public void remove(int no);
}
```

**EmployeeDAOImpl.java**

```
package com.nt.dao;

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

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class EmployeeDAOImpl implements EmployeeDAO {

    @Override
    public int register(int no, String fname, String lname, String mail) {
        Session session=null;
        Transaction tx=null;
        EmpDetails details=null;
        int id=0;
        // get HB Session
        session=HibernateUtil.getSession();
        //save obj
        details=new EmpDetails();
        details.setNo(no); details.setFname(fname); details.setLname(lname);
        details.setMail(mail);
        try{
            tx=session.beginTransaction();
            id=(Integer)session.save(details);
            tx.commit();
        }catch(Exception e){
            tx.rollback();
        }
        return id;
    }

    @Override
    public void modifyEmail(int no, String newEmail) {
```

```
Session session=null;
EmpDetails details=null;
Transaction tx=null;
//get Session
session=HibernateUtil.getSession();
//Load object
details=search(no);

if(details!=null){
//Update object
details.setMail(newEmail);
try{
tx=session.beginTransaction();
session.update(details);
tx.commit();
}
catch(Exception e){
tx.rollback();
}
}
}//if
}//method

@Override
public EmpDetails search(int no) {
Session session=null;
EmpDetails details=null;
// get Session
session=HibernateUtil.getSession();
//Load obj
details=(EmpDetails)session.get(EmpDetails.class,no );
return details;
}

@Override
public void remove(int no) {
Session session=null;
EmpDetails details=null;
Transaction tx=null;
//get Session
session=HibernateUtil.getSession();
//Load object
details=search(no);
//Delete obj
if(details!=null){
try{
tx=session.beginTransaction();
session.delete(details);
tx.commit();
}
}
```

```
//try  
catch(Exception e){  
tx.rollback();  
}  
//if  
//method  
//class
```

### EmployeeDAOFactory.java

```
package com.nt.dao;  
  
public class EmployeeDAOFactory {  
  
    public static EmployeeDAO getInstance(){  
        return new EmployeeDAOImpl();  
    }  
}
```

### ClientApp.java

```
package com.nt.test;  
  
import com.nt.dao.EmployeeDAO;  
import com.nt.dao.EmployeeDAOFactory;  
import com.nt.utility.HibernateUtil;  
  
public class ClientApp {  
  
    public static void main(String[] args) {  
        // get DAO instance(obj)  
        EmployeeDAO dao=EmployeeDAOFactory.getInstance();  
        // call persistence methods  
        System.out.println(dao.register(1103,"raja", "rao", "rao@gmail.com"));  
        System.out.println("1103 Emp Details"+dao.search(1103));  
        System.out.println(dao.modifyEmail(101, "y@x.com"));  
        //System.out.println(dao.removeEmp(8888));  
        //close Session  
        HibernateUtil.closeSession();  
        //close SEssionFactory  
        HibernateUtil.closeSessionFactory();  
    }  
}
```

Layered Application:-

HBProj3(HibernateUtil+DAO+DTO+BO)

- src
  - com.nt.bo
    - StudentBO.java
  - com.nt.cfgs
    - hibernate.cfg.xml
    - student.hbm.xml
  - com.nt.controller
    - StudentController.java
  - com.nt.dao
    - StudentDAO.java
    - StudentDAOFactory.java
    - StudentDAOImpl.java
  - com.nt.dto
    - StudentDTO.java
  - com.nt.service
    - StudentService.java
  - com.nt.test
    - ClientApp.java
  - com.nt.utility
    - HibernateUtil.java

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/student.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

student.cfg.xml

```
<!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.bo.StudentBO" table="Student_tab">
    <id name="sno" /> <!-- Singular Id field -->
    <property name="sname" />
    <property name="total" />
```

```
<property name="avg" />
<property name="result" />
</class>
</hibernate-mapping>
```

**StudentBO.java**

```
package com.nt.bo;

public class StudentBO {
    private int sno;
    private String sname;
    private int total;
    private float avg;
    private String result;

    public int getSno() {
        return sno;
    }

    public void setSno(int sno) {
        this.sno = sno;
    }

    public String getSname() {
        return sname;
    }

    public void setSname(String sname) {
        this.sname = sname;
    }

    public int getTotal() {
        return total;
    }

    public void setTotal(int total) {
        this.total = total;
    }

    public float getAvg() {
        return avg;
    }

    public void setAvg(float avg) {
        this.avg = avg;
    }

    public String getResult() {
        return result;
    }

    public void setResult(String result) {
        this.result = result;
    }
}
```

**StudentDTO.java**

```
package com.nt.dto;

import java.io.Serializable;

public class StudentDTO implements Serializable{
    private int sno;
    private String sname;
    private int m1,m2,m3;

    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getName() {
        return sname;
    }
    public void setName(String sname) {
        this.sname = sname;
    }
    public int getM1() {
        return m1;
    }
    public void setM1(int m1) {
        this.m1 = m1;
    }
    public int getM2() {
        return m2;
    }
    public void setM2(int m2) {
        this.m2 = m2;
    }
    public int getM3() {
        return m3;
    }
    public void setM3(int m3) {
        this.m3 = m3;
    }
}
```

**StudentController.java**

```
package com.nt.controller;

import com.nt.dto.StudentDTO;
import com.nt.service.StudentService;
```

```
public class StudentController {  
  
    public String process(int sno, String sname, int m1, int m2, int m3){  
        StudentDTO dto=null;  
        StudentService service=null;  
        String result=null;  
        //prepare DTO obj  
        dto=new StudentDTO();  
        dto.setSno(sno); dto.setSname(sname); dto.setM1(m1);  
        dto.setM2(m2); dto.setM3(m3);  
        // use Service class  
        service=new StudentService();  
        result=service.generateResult(dto);  
        return result;  
    }  
  
}
```

### StudentDAO.java

```
package com.nt.dao;  
  
import com.nt.bo.StudentBO;  
public interface StudentDAO {  
    public int insert(StudentBO bo);  
}
```

### StudentDAOImpl.java

```
package com.nt.dao;  
  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import com.nt.bo.StudentBO;  
import com.nt.utility.HibernateUtil;  
  
public class StudentDAOImpl implements StudentDAO {  
  
    @Override  
    public int insert(StudentBO bo) {  
        Session session=null;  
        Transaction tx=null;  
        int no=0;  
        //get Session  
        session=HibernateUtil.getSession();  
        //Save obj  
        try{  
            tx=session.beginTransaction();  
            session.save(bo);  
            tx.commit();  
            no=1;  
        }catch(Exception e){  
            e.printStackTrace();  
            if(tx!=null)  
                tx.rollback();  
        }  
    }  
}
```

```

tx=session.beginTransaction();
no=(Integer)session.save(bo);
tx.commit();
}
catch(Exception e){
tx.rollback();
}
HibernateUtil.closeSession();
return no;
}//insert
}//class

```

**StudentDAOFactory.java**

```

package com.nt.dao;

public class StudentDAOFactory {

public static StudentDAO getInstance(){
return new StudentDAOImpl();
}

}

```

**StudentService.java**

```

package com.nt.service;

import com.nt.bo.StudentBO;
import com.nt.dao.StudentDAO;
import com.nt.dao.StudentDAOFactory;
import com.nt.dto.StudentDTO;

public class StudentService {

public String generateResult(StudentDTO dto){
int total=0;
float avg=0.0f;
String result=null;
// Write B.logic
total=dto.getM1()+dto.getM2()+dto.getM3();
avg=total/3.0f;
if(avg>=35)
result="pass";
else
result="fail";
//prepare BO
StudentBO bo=new StudentBO();
bo.setSno(dto.getSno());
bo.setSname(dto.getSname());
}

```

```

bo.setTotal(total);
bo.setAvg(avg);
bo.setResult(result);
//use DAO
StudentDAO dao=StudentDAOFactory.getInstance();
int no=dao.insert(bo);

return no+" student avg:"+avg+" result is"+result;
}//method
}//class

```

**HibernateUtil.java**

```

package com.nt.utility;
// same as above

```

**ClientApp.java**

```

package com.nt.test;

import com.nt.controller.StudentController;
import com.nt.utility.HibernateUtil;

public class ClientApp {
public static void main(String[] args) {
StudentController controller=null;
//Create Controller
controller=new StudentController();
String result=controller.process(105,"raja2",22,36,12);
System.out.println("Result:::"+result);
//Close Session
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

**Note:**

While developing hibernate mapping file specifying the column name is optional if domain class property names and DATABASE table column names are matching.

**Procedure to crate library for hibernate in eclipse:**

Right Click on project → buildpath → add library → user library folder → new HIB Lib → add external jar → browse and select → hibernate\_home → lib → required folder jar files → ok

**To add library in the project:**

Right click on project → build path → configure build path → add lib → user library → select your library → ok.

**Procedure to attach source code and documentation to the jar file of hibernate lib:**

Right click on project → build path → configure build path → remove existing library → add

library → user library → expand hibernate library → expand hibernate-core-<ver>-final.jar

- For source attachment : external location → external folder → <hibernate\_home>\project\hibernate-core → ok.
- For java doc attachment: java doc location select java doc in archive → select external file → browse → select hibernate framework zip file → path in archive → browse → select documentation folder → ok

We can also write <id>, <property>tags mapping file in two ways:

<id> tag:

<id name = "no">

    <column name="id"/>

</id>

Or

<id name="no" column="id"/>

<property> tag:

<property name="fname" column ="firstname"/>

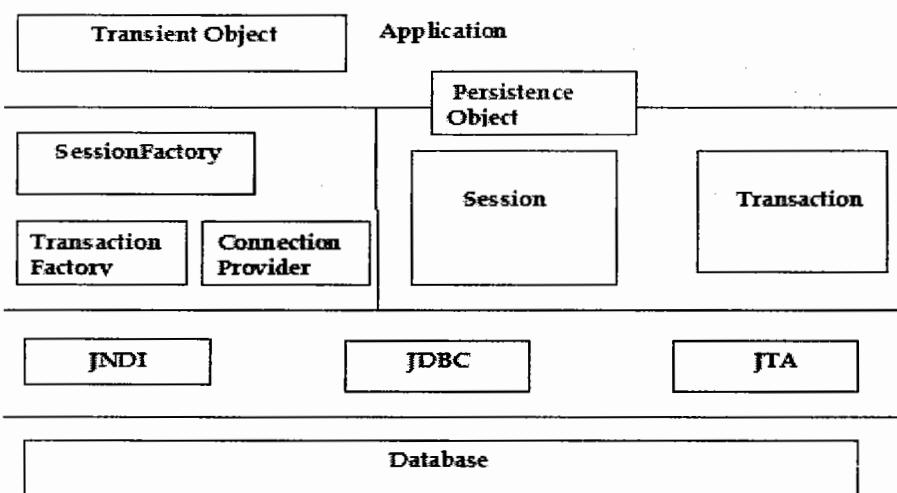
Or

<property name ="fname">

    <column name="firstname"/>

</property>

### Comprehensive architecture of Hibernate:



### Different States of Objects in Hibernate:

1) transient state 2) persistent state 3) Detached state

- Domain class object which is not associated with session is called **transient state object**

```
EmpDetails ed = new EmpDetails(); //transient state object
```

```
ed.setNo(1001);
```

.....

.....

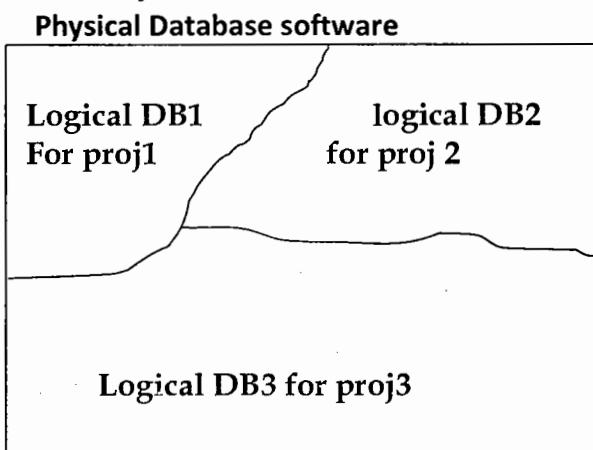
- The domain class object that is associated with session and performs persistence operation is called **persistence state object**  
`session.save(ed); //ed is persistence state object`
- The domain class object which is previously persistent is called **detached state object**  
`ses.close(); // ed becomes detached state object`

→ Connection provider is useful for hibernate to instruct Hibernate framework to create Jdbc connection pool. Hibernate internally can use c3p0, proxool and etc... softwares to create JDBC connection pool.

→ TransactionFactory will be used by session object while creating Transaction object it can use JDBC/JTA/CORBA/ server managed Transaction service internally to crate Transaction object.

→ Every physical DATABASE software installation contains 1 or more logical DBs on 1 per project basis. In oracle, every logical DATABASE is identified with its SID, In mySQL every logical DATABASE is identified with its logical DATABASE name.

### Working with MySQL:-



→ In real time people use GUI Database tool to perform Database operations eg: TOAD for oracle, TOAD for mysql, SQLYog for MySQL and etc...

#### Procedure to use SQLyog for mySQL to interact with mySQL Database software and to perform logical Database creation having Database table.

- Launch SQLyog → continue → new → name → password → root → connect
- Right click on root at localhost → create database → enter new database name → NTHB67DB → create
- Right click on NTHB67DB → create table
- Insert record into Database table.

#### Devx company supplied type4 mechanism based JDBC driver for mysql is called connector/i jdbc driver the details are:

Driver class name:                   org.gjt.mm.mysql.Driver  
    Or  
    com.mysql.jdbc.Driver

Jdbc url:                           jdbc:mysql://<logical Database> (for local mysql) (or)  
    Jdbc:mysql://<host>:<port no>/<logical Database> (For remote mysql)  
    Jar file:mysql.connector-java.5.1.6.jar.

**Procedure to interact with mySQL Database software from hibernate application:**

1. Keep First project ready
2. Add jar files to build path
3. Add mySQL Database software related configuration in hibernate.cfg.xml  
**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

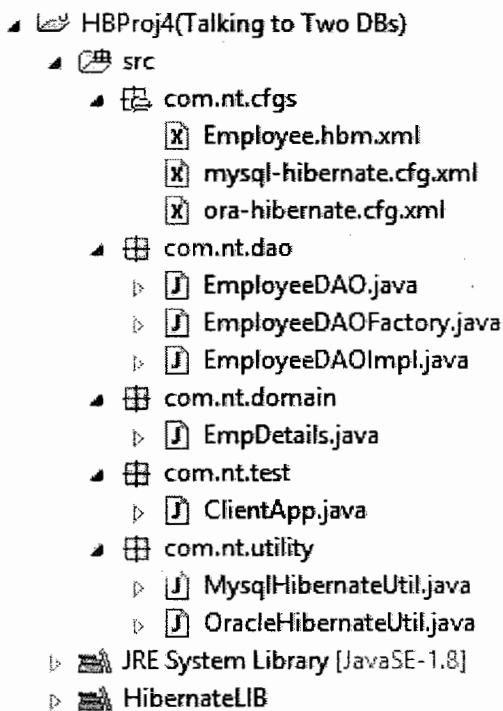
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
    <property name="connection.url">jdbc:mysql:///nthsb47db</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <property name="show_sql">true</property>
    <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**Interacting with multiple Database softwares**

→ For interacting with multiple DATABASE softwares, we need to create multiple SessionFactory objects having the support of multiple hibernate configuration files.

**Use cases:**

- Transferring customer details from 1 mobile network to another network as part of mobile number portability.
- Transferring employee detail from 1 department to another department.
- Transferring money between two accounts of two different banks.



[mysql-hibernate.cfg.xml](#)

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql:///nthb47db</property>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
        <property name="show_sql">true</property>
        <property name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

[ora-hibernate.cfg.xml](#)

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

[Employee.hbm.xml](#)

```
<!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.domain.EmployeeDetails" table="Employee">
        <id name="no"><!-- Singular Id field -->
            <column name="EID"/>
        </id>
        <property name="fname" >
            <column name="FIRSTNAME"/>
        </property>
    </class>
</hibernate-mapping>
```

```
</property>
<property name="lname" >
    <column name="LASTNAME"/>
</property>
<property name="mail" >
    <column name="EMAIL"/>
</property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;
public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;
    private String addrs;

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
}
```

```
public String getMail() {  
  
    return mail;  
}  
public void setMail(String mail) {  
  
    this.mail = mail;  
}  
  
public String getAddrs() {  
    return addrs;  
}  
  
public void setAddrs(String addrs) {  
    this.addrs = addrs;  
}  
  
@Override  
public String toString() {  
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

#### EmployeeDAO.java

```
package com.nt.dao;  
  
public interface EmployeeDAO {  
  
    public String transferEmployee(int no);  
}
```

#### EmployeeDAOFactory.java

```
package com.nt.dao;  
  
public class EmployeeDAOFactory {  
  
    public static EmployeeDAO getInstance(){  
        return new EmployeeDAOImpl();  
    }  
}
```

#### EmployeeDAOImpl.java

```
package com.nt.dao;  
  
import org.hibernate.Session;
```

```
import org.hibernate.Transaction;  
  
import com.nt.domain.EmpDetails;  
import com.nt.utility.MySQLHibernateUtil;  
import com.nt.utility.OracleHibernateUtil;  
  
public class EmployeeDAOImpl implements EmployeeDAO {  
  
    @Override  
    public String transferEmployee(int no) {  
        Session oraSes=null;  
        Session mysqlSes=null;  
        EmpDetails ed=null;  
        Transaction tx=null;  
  
        // Load obj from Oracle Employee Table  
        oraSes=OracleHibernateUtil.getSession();  
        ed=(EmpDetails)oraSes.get(EmpDetails.class, no);  
        // save obj to mysql Employee Table  
        mysqlSes=MysqlHibernateUtil.getSession();  
        try{  
            tx=mysqlSes.beginTransaction();  
            mysqlSes.save(ed);  
            tx.commit();  
        }  
        catch(Exception e){  
            tx.rollback();  
        }  
        MysqlHibernateUtil.closeSession();  
        //Delete obj from Employee Table  
        try{  
            tx=oraSes.beginTransaction();  
            oraSes.delete(ed);  
            tx.commit();  
        }  
        catch(Exception e){  
            tx.rollback();  
        }  
        OracleHibernateUtil.closeSession();  
        return ed.getNo()+" is successfully transferred";  
    }  
    //transferEmployee(-)  
}
```

### MysqlHibernateUtil.java

```
package com.nt.utility;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;
```

```
import org.hibernate.cfg.Configuration;

public class MysqlHibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal<Session>();
    private static SessionFactory factory;

    static{
        Configuration cfg=null;
        //Create SessionFactory obj
        cfg=new Configuration();
        cfg=cfg.configure("/com/nt/cfgs/mysql-hibernate.cfg.xml");
        factory=cfg.buildSessionFactory();

    }

    public static Session getSession(){

        Session session=null;
        if(threadLocal.get()==null){
            session=factory.openSession();
            threadLocal.set(session);
        }
        session=threadLocal.get();
        System.out.println(Thread.currentThread().hashCode()+"---->"+session.hashCode());
        return session;
    }//getSession()

    public static void closeSession(){
        Session session=null;
        session=threadLocal.get();
        session.close();
        threadLocal.remove();
    }

    public static void closeSessionFactory(){
        factory.close();
    }
}
```

#### OracleHibernateUtil.java

```
package com.nt.utility;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;
```

```
public class OracleHibernateUtil {  
    private static ThreadLocal<Session> threadLocal=new ThreadLocal<Session>();  
    private static SessionFactory factory;  
  
    static{  
        Configuration cfg=null;  
        //Create SessionFactory obj  
        cfg=new Configuration();  
        cfg=cfg.configure("/com/nt/cfgs/ora-hibernate.cfg.xml");  
        factory=cfg.buildSessionFactory();  
    }  
  
    public static Session getSession(){  
  
        Session session=null;  
        if(threadLocal.get()==null){  
            session=factory.openSession();  
            threadLocal.set(session);  
        }  
        session=threadLocal.get();  
        System.out.println(Thread.currentThread().hashCode()+"---->"+session.hashCode());  
        return session;  
    }//getSession()  
  
    public static void closeSession(){  
        Session session=null;  
        session=threadLocal.get();  
        session.close();  
        threadLocal.remove();  
    }  
  
    public static void closeSessionFactory(){  
        factory.close();  
    }  
}
```

**ClientApp.java**

```
package com.nt.test;  
import com.nt.dao.EmployeeDAO;  
import com.nt.dao.EmployeeDAOFactory;  
import com.nt.utility.MySQLHibernateUtil;  
import com.nt.utility.OracleHibernateUtil;  
  
public class ClientApp {  
    public static void main(String[] args) {
```

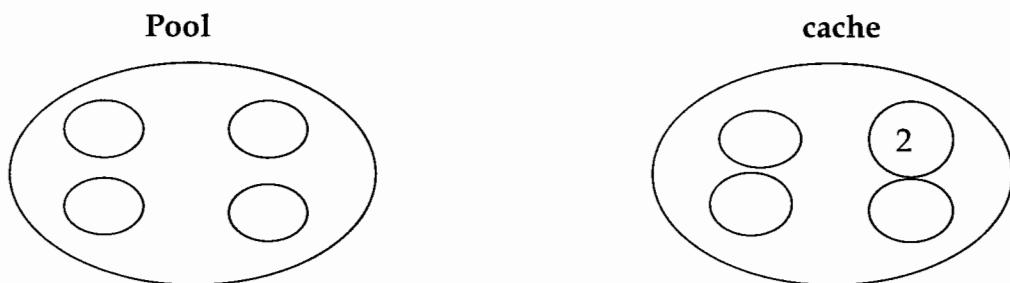
```
//get DAO
EmployeeDAO dao=EmployeeDAOFactory.getInstance();
System.out.println(dao.transferEmployee(1010));

//close SessionFactory
OracleHibernateUtil.closeSessionFactory();
MysqlHibernateUtil.closeSessionFactory();
}//main
}//class
```

Jar files: Hibernate LIB+mysql-connector-java-3.1.6.jar

### Connection pooling:

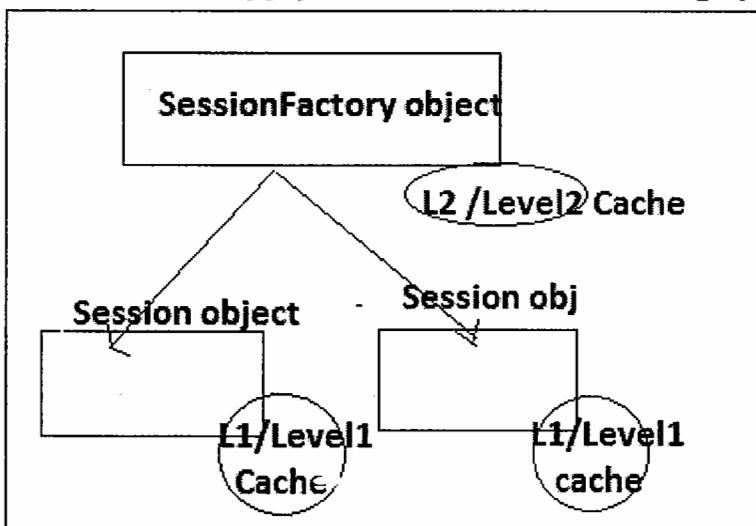
- Set of same items is called **pool**. Set of different items is called **cache**.
- pool and caches are given for the reusability of the items.



### Cache /Buffer:

- It is temporary that holds data for temporary period and reduces network trips between client and server across the multiple same requests.
- The buffer/cache of browser holds the data given by response for given request and it will be used across the multiple same requests.
- The cache of hibernate application resides at client application by holding the domain class objects and uses that across the multiple same requests.

### **Java App (Hibernate Persistence logic)**



**Hibernate supports two levels of cache:**

1. Level1 cache/Local cache/L1 cache/Session cache
2. Level2 cache/Global cache/L2 cache/Session Factory Cache

**Level1 cache:**

- It will be created when session object is created and it will be closed when session object is closed. We don't need to enable or disable this cache because it will be activated and deactivated automatically. It is 1 per session object having capability to hold domain class objects.
- Level1 cache keeps the tracks of all the modifications done in loaded objects and makes hibernate to generate single update select query at the end of Transaction when Transaction is committed instead of generating multiple update queries, it creates single update query.

**Sample code:**

```

EmpDetails ed=(EmpDetails)session.get(EmpDetails.class,1010);
Transaction tx=null;
try{
tx=session.beginTransaction();
ed.setMail("rao1@x.com");
ed.setMail("rao2@x.com");
ed.setMail("rao@gmail.com");
ed.setLname("chari");
ed.setLname("rao");
tx.commit();
}
catch(Exception e){
tx.rollback();
}

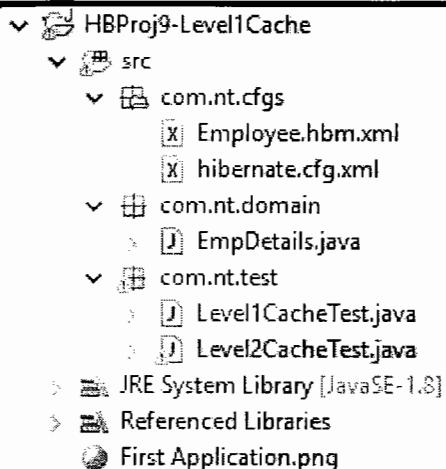
```

==>When Session object tries to load the object from DATABASE first it will check in Level1 cache, if it is available then it gets that object from Level1 cache ,if it is not available then it will get that object from Database and keeps in Level1 cache and uses that object across the multiple loadings of the same object.

```

// Gets from DATABASE and keeps in Level1 cache
EmpDetails ed=(EmpDetails)session.get(EmpDetails.class,1010);
System.out.println(ed);
//gets From Level1 cache
EmpDetails ed1=(EmpDetails)session.get(EmpDetails.class,1010);
System.out.println(ed1);
//session.evict(ed); // Removes from Level1 cache
session.clear(); //Removes all objs from Level1 cache
// Gets from DATABASE and keeps in Level1 cache
EmpDetails ed2=(EmpDetails)session.get(EmpDetails.class,1010);
System.out.println(ed2);

```

**Employee.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- O-R mapping cfgs -->
    <class name="com.nt.domain.EmpDetails" table="Employee">
        <id name="no" column="EID"/>

        <property name="fname">
            <column name="FIRSTNAME"/>
        </property>
        <property name="lname" column="LASTNAME"/>
        <property name="mail" column="EMAIL"/>
    </class>
</hibernate-mapping>
```

**hibernate.cfg.xml:-**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- connection 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>
        <!-- Dialect cfg -->
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <!-- mapping file -->
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

**EmpDetails.java:-**

```

package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {

```

```
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }

    @Override
    public String toString() {
        return "EmpDetails [no=" + no + ", fname=" + fname + ",
lname=" + lname + ", mail=" + mail + "]";
    }
}
```

**Level1CacheTest.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.domain.EmpDetails;

public class Level1CacheTest {

    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //read both xml files
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        //build SessionFactory
        factory=cfg.buildSessionFactory();
        //create Session
        ses=factory.openSession();

        //Load object
        details=ses.load(EmpDetails.class,1001);
        try{
            tx=ses.beginTransaction();
```

```
        details.setFname("Rajesh");
        details.setLname("rao");
        details.setLname("rao1");
        details.setMail("rao@1.com");
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }

    //close objs
    ses.close();
    factory.close();
} //main
}//class
```

**Level2CacheTest.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.domain.EmpDetails;

public class Level2CacheTest {

    public static void main(String[] args) {
        Configuration cfg=null;
        SessionFactory factory=null;
        Session ses=null;
        EmpDetails details=null, details1=null, details2=null;
        Transaction tx=null;
        //Activate HB framework based on jars added to build path
        cfg=new Configuration();
        //read both xml files
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        //build SessionFactory
        factory=cfg.buildSessionFactory();
        //create Session
        ses=factory.openSession();

        //Loads from DB and keeps in L1 cache
        details=ses.get(EmpDetails.class,1001);
        System.out.println(details);
        System.out.println("-----");
        //Loads from L1 cache
        details1=ses.get(EmpDetails.class,1001);
```

```

System.out.println(details);
System.out.println("-----");
//remove from L1 cache
ses.evict(details);
//Loads from DB and keeps in L1 cache
details2=ses.get(EmpDetails.class,1001);
System.out.println(details2);
System.out.println("-----");

//close objs
ses.close();
factory.close();
}//main
}//class

```

**To control Level1/L1 cache:**

1. ses.evict() : Evicts specific object from L1 cache.
2. ses.clear() : clears all the objects from L1 cache.
3. ses.close() : Closes the Session and its L1 Cache.

**Hibernate can keeps Domain class object in 3 states:**

1. **Transient** : Not associated with session
2. **Persistent**: Associated with session
3. **Detached** : Previously associated with session but not currently.

**Transient:**

→ Does not contain id value , does not represent DATABASE table record and does not associate with session object (i.e. Level1 cache). Changes done in this object does not reflect to DATABASE Table record.

*eg1: EmpDetail ed = null; //transient*

*eg2:EmpDetails ed = new EmpDetail(); //transient*

**Persistence:**

→ Associated with session ,contains id value, represents Database table record and maintains synchronization with Database table. As part of O-R mapping persistence logic we use this state objects a lot to perform persistence operations. All the objects of Level1 cache are persistent state objects.

->L1 cache of session object maintain persistent state object when transaction is committed all these objects will be synchronized with database table records.

**Example1:**

*ed= ses.get(EmpDetails.class,1101) // persistent*

**Example2:**

```

try{
    tx= ses.beginTransaction();
    ses.save(ed); //persistent
    tx.commit();
}

```

```

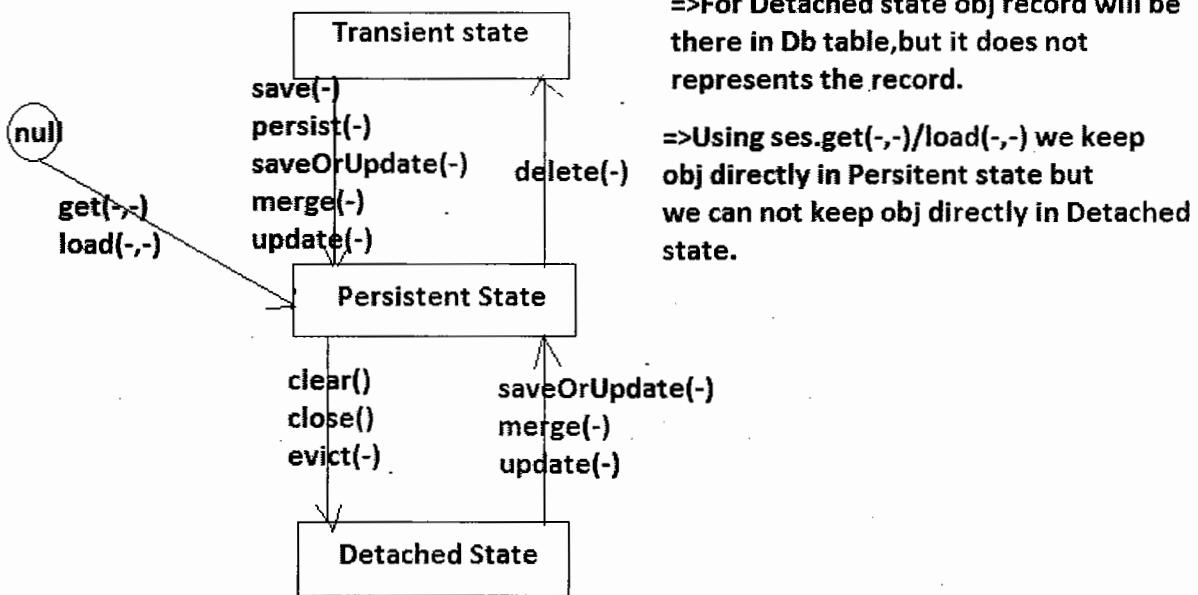
        }
        catch(Exception e){
        .....//
        }
    
```

**Detached (Previously Persistent):**

→ Previously associated with session, but not currently outside the session. Contains id value but does not represent db table record and does not maintain synchronization with DATABASE table record.

```

EmpDetail ed = null; //transient state
ed = ses.get(EmpDetails.class,1101) // persistent
    
```

**Life cycle diagram of domain class object state:****3 states of Domain class objs**

=>For transient state obj there will not be any record in Db table

=>For Detached state obj record will be there in Db table, but it does not represents the record.

=>Using ses.get(-)/load(-,-) we keep obj directly in Persistent state but we can not keep obj directly in Detached state.

- If we modify the **Persistent state object** in a TRANSACTION then the modification will be reflected to Database table record.
- If we modify the **Detached state object** or **Transient state object** even in the transaction, then the modification will not reflected because those objects are not representing Database table records.

**Q. What is different between ses.update(-) or ses.merge(-)?**

Ans: Both methods are useful to make transient state object ,detached state object as persistence state object. if we use ses.update(-) to make detached state object as persistence state object and if the Level1 cache of session object is already having one more object with same identity value then application throws **org.hibernate.NonUniqueObjectException**. If we use ses.merge(-) method the modifications will be merged and no exception will be raised.ses.merge(-) is powerful than ses.update(-) method.

**Example:**

```

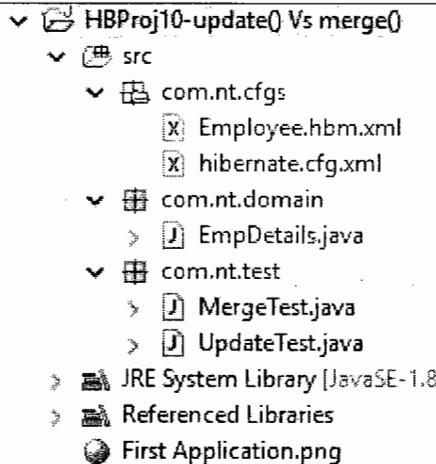
Session ses1=factory.openSession();
EmpDetails ed1=(EmpDetails)ses1.get(EmpDetails.class,1001); //persistent
System.out.println(ed1);
ses1.close();
ed1.setMail("bigb@124.com"); //Detached

```

```

Session ses2=factory.openSession();
EmpDetails ed2=(EmpDetails)ses2.get(EmpDetails.class,1001); //persistent
System.out.println(ed2);
Transaction tx=null;
try{
tx=ses2.beginTransaction();
ses2.update(ed1);
tx.commit();
}
catch(Exception e){
e.printStackTrace();
tx.rollback();
}

```

**Employee.hbm.xml:-**

```

<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- O-R mapping cfgs -->
    <class name="com.nt.domain.EmpDetails" table="Employee">
        <id name="no" column="EID"/>

        <property name="fname">
            <column name="FIRSTNAME"/>
        </property>
        <property name="lname" column="LASTNAME"/>
        <property name="mail" column="EMAIL"/>
    </class>

```

```
</hibernate-mapping>
```

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- connection 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>
        <!-- Dialect cfg -->
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <!-- mapping file -->
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**EmpDetails.java:-**

```
package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
```

```

        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }

    @Override
    public String toString() {
        return "EmpDetails [no=" + no + ", fname=" + fname + ",
lname=" + lname + ", mail=" + mail + "]";
    }
}

```

**MergeTest.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.domain.EmpDetails;

public class MergeTest {

    public static void main(String[] args) {
        SessionFactory factory=null;
        Session ses1=null,ses2=null;
        EmpDetails details1=null,details2=null,details3=null;
        Transaction tx=null;
        //create SessionFactory object
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        //create Session
        ses1=factory.openSession();
        ses2=factory.openSession();

        //Load object using session1(ses1)
        details1=ses1.get(EmpDetails.class,1001); //persistent
        System.out.println("ses1-->"+details1);
        ses1.close(); // detached
    }
}

```

```

//Load object using session2(ses2)
details2=ses2.get(EmpDetails.class,1001); //persistent
System.out.println("ses2--->"+details2);
//make details1 objs persistent obj using ses2
try{
    tx=ses2.beginTransaction();
    details1.setMail("116@456.com");
    details3=(EmpDetails)ses2.merge(details1);
    tx.commit();
    System.out.println("after merge()"+details3);
    System.out.println("Detached obj has become persistent
object");
} //try
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
}

//close objs
ses2.close();
factory.close();
}//main
}//class

```

UpdateTest.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.domain.EmpDetails;

public class UpdateTest {

    public static void main(String[] args) {
        SessionFactory factory=null;
        Session ses1=null,ses2=null;
        EmpDetails details1=null,details2=null;
        Transaction tx=null;
        //create SessionFactory object
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory(),
        //create Session
        ses1=factory.openSession();
        ses2=factory.openSession();

        //Load object using session1(ses1)

```

```

details1=ses1.get(EmpDetails.class,1001); //persistent
System.out.println("ses1-->" + details1);
ses1.close(); // detached

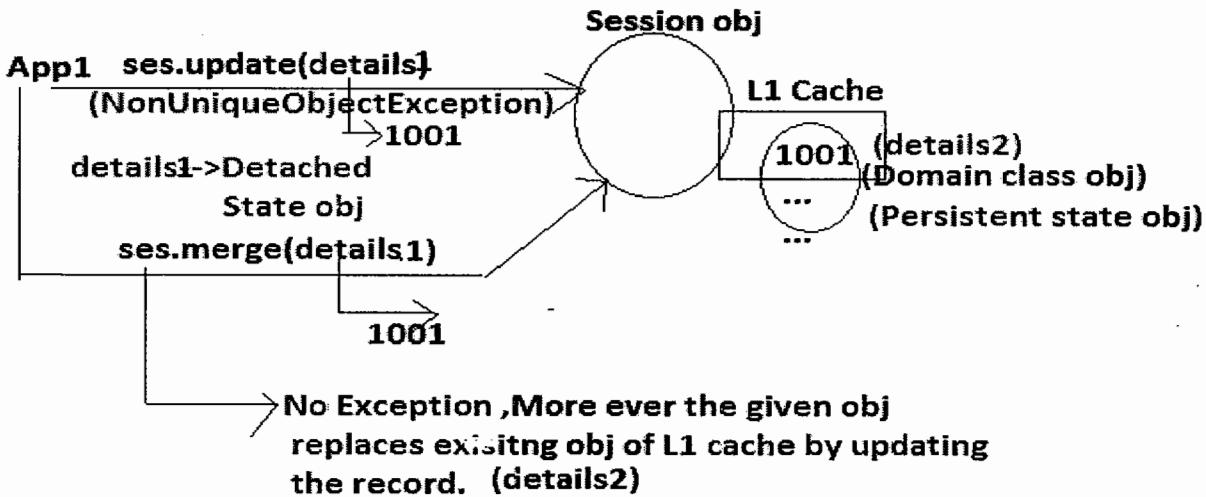
//Load object using session2(ses2)
details2=ses2.get(EmpDetails.class,1001); //persistent
System.out.println("ses2++->" + details2);
//make details1 objs persistent obj using ses2
try{
    tx=ses2.beginTransaction();
    ses2.update(details1);
    tx.commit();
    System.out.println("Detached obj has become persistent
object");
} //try
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
}

//close objs
ses2.close();
factory.close();
} //main
}//class

```

→ Ses2 object's Level1 cache is having ed2 object With identity value 1001 when so ses.update() is called with ed1 having same identity value 1001 then hibernate throws **org.hibernate.NonUniqueObjectException** because ses2 object Level1 cache is having 1 more object with same identity value. To overcome this problem use **ses.merge()**.

eg: ses2.merge(ed1) //does not throw Exception but merge the changes.Place instead of ses2.update()method



**Note:**

- Session object and its Level1 cache can't maintain multiple persistent state objects with same identity values i.e. persistent state objects must be unique objects.

**Use cases:**

- If clash comes between transferring voter id /voter card with old Voter Card Details then we can merge details using ses.merge(-) method.
- If the old registration details of customer are giving clash with new registration detail then we can merge the details using ses.merge(-)method.

**Generators in Hibernate:**

- JVM identifies the object using hashCode ,similarly hibernate identifies the Domain class object using identity values. The identity value will be stored into Identity filed/IdentifierField or property of Domain class ,that is configured by using <id> tag
- Hibernate gives predefined generators /algorithms to generate the identity values dynamically, if needed we can develope custom generators.
- Every generator of hibernate contains one **nick name** or **one class name**.
- All the classes of **generators** implements **org.hibernate.id.IdentifierGenerator(I)** to configure in hibernate mapping file, we need to use <generator> tag which is the sub tag of <id>tag.
- We can configure the generators only while working with **singular Identifier field Configuration**.

**Hibernate supplied predefined Generators:**

- increment → org.hibernate.id.IncrementGenerator
- assigned → org.hibernate.id.Assigned
- identity → org.hibernate.id.IdentityGenerator
- sequence -→ org.hibernate.id.SequenceGenerator
- hilo → org.hibernate.id.TableHiLoGenerator (Removed in 5.x)
- seqhilo → org.hibernate.id.SequenceHiLoGenerator
- native → no class
- select → org.hibernate.id.SelectHiLoGenerator
- foreign → org.hibernate.id.ForiegnGenerator
- uuid →
- guid →
- and etc.

**Note:-**

We can configure generator only while working with singular identity fields.

**While choosing generator considers:**

- Make sure that the identity value generated by generator is compatible to store in identity filed.

- Make sure that the underlying DATABASE software supports the generators.

**Syntax:**

```
<id name="....." column="....."> <!-- Singular Id field -->
    <generator class="....."/>
</id>
```

**assigned:**

- This generator works in all DATABASE softwares.
- Lets the application to assign the identity value to the identifier filed/property. This is default generator of the application if no generator is configured. it is useful to assign identity value to domain class object based on application requirement.

**Use cases:**

- Generating RTA registration number based on the number chosen by customer by paying premium amount.
- Choosing the mobile number.
- Choosing the bank Account number.

**Example:**

- Keep hibernate first applicaton ready.
- Configure **assigned generator** in mapping file

**In Employee.hbm.xml**

```
<id name="no" column="eid"> <!-- Singular Id field -->
    <generator class="assigned "/>
</id>
```

- Develop the client application having logic to save object.

**ClientApp.java**

```
package com.nt.test;

import org.hibernate.Transaction;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

import com.nt.domain.EmpDetails;

public class Generators_ClientApp {
    public static void main(String[] args) {
        //create Session object
        Configuration cfg=new Configuration();
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        SessionFactory factory=cfg.buildSessionFactory();
        Session session=factory.openSession();
```

```

//Save object
EmpDetails details=new EmpDetails();
details.setNo(1113);
details.setFname("raja");
details.setLname("rao");
details.setMail("rao@xx.com");
Transaction tx=null;
try{
    tx=session.beginTransaction();
    int id=(Integer)session.save(details);
    System.out.println("Generated id value"+id);
    tx.commit();
}
catch(Exception e){
    tx.rollback();
}

//close objs
session.close();
factory.close();
}//main
}//class

```

**Note:**

→ assigned generator makes the application to set identity value to object before calling ses.save(-) method or ses.persist(-).

**increment:**

- Generates the identity values of type long, short, int that are unique. this generator works with all Database softwares. uses  $<\max Value> + 1$  formula to generate the identity value.
- Does not consider deleted values and uses only currently available values to generate the next identity value.

**Example:****In Employee.hbm.xml**

```

<id name="no" column="eid"><!-- Singular Id field -->
    <generator class="increment"/>
</id>

```

**identity:**

→ Supports identity columns in DB2, mysql, postgresql, mysql server, Sybase and hypersonic sql . Returns identity value of type long, short, or int and uses  $<\text{previous value}+1>$  formulae to generate the identity value.

→ The column on which auto increment constraint is applied to generate next identity value based on previous generated value is called **identity column**. Oracle does not support identity columns, but mysql supports the identity columns.

**Note:**

- Make sure that application is pointing to mysql DATABASE software.
- Apply auto increment constraint on eid column using SQL yog
- Configure **identity** generator in mapping file

**In Employee.hbm.xml**

```
<id name="no" column="eid"> <!-- Singular Id field -->
  <generator class="identity" />
</id>
```

**Different between increment and identity**

increment	Identity
<ol style="list-style-type: none"> <li>1. Works in all DATABASE softwares</li> <li>2. Uses max value+1 formulae to generate the identity value</li> <li>3. Does not consider deleted value while generating the identity value.</li> <li>4. The column mapped with identity property need not have to special constraint (apart from primary key)</li> <li>5. Hibernate framework generates this identity value and assigns value to the domain object before saving the object.</li> </ol>	<ol style="list-style-type: none"> <li>1. Works in a DATABASE software that supports identity columns (like mysql)</li> <li>2. Uses previous value+1 to generate the identity value.</li> <li>3. Considers</li> <li>4. Need to have auto increment constraint to make the column as identity column.</li> <li>5. DATABASE software generates the identity value while saving the object and assigns the object.</li> </ol>

**Use cases for identity generator:**

- Generating adhar no
- Generating voter id and etc...

Note: In these cases the number will not be assigned to others even though card holders are expired (passed away).

**Sequence:**

→ Sequence generator uses sequence concept of DB2, postgresql, oracle, sap DATABASE and etc... to generate the identity of type long, int or short. works in oracle database software. After configuring this generator, if no sequence name is specified then the hibernate framework creates default sequence hibernate\_sequence having logic increment by 1.

**Example1:** (working with default hibernate\_sequence in oracle)

- a) Make sure that “**hbm2ddl.auto** property” placed in hibernate.cfg.xml with value “update/create”

**In hibernate.cfg.xml**

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

- b) Configure the sequence generator in hibernate mapping in Employee.hbm.xml

```
<id name="no" column="EID">
    <generator class="sequence"/>
</id>
```

**Example2:** using custom sequence (in oracle)

- Create sequence in oracle Database software
- SQL> crate sequence **EMPNO\_SEQUENCE** increment by 5 start with 1000;
- Configure the above sequence in mapping file

In Employee.hbm.xml:

```
<id name="no" column="EID"><!-- Singular Id field -->
    <generator class="sequence">
        <param name="sequence_name" >EMPNO_SEQUENCE</param>
    </generator>
</id>
```

→Same as identity generator with respect oracle but gives flexible to specify start range and end range ,increment by and etc ... other options.

**hilo:**

→Uses hi/lo algorithm to efficiently generates identity of type long, short, or int. Takes given table column value as the source of the high value and max\_lo param as the value of the low value. it works with all Database softwares.

**Example:**

- Create table having column with initial value

HI\_table  
hi\_col(number)  
100

- Configure hilo generator in mapping file

```
<id name="no" column="EID"><!-- Singular Id field -->
    <generator class="hilo">
        <param name="table" > HI_table</param>
        <param name="column" > hi_col </param>
        <param name="max_lo" > 5</param>
    </generator>
</id>
```

**Note:**

After each identity value generation the given helper table column incremented by 1.

**The formulae of hilo algorithm:**

$$\text{given higher table col value} * \text{max\_lo} + \text{param} + 1$$
  

$$(\text{hi value}) - (\text{low value})$$

100 \* (5+1) ==600  
 101 \* (5+1) ==606  
 102 \* (5+1) ==612

**Use cases:**

- Useful to generate lucky draw coupon numbers
- Generate OTP numbers.
- Password recovery.

**seqhilo:**

- Use hi/lo algorithm to generate identity value of type long, short, int where the given sequence generated value will be used as the source of the **hi value** and **max\_lo** param value will be used as the source of the **low value**.
- This generator works in that database software that supports **sequence** like oracle.

**Example:**

- a) Create sequence in oracle or use the existing sequence of oracle database software
- b) Configure "seqhilo" generator in mapping file

```
<id name="no" column="EID"> <!-- Singular Id field -->
  <generator class="hilo">
    <param name="sequence" >EMPNO_SEQUENCE</param>
    <param name="max_lo" >10</param>
  </generator>
</id>
```

**The formulae of seqhilo**

<sequence generated value> \* <max\_lo> + 1  
 (hi value) (low value)  
 1020 \* (10+1)=11220  
 1025 \* (10+1)=11275  
 1030 \* (10+1)=11330

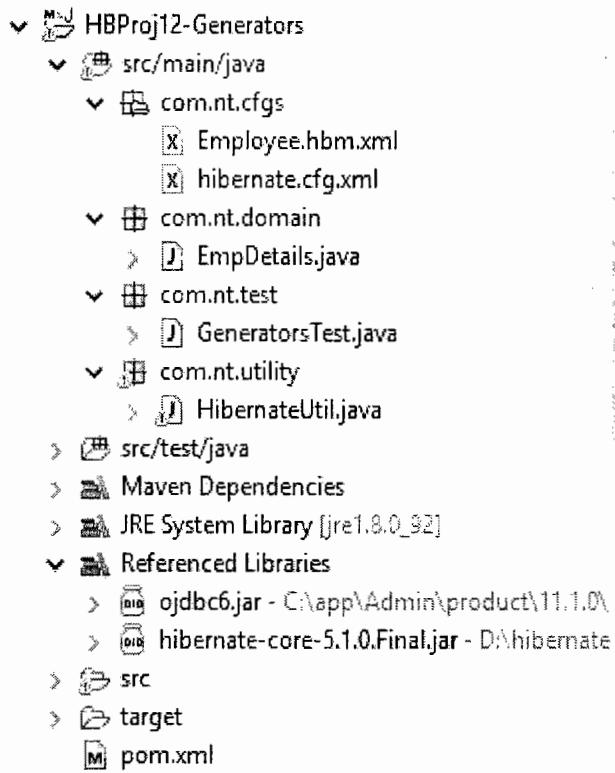
**native:**

→ Picks up and uses **identity** or **sequence** or **hilo** generator based on the capability of underlying Database software.

- If underlying Database software supports **identity column** then uses "**identity**" generator.
- If underlying Database software supports sequence then uses "**sequence**" generator.
- If underlying Database software does not support sequence/identity columns then it uses **hilo generator**.
- If underlying Database software supports both identity column and sequence then it uses **identity generator**.
- When it is configured for oracle it uses **sequence** generator, similarly when it is configured for mysql then it uses **identity** generator.

**Example:**

```
<id name="no" column="EID">
  <generator class="native"/>
</id>
```

**Employee.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <!-- O-R mapping cfgs -->
  <class name="com.nt.domain.EmpDetails" table="Employee">
    <!-- <id name="no" column="EID">
        <generator class="assigned"/>
    </id> -->

    <!-- <id name="no" column="EID">
        <generator class="org.hibernate.id.Assigned"/>
    </id> -->

  <!-- <id name="no" column="EID"/> -->
  <!-- <id name="no" column="EID">
      <generator class="identity"/>
  </id>
  -->
```

```
<!--    <id name="no" column="EID">
      <generator class="sequence"/>
    </id> -->

<!--<id name="no" column="EID">
    <generator class="sequence">
        <param name="sequence_name">EID_seq</param>
    </generator>
  </id> -->

<!--    <id name="no" column="EID">
      <generator class="increment"/>
    </id> -->
<!--    <id name="no" column="EID">
      <generator class="hilo"/>
    </id>
  -->
<!--    <id name="no" column="EID">
      <generator class="hilo">
          <param name="table">hi_tab</param>
          <param name="column">hi_col</param>
          <param name="max_lo">5</param>
      </generator>
    </id>
  -->
<!-- <id name="no"  column="EID">
    <generator class="seqhilo">
        <param name="max_lo">5</param>
    </generator>
  </id> -->

<!-- <id name="no"  column="EID">
    <generator class="seqhilo">
        <param name="sequence">EID_SEQ1</param>
        <param name="max_lo">5</param>
    </generator>
  </id>
  -->

<!-- <id name="no"  column="EID">
    <generator class="native">
        <param name="sequence_name">EID_SEQ1</param>
    </generator>
  </id> -->

<!-- <id name="no"  column="EID">
    <generator class="native"/>
  </id>
  -->
```

```

<id name="no" column="EID">
    <generator class="sequence-identity"/>
</id>

<property name="fname">
    <column name="FIRSTNAME"/>
</property>
<property name="lname" column="LASTNAME"/>
<property name="mail" column="EMAIL"/>
</class>
</hibernate-mapping>

```

**hibernate.cfg.xml:-**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

**EmpDetails.java:-**

```

package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
}

```

```
public void setNo(int no) {
    this.no = no;
}
public String getFname() {
    return fname;
}
public void setFname(String fname) {
    this.fname = fname;
}
public String getLname() {
    return lname;
}
public void setLname(String lname) {
    this.lname = lname;
}
public String getMail() {
    return mail;
}
public void setMail(String mail) {
    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ",
    lname=" + lname + ", mail=" + mail + "]";
}
}
```

**GeneratorsTest.java:-**

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class GeneratorsTest {

    public static void main(String[] args) {
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        int idval=0;
        //get Session
        ses=HibernateUtil.getSession();
        //save object
        details=new EmpDetails();
        details.setNo(1235);
```

```

details.setFname("karan");
details.setLname("rao");
details.setMail("rao@gmail.com");
try{
    tx=ses.beginTransaction();
    idval=(Integer)ses.save(details);
    System.out.println("Generated id value:::"+idval);
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}

/* try{
    tx=ses.beginTransaction();
    ses.persist(details);
    tx.commit();
    System.out.println("object is saved");
} //try
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
}*/

//close objects
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}//main
}//class

```

HibernateUtil.java:-

```

package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml")
.buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){

```

```

Session ses=null;
ses=threadLocal.get();
if(ses==null){
    ses=factory.openSession();
    threadLocal.set(ses);
}
System.out.println("Session obj hashCode::"+ses.hashCode());
return ses;
}//getSession()

public static void closeSession(){
Session ses=null;

ses=threadLocal.get();
if(ses!=null){
    ses.close();
    threadLocal.remove();
}
}

public static void closeSessionFactory(){
factory.close();
}

}//class

```

**pom.xml:-**

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>

<groupId>nit</groupId>
<artifactId>HBProj12-Generator</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>HBProj12-Generator</name>
<url>http://maven.apache.org</url>

<properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
</properties>

<dependencies>
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>

```

```
<version>3.8.1</version>
<scope>test</scope>
</dependency>
<!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.1.0.Final</version>
</dependency>

</dependencies>
</project>
```

### uuid:(universal unique id)

Generates 32 digit hexa decimal number as the string identity value it uses

- a) System ip address
- b) System clock and etc... details while generating this identity value.

#### Example:

- a) Keep First Project as the separate project
- b) Creates **employee1** Database table but alter eid column data type to varchar2
- c) Change **no** property type to string in EmpDetails.java file and reflect the change even in the setter and getter methods.

#### In EmpDetails.java

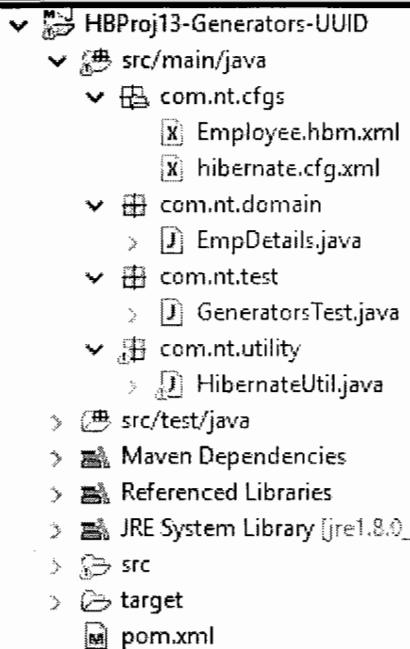
```
private String no;
```

- d) configure the generator in mapping file

```
<id name="no" column="EID">
    <generator class="uuid"/>
</id>
```

- e) Save the EmpDetails object In the Client Application.

→ This generator works with all Database softwares. This generator is useful to generate the random hexadecimal number as the identity value. May be useful to generate the Transaction id and e-commerce id.

**Employee.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!-- O-R mapping cfgs -->
    <class name="com.nt.domain.EmpDetails" table="Employee_UID">

        <id name="no" column="EID">
            <generator class="uuid"/>
        </id>

        <property name="fname">
            <column name="FIRSTNAME"/>
        </property>
        <property name="lname" column="LASTNAME"/>
        <property name="mail" column="EMAIL"/>
    </class>
</hibernate-mapping>
```

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/EmpLoyee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

**EmpDetails.java:-**

```
package com.nt.domain;

public class EmpDetails {
    private String no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public String getNo() {
        return no;
    }
    public void setNo(String no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
}
```

```
public void setLname(String lname) {
    this.lname = lname;
}
public String getMail() {
    return mail;
}
public void setMail(String mail) {
    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname +",
lname=" + lname + ", mail=" + mail + "]";
}

}
```

**GeneratorsTest.java:-**

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class GeneratorsTest {

    public static void main(String[] args) {
        Session ses=null;
        EmpDetails details=null;
        Transaction tx=null;
        String idval=null;
        //get Session
        ses=HibernateUtil.getSession();
        //save object
        details=new EmpDetails();
        //details.setNo(1235);
        details.setFname("karan");
        details.setLname("rao");
        details.setMail("rao@gmail.com");
        try{
            tx=ses.beginTransaction();
            idval=(String)ses.save(details);
            System.out.println("Generated id value:::"+idval);
            tx.commit();
        } //try
        catch(Exception e){
            tx.rollback();
        }
    }
}
```

```

/* try{
    tx=ses.beginTransaction();
    ses.persist(details);
    tx.commit();
    System.out.println("object is saved");
} //try
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
} */

//close objects
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}//main
}//class

```

HibernateUtil.java:-

```

package com.nt.utility;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        System.out.println("Session obj hashCode::"+ses.hashCode());
        return ses;
    }//getSession()

    public static void closeSession(){
        Session ses=null;

```

```
ses=threadLocal.get();
if(ses!=null){
    ses.close();
    threadLocal.remove();
}
}

public static void closeSessionFactory(){
    factory.close();
}

}//class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>nit</groupId>
  <artifactId>HBProj12-Generator</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>HBProj12-Generator</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.1.0.Final</version>
    </dependency>
  </dependencies>
</project>
```

**guid:**

→ Uses Database generated guid string on mysql and etc...database softwares

✓ HBProj14-Generators-GUID  
 ✓ src/main/java  
   ✓ com.nt.cfgs  
   ☒ Employee.hbm.xml  
   ☒ hibernate.cfg.xml  
   ✓ com.nt.domain  
     > EmpDetails.java  
   ✓ com.nt.test  
     > GeneratorsTest.java  
   ✓ com.nt.utility  
     > HibernateUtil.java  
 > src/test/java  
 > Maven Dependencies  
 > Referenced Libraries  
 > JRE System Library [jre1.8.0\_92]  
 > src  
 > target  
 pom.xml

**Employee.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <!-- O-R mapping cfgs -->
  <class name="com.nt.domain.EmpDetails" table="Employee_GUID">

    <id name="no" column="EID">
      <generator class="guid"/>
    </id>

    <property name="fname">
      <column name="FIRSTNAME"/>
    </property>
    <property name="lname" column="LASTNAME"/>
    <property name="mail" column="EMAIL"/>
  </class>
</hibernate-mapping>
```

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

  <session-factory>
```

```
<property  
name="connection.driver_class">com.mysql.jdbc.Driver</property>  
    <property name="connection.url">jdbc:mysql:///nths98db</property>  
    <property name="connection.username">root</property>  
    <property name="connection.password">root</property>  
    <property name="show_sql">true</property>  
    <property name="format_sql">true</property>  
    <property  
name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>  
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>  
</session-factory>  
  
</hibernate-configuration>
```

**EmpDetails.java:-**

```
package com.nt.domain;  
  
public class EmpDetails {  
    private String no;  
    private String fname;  
    private String lname;  
    private String mail;  
  
    public EmpDetails() {  
        System.out.println("EmpDetails:0-param constructor");  
    }  
  
    public String getNo() {  
        return no;  
    }  
    public void setNo(String no) {  
        this.no = no;  
    }  
    public String getFname() {  
        return fname;  
    }  
    public void setFname(String fname) {  
        this.fname = fname;  
    }  
    public String getLname() {  
        return lname;  
    }  
    public void setLname(String lname) {  
        this.lname = lname;  
    }  
    public String getMail() {  
        return mail;  
    }  
    public void setMail(String mail) {  
        this.mail = mail;  
    }  
}
```

```
@Override  
public String toString() {  
    return "EmpDetails [no=" + no + ", fname=" + fname + ",  
lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

**GeneratorsTest.java:-**

```
package com.nt.test;  
  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
import com.nt.domain.EmpDetails;  
import com.nt.utility.HibernateUtil;  
  
public class GeneratorsTest {  
  
    public static void main(String[] args) {  
        Session ses=null;  
        EmpDetails details=null;  
        Transaction tx=null;  
        String idval=null;  
        //get Session  
        ses=HibernateUtil.getSession();  
        //save object  
        details=new EmpDetails();  
        //details.setNo(1235);  
        details.setFname("karan");  
        details.setLname("rao");  
        details.setMail("rao@gmail.com");  
        try{  
            tx=ses.beginTransaction();  
            idval=(String)ses.save(details);  
            System.out.println("Generated id value:::"+idval);  
            tx.commit();  
        } //try  
        catch(Exception e){  
            tx.rollback();  
        }  
  
        /* try{  
            tx=ses.beginTransaction();  
            ses.persist(details);  
            tx.commit();  
            System.out.println("object is saved");  
        } //try  
        catch(Exception e){  
            tx.rollback();  
        }  
    }  
}
```

```
        e.printStackTrace();
    }*/  
  
    //close objects  
    HibernateUtil.closeSession();  
    HibernateUtil.closeSessionFactory();  
  
}//main  
}//class
```

**HibernateUtil.java:-**

```
package com.nt.utility;  
  
import org.hibernate.Session;  
import org.hibernate.SessionFactory;  
import org.hibernate.cfg.Configuration;  
  
public class HibernateUtil {  
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();  
    private static SessionFactory factory=null;  
    static{  
        factory=new Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").  
buildSessionFactory();  
        System.out.println("SessionFactory  
hashCode:::"+factory.hashCode());  
    }  
    public static Session getSession(){  
        Session ses=null;  
        ses=threadLocal.get();  
        if(ses==null){  
            ses=factory.openSession();  
            threadLocal.set(ses);  
        }  
        System.out.println("Session obj hashCode:::"+ses.hashCode());  
        return ses;  
    }//getSession()  
  
    public static void closeSession(){  
        Session ses=null;  
  
        ses=threadLocal.get();  
        if(ses!=null){  
            ses.close();  
            threadLocal.remove();  
        }  
    }  
  
    public static void closeSessionFactory(){
```

```
        factory.close();
    }

}//class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>nit</groupId>
  <artifactId>HBProj12-Generator</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>HBProj12-Generator</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.1.0.Final</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-
java -->
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.36</version>
    </dependency>
```

```
</dependencies>
</project>
```

**select:**

→Generates the identity value given by Database trigger by selecting unique row in another Database table.

**foreign:**

→Current object gets identity value from the associated object. It is very useful to build 1 to 1 association mapping. Generally **library membership id** is student registration id, So we can take the identity value as student object as the identity value of library membership object.

**Custom Id generators:**

- If predefined identifiers are not satisfying our application requirement then it is recommended to use custom generators.
- Every custom generator is a java class that implements **org.hibernate.id.IdentifierGenerator(i)**
- For custom generators we can't provide nick names ,so we need to configure class name directly.

**To configure custom generator in hibernate mapping file:**

```
<id name="no" column="EID">
  <generator class="<pkg>.<CustRandomGenerator>" />
</id>
```

**Example Application on Custom generator (Generating pseudo random number as Identity value):**

HBProj6(CustomGenerator1)  
 - src  
 - com.nt.cfgs  
 - Employee.hbm.xml  
 - hibernate.cfg.xml  
 - com.nt.domain  
 - EmpDetails.java  
 - com.nt.generators  
 - CustRandomGenerator.java  
 - com.nt.test  
 - Generators\_ClientApp.java

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

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

```

<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">scott</property>
<property name="connection.password">tiger</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

**Employee.hbm.xml**

```

!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.domain.EmpDetails" table="Employee">
  <id name="no" column="EID">
    <generator class="com.nt.generators.CustRandomGenerator"/>
  </id>
  <property name="fname" >
    <column name="FIRSTNAME"/>
  </property>
  <property name="lname" >
    <column name="LASTNAME"/>
  </property>
  <property name="mail" >
    <column name="EMAIL"/>
  </property>
</class>
</hibernate-mapping>

```

**EmpDetails.java**

```

package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private int no;
  private String fname;
  private String lname;
  private String mail;

  public EmpDetails(){
  System.out.println("EmpDetails:0-param constructor");
  }

  public int getNo() {
  }
}

```

```
return no;
}
public void setNo(int no) {

this.no = no;
}
public String getFname() {

return fname;
}
public void setFname(String fname) {

this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}
```

**CustRandomGenerator.java**

```
package com.nt.generators;

import java.io.Serializable;
import java.util.Random;
```

```

import org.hibernate.HibernateException;
import org.hibernate.engine.spi.SessionImplementor;
import org.hibernate.id.IdentifierGenerator;

public class CustRandomGenerator implements IdentifierGenerator{

    @Override
    public Serializable generate(SessionImplementor session,
                                 Object object) throws HibernateException {
        int id=0;
        Random rad=null;
        // Generator random number
        rad=new Random();
        id=rad.nextInt(10000);
        return id;
    }

}

```

**Generators ClientApp.java**

```

package com.nt.test;

import org.hibernate.Transaction;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

import com.nt.domain.EmpDetails;

public class Generators_ClientApp {
    public static void main(String[] args) {
        Configuration cfg=new Configuration();
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        SessionFactory factory=cfg.buildSessionFactory();
        Session session=factory.openSession();
        //Save obj
        EmpDetails details=new EmpDetails();
        details.setNo(1113);
        details.setFname("raja");
        details.setLname("rao");
        details.setMail("rao@xx.com");
        Transaction tx=null;
        try{
            tx=session.beginTransaction();
            int id=(Integer)session.save(details);
        }
    }
}

```

```

System.out.println("Generated id value"+id);
tx.commit();
}
catch(Exception e){
tx.rollback();
}

//close objs
session.close();
factory.close();
}//main
}//class

```

### Example Application2 on Custom Id generators to generate E001 E002 E003 as id values:

#### • HBProj7(CustomGenerator2)

- src
  - com.nt.cfgs
    - ☒ Employee.hbm.xml
    - ☒ hibernate.cfg.xml
  - com.nt.domain
    - ▷ EmpDetails.java
  - com.nt.generators
    - ▷ CustomIdGenerator.java
  - com.nt.test
    - ▷ Generators\_ClientApp.java

#### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

#### Employee.hbm.xml

```

<!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.domain.EmpDetails" table="Employee">
  <id name="no" column="EID">
    <generator class="com.nt.generators.CustomIdGenerator"/>
  </id>
  <property name="fname" >
    <column name="FIRSTNAME"/>
  </property>
  <property name="lname" >
    <column name="LASTNAME"/>
  </property>
  <property name="mail" >
    <column name="EMAIL"/>
  </property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private String no;
  private String fname;
  private String lname;
  private String mail;

  public EmpDetails(){
  System.out.println("EmpDetails:0-param constructor");
  }

  public String getNo() {
    return no;
  }
  public void setNo(String no) {
    this.no = no;
  }
  public String getFname() {
    return fname;
  }
  public void setFname(String fname) {
    this.fname = fname;
  }
  public String getLname() {
```

```
return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}
```

#### CustomIdGenerator.java

```
package com.nt.generators;

import java.io.Serializable;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

import org.hibernate.HibernateException;
import org.hibernate.engine.spi.SessionImplementor;
import org.hibernate.id.IdentifierGenerator;

public class CustomIdGenerator implements IdentifierGenerator {

@Override
public Serializable generate(SessionImplementor session, Object object)
throws HibernateException {
try {
// write jdb code
Connection con = session.connection();
PreparedStatement ps = con.prepareStatement("select seq1.nextval from dual");
ResultSet rs = ps.executeQuery();
int val = 0;
if (rs.next()) {
val = rs.getInt(1);
}
}
```

```
if(val < 10)
    return "E00" + val;
else if(val < 100)
    return "E0" + val;
else
    return "E" + val;
} catch (Exception e) {
e.printStackTrace();
return "Failed to generate Id value";
}
}// generate(-,-)
}// class
```

### Generators ClientApp.java

```
package com.nt.test;

import org.hibernate.Transaction;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

import com.nt.domain.EmpDetails;

public class Generators_ClientApp {
    public static void main(String[] args) {
        Configuration cfg=new Configuration();
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        SessionFactory factory=cfg.buildSessionFactory();
        Session session=factory.openSession();
        //Save obj
        EmpDetails details=new EmpDetails();
        details.setFname("raja");
        details.setLname("rao");
        details.setMail("rao@xx.com");
        Transaction tx=null;
        try{
            tx=session.beginTransaction();
            String id=(String)session.save(details);
            System.out.println("Generated id value"+id);
            tx.commit();
        }
        catch(Exception e){
            tx.rollback();
        }
    }
}
```

```
//close objs
session.close();
factory.close();
}//main
}//class
```

## Object Versioning:

- The process of keeping track of how many times the domain class object is loaded and modified is called **versioning**. Performing this work through Jdbc code need lot of event handling with complexity.
- Hibernate maintains object versioning through hibernate persistence logic. It does not take care of those modifications that are happened from SQL prompt.

### Use cases:

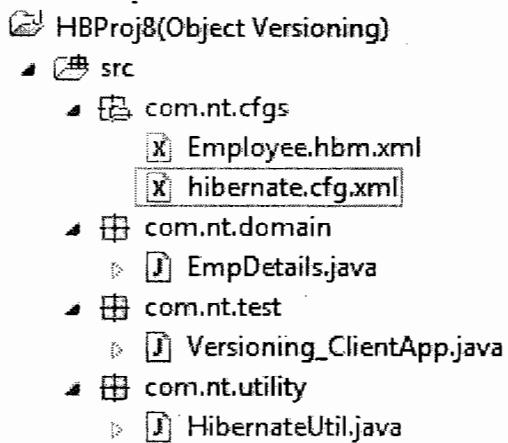
- Applying charges for utilizing address change service in the bank needs versioning to know how many times the address is changed.
- To applying charges on more than 5 utilizations of ATM service in a month needs versioning support.

### Example:

- Keep any hibernate application ready as separate project having HibernateUtil.java
- Add additional column “ver\_col” in Database table of type number  
SQL>Alter table employee add ver\_col number (5)
- Add additional property in domain class to keep track of versioning in EmpDetails.java  

```
private int ver;
//getter and setter
```
- Configure the above property in mapping file as versioning property using<version> tag.  

```
<version name="ver" column="ver_col"/>
```
- Develop the Client Application:



### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

```
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
    <id name="no" column="EID">
      <generator class="increment"/>
    </id>
    <version name="ver" column="ver_col"/>

    <property name="fname" >
      <column name="FIRSTNAME"/>
    </property>
    <property name="lname" >
      <column name="LASTNAME"/>
    </property>
    <property name="mail" >
      <column name="EMAIL"/>
    </property>
  </class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private int no;
  private String fname;
  private String lname;
  private String mail;
```

```
private int ver;

public void setVer(int ver){
    this.ver=ver;
}
public int getVer(){
    return ver;
}

public EmpDetails(){
System.out.println("EmpDetails:0-param constructor");
}

public int getNo() {

return no;
}
public void setNo(int no) {

this.no = no;
}
public String getFname() {

return fname;
}
public void setFname(String fname) {

this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
```

```
return "EmpDetails [no=" + no + ",fname=" + fname + ",lname=" + lname + ",mail=" + mail + "]";  
}  
  
}
```

**HibernateUtil.java**

Same as above

**Versioning\_ClientApp.java**

```
package com.nt.test;  
  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import com.nt.domain.EmpDetails;  
import com.nt.utility.HibernateUtil;  
  
public class Versioning_ClientApp {  
    public static void main(String[] args) {  
        // Get Session obj  
        Session ses=HibernateUtil.getSession();  
        /* //Save obj  
        EmpDetails details=new EmpDetails();  
        details.setNo(1113);  
        details.setFname("raja");  
        details.setLname("rao");  
        details.setMail("rao@xx.com");  
        Transaction tx=null;  
        try{  
            tx=ses.beginTransaction();  
            int id=(Integer)ses.save(details);  
            System.out.println("Generated id value"+id);  
            tx.commit();  
        }  
        catch(Exception e){  
            tx.rollback();  
        }  
        System.out.println("Object Saved with Version"+details.getVer()); */  
  
        //Load object  
        EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,1);  
        //Modify the obj  
        Transaction tx=null;  
        try{  
            tx=ses.beginTransaction();  
            ed.setMail("x@y1.com");  
            tx.commit();  
        }  
        catch(Exception e){  
            tx.rollback();  
        }
```

```

    }
System.out.println("Object is modified for "+ed.getVer()+" times");

```

```

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

## Hibernate Timestamp:

- Make hibernate to keep track of when (on which date and time) the object saved/updated .versioning keeps track of how many times the object is updated whereas timestamp keeps track when the object is **lastly accessed and updated**. It does not maintain history of object modifications. It just maintains when the object is saved or lastly updated (one value at a time).
- While working with xml configuration based hibernate programming we can't apply both versioning and time stamp features at a time in the application because we can place either **<version> or <timestamp> after<id> tag in hibernate mapping file.**

### Use cases:

- In a bank separate Database table will be maintained for daily transactions. In that database table each transaction date and time will be kept tracked by using time stamp feature.
- Every stock market share value maintain last updated date and time.
- Every mobile sim contains/maintains when it is lastly used.
- Every bank account maintains when it is lastly operated.

### Example:

- Keep first application ready as separate Project
- Add new column of time timestamp in employee Database table.

**SQL> alter table employee drop column ver\_col;**

**SQL> alter table employee add last\_updated\_timestamp;**

- Add new property to domain class of the type java.sql.TimeStamp;

**private Timestamp lstUpd;**

**//getter and setter**

- Configure the above property in mapping file using **<timestamp>tag (after <id> tag)**  
**<timestamp name="lstUpd" column="last\_updated"/>**

- Develop the ClientApp.

 HBProj9(Object timestamping)

```

    ▲ src
      ▲ com.nt.cfgs
        X Employee.hbm.xml
        X hibernate.cfg.xml
      ▲ com.nt.domain
        ▶ J EmpDetails.java
      ▲ com.nt.test
        ▶ J Timestamping_ClientApp.java
      ▲ com.nt.utility
        ▶ J HibernateUtil.java

```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="connection.pool_size">15</property>

    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>

```

[Employee.hbm.xml](#)

```

<!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.domain.EmpDetails" table="Employee">
    <id name="no" column="EID">
      <generator class="increment"/>
    </id>
    <timestamp name="lstUpd" column="last_updated"/>
    <property name="fname" >
      <column name="FIRSTNAME"/>
    </property>
  </class>
</hibernate-mapping>

```

```
<property name="lname" >
    <column name="LASTNAME"/>
</property>
<property name="mail" >
    <column name="EMAIL"/>
</property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

import java.sql.Timestamp;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    private Timestamp lstUpd;

    public void setLstUpd(java.sql.Timestamp lstUpd){
        this.lstUpd=lstUpd;
    }
    public java.sql.Timestamp getLstUpd(){
        return lstUpd;
    }

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
```

```

this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}

```

**HibernateUtil.java**

Same as above

**Timestamping ClientApp.java**

```

package com.nt.test;

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

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class Timestamping_ClientApp {
public static void main(String[] args) {
// Get Session obj
Session ses=HibernateUtil.getSession();
// Save obj
// EmpDetails details=new EmpDetails();
// details.setNo(1113);
// details.setFname("raja");
// details.setLname("rao");
// details.setMail("rao@xx.com");
// Transaction tx=null;
// try{

```

```

// tx=ses.beginTransaction();
// int id=(Integer)ses.save(details);
// System.out.println("Generated id value"+id);
// tx.commit();
// }
// catch(Exception e){
// tx.rollback();
// }
// System.out.println("Object Saved at "+details.getLstUpd());

//Load object
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,2);
//Modify the obj
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ed.setMail("x@y1.com");
    tx.commit();
}
catch(Exception e){
    tx.rollback();
}
System.out.println("Object is modified for "+ed.getLstUpd());

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

## **Connection pooling in Hibernate:**

- JDBC connection pool is a factory that contains set of readily available JDBC connection objects.
- Hibernate **SessionFactory** object represents JDBC connection pool and each connection object will be used while creating **Hibernate Session** object.
- Hibernate Session object represents session with Database software through hibernate framework because of this connection object.
- If not connection pool/connection provider is configured explicitly, then hibernate application uses hibernate framework managed built-in JDBC connection pool which is not good for **production use**. Becoz of bad performance and inability of working with connection objects.
- The built-in JDBC connection pool managed by hibernate cannot pool and it also gives bad performance. So do not use this connection pool in **real time practices**.
- To control this built-in JDBC con pool's the max size we can use "**connection.pool\_size**" property.

### In hibernate.cfg.xml

```
<property name="connection.pool_size">20</property>
```

**Q. What is the JDBC connection pool that u have used in hibernate application?**

Ans:

- If hibernate code is placed in standalone application/project that runs outside the server then use “**c3p0**” or “**proxool**” software based JDBC connection pool.
- If hibernate code is placed in web application or enterprise application that is deployable in web server/application server then we can use **server managed JDBC connection pool**.
- Hibernate framework uses connection provider classes that is configured in **hibernate.cfg.xml** file as the value of **connection.provider\_class** property of **hibernate.cfg.xml** file
  - a) **org.hibernate.engine.JDBC.connection.internal.DriverManagerConnectionProviderImpl**( to use hibernate built-in jdbc connection pool) default.
  - b) **org.hibernate.c3p0.internal.C3P0ConnectionProvider** (to use c3p0 jdbc connection pool)
  - c) **org.hibernate.proxool.internal.ProxoolConnectionProvider**(to use proxool jdbc connection pool)
  - d) **org.hibernate.engine.JDBC.connection.internal.DatasourceConnectionProvider**(to use server managed jdbc connection pool)

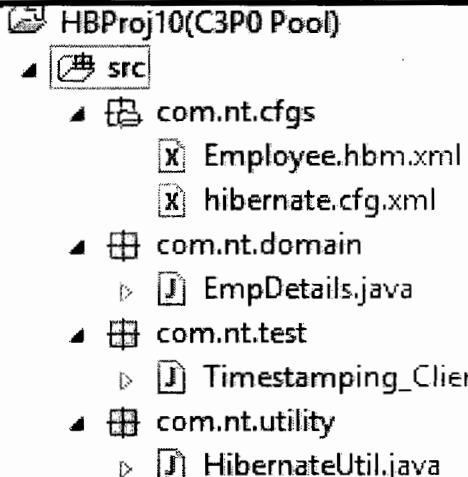
## Procedure to use c3p0 JDBC connection pool in hibernate connection pool.

- a) Keep first application ready as separate project.
- b) Add **c3p0 connection pool** software related jar file to the build path of the project.
- c) Configure c3p0 pool related connection provider class, properties in hibernate configuration file.

### In hibernate.cfg.xml

```
<property name="c3p0.max_size">100</property>
<property name="c3p0.min_size">10</property>
<property name="c3p0.timeout"> 5000</property>
<property name="c3p0.acquire_increment">2</property>
<property
name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider
</property>
```

**Note:** We can gather all the above property names and values from <hibernate\_home>/project/etc/hibernate\_properties file.

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="connection.pool_size">15</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>

        <property name="c3p0.max_size">100</property>
        <property name="c3p0.min_size">10</property>
        <property name="c3p0.timeout"> 5000</property>
        <property name="c3p0.acquire_increment">2</property>

        <property
            name="connection.connection_provider">org.hb.c3p0.internal.C3P0ConnectionProvider</property>
            <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
            <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID">
            <generator class="increment"/>
    </class>
</hibernate-mapping>
```

```
</id>
<timestamp name="lstUpd" column="last_updated"/>
<property name="fname" >
  <column name="FIRSTNAME"/>
</property>
<property name="lname" >
  <column name="LASTNAME"/>
</property>
<property name="mail" >
  <column name="EMAIL"/>
</property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

import java.sql.Timestamp;

public class EmpDetails {
  //domain class properties
  private int no;
  private String fname;
  private String lname;
  private String mail;

  private Timestamp lstUpd;

  public void setLstUpd(java.sql.Timestamp lstUpd){
    this.lstUpd=lstUpd;
  }
  public java.sql.Timestamp getLstUpd(){
    return lstUpd;
  }

  public EmpDetails(){
  System.out.println("EmpDetails:0-param constructor");
  }

  public int getNo() {
    return no;
  }
  public void setNo(int no) {

    this.no = no;
  }
  public String getFname() {
```

```
return fname;
}
public void setFname(String fname) {

this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}

}
```

### HibernateUtil.java

Same as previous

### ClientApp.java

```
package com.nt.test;

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

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class ClientApp {
public static void main(String[] args) {
// Get Session obj
Session ses=HibernateUtil.getSession();
```

```

//Save obj
EmpDetails details=new EmpDetails();
details.setNo(1113);
details.setFname("raja");
details.setLname("rao");
details.setMail("rao@xx.com");
Transaction tx=null;
try{
tx=ses.beginTransaction();
int id=(Integer)ses.save(details);
System.out.println("Generated id value"+id);
tx.commit();
} catch(Exception e){
tx.rollback();
}
System.out.println("Object Saved at "+details.getLstUpd());
//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

### **Procedure to configure proxool JDBC connection pool in our hibernate application:**

- Keep any hibernate project ready as separate copy.
- Add proxool pool related jar file to build path <hibernate\_home>/lib/optional/proxool folder 2jar files.
- Configure proxool pool related connection provider class in hibernate.cfg.xml
 

```

<property name="connection.connection_provider">
  org.hibernate.proxool.internal.ProxoolConnectionProvider
</property>
```
- Develop a separate xml file with any name having proxool pool configurations.

#### **proxool.cfg.xml**

```

<proxool-config>
  <proxool>
    <alias>pool1</alias>
    <driver-url>jdbc:oracle:thin:@localhost:1521:xe</driver-url>
    <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
    <driver-properties>
      <property name="user" value="scott"/>
      <property name="password" value="tiger"/>
    </driver-properties>
    <minimum-connection-count>10</minimum-connection-count>
    <maximum-connection-count>20</maximum-connection-count>
  </proxool>
</proxool-config>

```

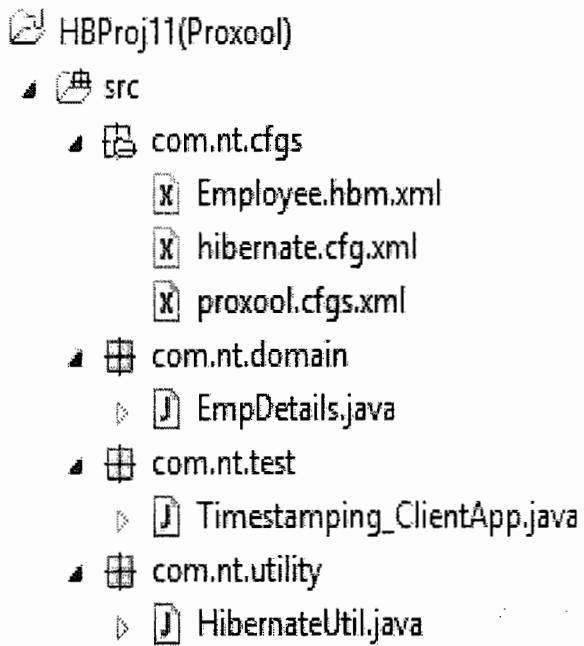
- e) configure proxool pool properties in hibernate cfg file

```
<property name="proxool.pool_alias">pool1</property>
<property name="proxool.xml">com/nt/cfgs/proxool.cfgs.xml</property>
```

->*Both these names must match in one xml file of proxool pool configuration ,we can Configure multiple connection pool with different aliasname.So different aliasname must be specified in hibernate configuration file.*

=>Since JDBC properties are placed in separate proxool.cfg.xml there is no need of writing same properties in hibernate.cfg.xml file

- f) Develop the client application:



### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>

        <property
            name="connection.connection_provider">org.hibernate.proxool.internal.ProxoolConnectionProvider
        </property>

        <property name="proxool.pool_alias">pool1</property>
        <property name="proxool.xml">com/nt/cfgs/proxool.cfgs.xml</property>

        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**proxool.cfgs.xml**

```
<proxool-config>
    <proxool>
        <alias>pool1</alias>
        <driver-url>jdbc:oracle:thin:@localhost:1521:xe</driver-url>
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <driver-properties>
            <property name="user" value="scott"/>
            <property name="password" value="tiger"/>
        </driver-properties>
        <minimum-connection-count>10</minimum-connection-count>
        <maximum-connection-count>20</maximum-connection-count>
    </proxool>
</proxool-config>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID">
            <generator class="increment"/>
        </id>
        <property name="fname" >
            <column name="FIRSTNAME"/>
        </property>
    </class>
</hibernate-mapping>
```

```
</property>
<property name="lname" >
    <column name="LASTNAME"/>
</property>
<property name="mail" >
    <column name="EMAIL"/>
</property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

import java.sql.Timestamp;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    private Timestamp lstUpd;

    public void setLstUpd(java.sql.Timestamp lstUpd){
        this.lstUpd=lstUpd;
    }
    public java.sql.Timestamp getLstUpd(){
        return lstUpd;
    }

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
```

```
this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}
```

**HibernateUtil.java**

Same as above

**ClientApp.java**

```
package com.nt.test;

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

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class ClientApp {
public static void main(String[] args) {
// Get Session obj
Session ses=HibernateUtil.getSession();
//Save obj
EmpDetails details=new EmpDetails();
details.setNo(1113);
details.setFname("raja");
details.setLname("rao");
details.setMail("rao@xx.com");
```

```
Transaction tx=null;
try{
tx=ses.beginTransaction();
int id=(Integer)ses.save(details);
System.out.println("Generated id value"+id);
tx.commit();
} catch(Exception e){
tx.rollback();
}
System.out.println("Object Saved at "+details.getLstUpd());

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class
```

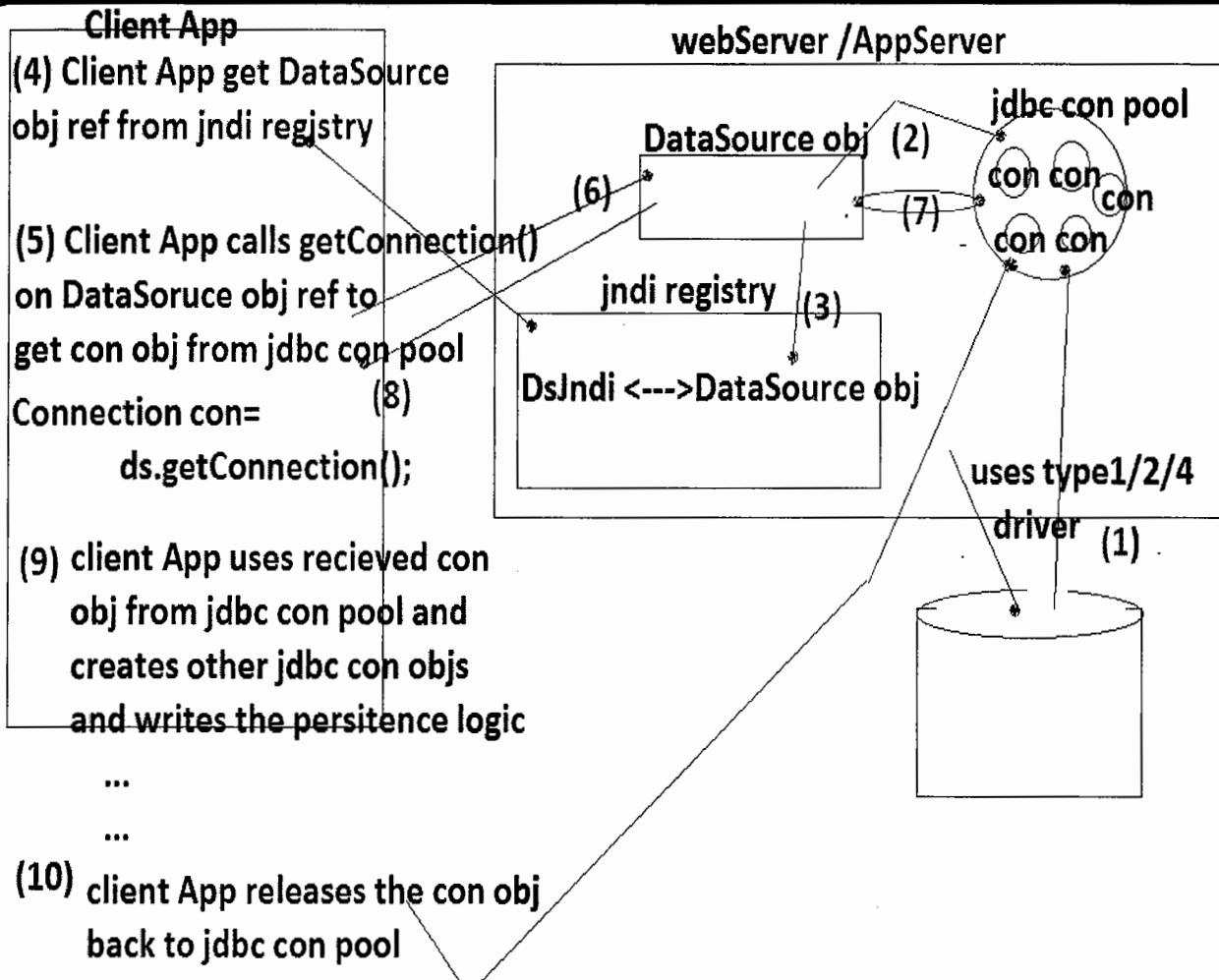
#### Different types of JDBC Connection pools:

##### **1. client side JDBC connection pool:**

- created and managed outside the server softwares
  - Suitable for standalone projects
- eg:c3p0,proxool,apache dbcp

##### **2. Server side JDBC connection pool:**

- Created and managed in web server/ application server softwares.
- Suitable in the applications that are deployable in web server/application server softwares.
- Datasource object represents JDBC connection pool .Each JDBC connection object from JDBC connection pool must be accessed through JDBC Data source object.
- Datasource object is the object of java class that implements javax.sql.DataSource(i)
- For global visibility and accessibility this JDBC DataSource object reference will be maintained in jndi registry having nick name/jndiname
- To provide global visibility or accessibility to object /object reference it will be placed in jndi registry.  
eg: rmi registry, cos registry, ldap registry and etc...
- Every web server/ application server gives built-in jndi registry.



→ Client application uses the received connection object to create other JDBC object and to develop persistence logic.

→ Client application calls **con.close()** to release the connection object back to JDBC connection.

#### With respect to the diagram:

1. TL(Team Leader)/PL (Project Leader) makes server to use type1/type2/type4 driver to interact with Database software and to create JDBC connection object in JDBC connection pool.
- 2&3. Datasource object represents JDBC connection pool and it will be placed in jndi registry with nickname/alias name for global visibility.

**Note:** We can see client application running outside the server(standalone application) or inside server(servlet).

→ In one server we can have multiple JDBC connection pools belonging to one DB software or multiple Database softwares.

→ TL/PL creates connection pool in server and all team members will utilize that connection pool.

#### WebLogic:

Type: application sever software

Vendor: BEA systems(oracle corp)

Port no: 7001

Commercial software

Allow to domains each domain act as 1 application server .No default domains

Version: 12c(compatible with jdk1.7)

To open admin console: http:localhost:7001/console

To download software: [www.commerce.bea.com](http://www.commerce.bea.com) or [www.oracle.com](http://www.oracle.com)

→ If multiple projects are using weblogic then weblogic software will be installed only for one time in common computer, but multiple domains will be created for multiple projects on 1 per domain basis. Every domain is one instance of weblogic acting as separate application server.

### **Procedure to create user defined domain server in weblogic 12c.**

All programs→Oracle→oracle home→web logic server → configuration wizard→create a new domain→domain name→next → next → next → username:javaboss and password:javaboss→ next→ mode→ development → next → select administration server → next→ port no: 7080 → next→create.

### **Procedure to create DataSource representing JDBC Connection pool for Oracle in weblogic Domain server.**

1. Start domain server and open admin console

2. Create JDBC DataSource pointing to JDBC connection pool for oracle.

Admin console→ service→DataSource→new generic data source→  
(logical name)→DsJndi→next→database driver→Oracle thin driver for service connections  
→next →next→database name(xe)→host name(localhost)→database user(scott)→password(tiger)→confirm password(tiger)→next→ test configuration→ next→ select→admin server→finish.

→The moment we press finish button, the datasource object that is created will be placed in built-in JNDI registry.

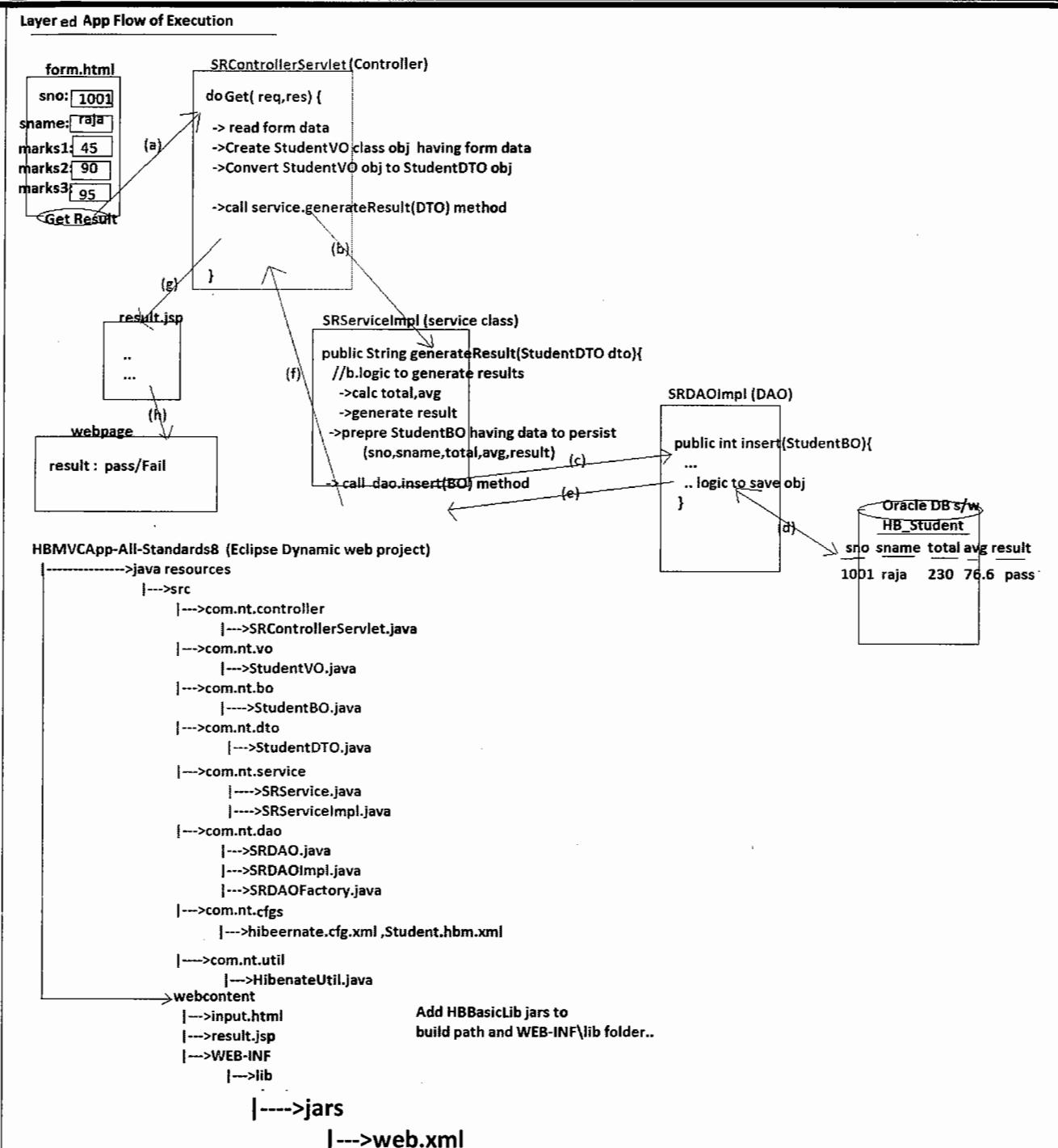
3. Specify the JDBC connection pool properties.

Admin console screen→ services→ data source name→ launch ds1→ connection pool tab  
→ initial capacity =10→ maximum capacity =100→minimum capacity =2→ save→advanced→shrink frequency=1000 →save.

#### **Note:**

Keep the data source object reference in jndi registry with nick name DsJndi for global visibility.

#### **Layered application flow:**



- This deployable web application that is using hibernate persistence logic to interact with DATABASE software, so use server managed JDBC connection pool
- Place the jar files of hibernate libraries in class path and also WEB-INF/lib folder. In case eclipse place in build path and perform **deployment assembly**.

**HBMVCApp-All-Standards12 (Eclipse Dynamic web project)**

```

----->java resources
    |--->src
        |--->com.nt.controller
            |--->SRControllerServlet.java
        |--->com.nt.vo
            |--->StudentVO.java
        |--->com.nt.bo
            |--->StudentBO.java
        |--->com.nt.dto
            |--->StudentDTO.java
        |--->com.nt.service
            |--->SRSERVICE.java
            |--->SRSERVICEImpl.java
        |--->com.nt.dao
            |--->SRDAO.java
            |--->SRDAOImpl.java
            |--->SRDAOFacory.java
        |--->com.nt.cfgs
            |--->hibernate.cfg.xml ,Student.hbm.xml
        |--->com.nt.util
            |--->HibenateUtil.java
----->webcontent
    |--->input.html
    |--->result.jsp
    |--->WEB-INF
        |--->lib
            |--->jars
        |--->web.xml

```

Add HBBasicLib jars to  
build path and WEB-INF\lib folder..

**hibernate.cfg.xml**

```

<!-- <!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"> -->

<hibernate-configuration>
    <session-factory>
        <!-- connection 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="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="connection.datasource">DsJndi</property>

```

```

<property name="connection.provider_class">
    org.hibernate.engine.jdbc.connections.internal.DataSourceConnectionProviderImpl
</property>
<!-- mapping file name -->
<mapping resource="com/nt/cfgs/student.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

***student.hbm.xml***

```

<!-- <!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.bo.StudentBO" table="hb_student">
    <id name="sno"/>
    <property name="sname"/>
    <property name="total"/>
    <property name="avg" />
    <property name="result"/>
</class>
</hibernate-mapping>

```

***StudentVO.java***

```

package com.nt.vo;

public class StudentVO {
    private String sno;
    private String sname;
    private String m1,m2,m3;

    public String getSno() {
        return sno;
    }
    public void setSno(String sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getM1() {
        return m1;
    }
    public void setM1(String m1) {
        this.m1 = m1;
    }
}

```

```
}

public String getM2() {
    return m2;
}

public void setM2(String m2) {
    this.m2 = m2;
}

public String getM3() {
    return m3;
}

public void setM3(String m3) {
    this.m3 = m3;
}

}
```

### StudentDTO.java

```
package com.nt.dto;

import java.io.Serializable;

public class StudentDTO implements Serializable {
    private int sno;
    private String sname;
    private int m1,m2,m3;

    public int getSno() {
        return sno;
    }

    public void setSno(int sno) {
        this.sno = sno;
    }

    public String getSname() {
        return sname;
    }

    public void setSname(String sname) {
        this.sname = sname;
    }

    public int getM1() {
        return m1;
    }

    public void setM1(int m1) {
        this.m1 = m1;
    }

    public int getM2() {
        return m2;
    }
```

```
public void setM2(int m2) {
    this.m2 = m2;
}
public int getM3() {
    return m3;
}
public void setM3(int m3) {
    this.m3 = m3;
}
}
```

**StudentBO.java**

```
package com.nt.bo;

public class StudentBO {
    private int sno;
    private String sname;
    private int total;
    private float avg;
    private String result;

    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getName() {
        return sname;
    }
    public void setName(String sname) {
        this.sname = sname;
    }
    public int getTotal() {
        return total;
    }
    public void setTotal(int total) {
        this.total = total;
    }
    public float getAvg() {
        return avg;
    }
    public void setAvg(float avg) {
        this.avg = avg;
    }
    public String getResult() {
        return result;
    }
}
```

```
public void setResult(String result) {
    this.result = result;
}

}
```

**SRCDAO.java:**

```
package com.nt.dao;

import com.nt.bo.StudentBO;

public interface SRCDAO {
    public int insert(StudentBO bo);
}
```

**SRCDAOImpl.java**

```
package com.nt.dao;

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

import com.nt.bo.StudentBO;
import com.nt.utility.HibernateUtil;

public class SRCDAOImpl implements SRCDAO {

    @Override
    public int insert(StudentBO bo) {
        Session ses=null;
        Transaction tx=null;
        int cnt=0;
        //get Session
        ses=HibernateUtil.getSession();
        //write persistence logic
        try{
            tx=ses.beginTransaction();
            ses.save(bo);
            tx.commit();
            cnt=1;
        }
        catch(Exception e){
            cnt=0;
        }
        HibernateUtil.closeSession();
        return cnt;
    }//method

}//class
```

**SRCDAOFactory.java**

```
package com.nt.dao;

public class SRCDAOFactory {

    public static SRCDAO getInstance(){
        return new SRCDAOImpl();
    }

}
```

**HibernateUtil.java****SRCController.java**

```
package com.nt.controller;

import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.nt.dto.StudentDTO;
import com.nt.service.SRCService;
import com.nt.vo.StudentVO;

public class SRCController extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
    IOException {
        String no=null;
        String name=null;
        String m1=null,m2=null,m3=null;
        StudentVO vo=null;
        StudentDTO dto=null;
        SRCService service=null;
        String result=null;
        RequestDispatcher rd=null;
        //read form data
        no=req.getParameter("sno");
        name=req.getParameter("sname");
        m1=req.getParameter("m1");
        m2=req.getParameter("m2");
        m3=req.getParameter("m3");
        //prepare StudentVO
    }
}
```

```

vo=new StudentVO();
vo.setSno(no);
vo.setSname(name);
vo.setM1(m1); vo.setM2(m2); vo.setM3(m3);
//prepare StudentDTO
dto=new StudentDTO();
dto.setSno(Integer.parseInt(vo.getSno()));
dto.setSname(vo.getSname().trim());
dto.setM1(Math.round(Float.parseFloat(vo.getM1())));
dto.setM2(Math.round(Float.parseFloat(vo.getM2())));
dto.setM3(Math.round(Float.parseFloat(vo.getM3())));
//use Service class
service=new SRCService();
result=servicegenerateResult(dto);
//keep result in the request attribute
req.setAttribute("resultMsg",result);
//forward request to result.jsp
rd=req.getRequestDispatcher("/result.jsp");
if(rd!=null)
rd.forward(req,res);
}//doGet(-,-)
@Override
public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
doGet(req,res);
}
}
}

```

**SRCService.java**

```

package com.nt.service;

import com.nt.bo.StudentBO;
import com.nt.dao.SRCDAO;
import com.nt.dao.SRCDAOFactory;
import com.nt.dto.StudentDTO;

public class SRCService {

    public String generateResult(StudentDTO dto){
        int total=0;
        float avg=0.0f;
        String result=null;
        SRCDAO dao=null;
        int count=0;
        StudentBO bo=null;
        //calc total,avg and generate result
        total=dto.getM1()+dto.getM2()+dto.getM3();
    }
}

```

```
avg=total/3.0f;
if(avg>=35)
    result="pass";
else
    result="fail";
//prepare BO/Domain class obj having data to persisted
bo=new StudentBO();
bo.setSno(dto.getSno());
bo.setSname(dto.getSname());
bo.setTotal(total);
bo.setAvg(avg);
bo.setResult(result);

//use dAO
dao=SRCDAOFactory.getInstance();
count=dao.insert(bo);

if(count==1)
    return result+" Student Registration successful";
else
    return result+" student registration failed";
}//generateResult
}//class
```

#### ***SessionFactoryClosingContextListener.java***

```
package com.nt.listeners;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import com.nt.utility.HibernateUtil;

public class SessionFactoryClosingContextListener implements ServletContextListener {

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("Servletcontext obj destroyed");
        HibernateUtil.closeSessionFactory();

    }

    @Override
    public void contextInitialized(ServletContextEvent arg0) {
        System.out.println("Servletcontext obj is created");

    }
}
```

**web.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
  app_3_0.xsd" id="WebApp_ID" version="3.0">
  <display-name>HBMVCApp-All-Standards3</display-name>
  <welcome-file-list>
    <welcome-file>input.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <servlet-name>student</servlet-name>
    <servlet-class>com.nt.controller.SRCController</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>student</servlet-name>
    <url-pattern>/controller</url-pattern>
  </servlet-mapping>
  <listener>
    <listener-class>com.nt.listeners.SessionFactoryClosingContextListener</listener-class>
  </listener>
</web-app>

```

**Input.html**

```

<h1><center> Student Registration</center></h1>
<form action="controller" method="post">
  sno: <input type="text" name="sno" placeholder="type sno"><br>
  sname: <input type="text" name="sname" placeholder="type sname"><br>
  Marks1: <input type="text" name="m1" placeholder="marks1"><br>
  Marks2: <input type="text" name="m2" placeholder="marks2"><br>
  Marks3: <input type="text" name="m3" placeholder="marks3"><br>
  <input type="submit" value="submit">
</form>

```

**result.jsp**

```

<hr/>
<h1>Result::::${resultMsg}</h1>
<hr/>

```

**Note:**

The jar file added to buildpath of the project will not be moved to automatically to move them use deployment assembly option.

**Deploye the web application in web logic server:**

- Export the war file of the Web application to created domain.  
D:oracleweblogic\user\_project\domain\WLNTHB47domain\autodeploy\webHbApp.war
- Start the weblogic server  
<weblogic\_home>\user\_project\domains\WLNTHB47\start weblogic.cmd
- Test the web application.

**Glassfish:**

Type: application server

Vendor: SunMs(Oracle corp.)

Open source

Version: 4.x (compatible with jdk 1.7)

**Default port no: 8080(http) 4848 (admin console)**

Allows to create domains, default domain name is "domain1"

To download software : [www.glassfish.org](http://www.glassfish.org)

**Procedure to create user-defined domain in glassfish server:**

<Glassfish\_home>\glassfish\bin>asadmin create-domain --user=testuser --adminport=4343

GFNTHB57Domain

Enter password: testuser

Retype the password: testuser

**Procedure to change the above http port no of GFNTHB57Domain server:**

Go to <glassfish4.x\_home>\glassfish\domain\ GFNTHB57Domain\config\domain.xml

Modify the port attribute of first<Network-listener>tag.

**Procedure to create Data Source pointing to JDBC connection pool for oracle**

**GFNTHB57Domain domain of GlassFish 4.x server.**

**Step1:** place odbc14/ojdbc6.jar in<Glassfish\_home>\domain\GFNTHB57\lib\ext folder.

**Step2:** start GFNTHB57 domain server of glassfish

<Glassfish\_home>\bin\asadmin start-domain GFNTHB57

**Step3:** open admin console:

<http://localhost:4343> --> username: ..... password .....

**Step4:** create JDBC connection pool for oracle:

Admin screen console→resource→jdbc→jdbc connection pool → new →general settings

Pool name : pool1

Resource type: java.sql.Driver

DataBase vendor: Oracle→next

Initial and min pool size : 8

Max pool size: 100  
 Pool Resize Quantity: 2  
 Idle Timeout: 300 sec  
 Max Wait Time: 6000  
 url: jdbc:oracle:thin:@localhost:1521:xe  
 user: scott  
 password: tiger  
 finish → launch pool → ping → save.

#### **Step 5: create DataSource pointing to the above JDBC connection pool**

Admin console → Resources → jdbc → jdbc resources (data source) → new → resources (data source name) → new → Jndi name: DSJndi → Pool name: pool1 → ok.

#### **Note:**

Having the JNDI name “DSJndi” the Data source object reference will be placed in JNDI of glassfish sever.

#### **Procedure to deploye web application in glassfish4.x server from eclipse IDE.**

Right click on dynamic web project → export → war file → browse → d:glassfish4.x\glassfish\domain\GFNTHB47domain\auto deploy  
 Test the web application  
<http://localhost:6666/WebHBAApp>

#### **Procedure to create JDBC connection pool for Oracle and DataSource in tomcat8/9.**

**Step1: make sure that tomcat-jdbc .jar , ojdbc6.jar file are placed in <tomcat\_home>\lib folder.**  
 In tomcat 8/9 server tomcat-jdbc.jar file comes along with tomcat software installation. If not coming ,get it by extracting apache-tomcat-jdbc-1.1.0.1 –bin.zip (download from internet)

**Step2: write the following < Resource> tag in <Context> of <Tomcat\_home>\conf\context.xml**

```
<Resource auth="Container" name="DSJndi" type="javax.sql.DataSource"
  factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
  driverClassName="oracle.jdbc.driver.OracleDriver"
  maxActive="100" maxIdle="30" maxWait="10000" username="scott" password="tiger"
  url="jdbc:oracle:thin:@localhost:1521:xe"/>
```

**Step3: restart the server.**

#### **Procedure to make the above web application running in tomcat server using that server managed JDBC connection pool:**

- Provide the following jndi name in hibernate.cfg.xml  
`<property name="connection.datasource"> java:/comp/env/DSJndi </property>`
- Test the web application.

#### **Note:**

→ If we don't want to use above connection pool from the tomcat server that is linked with eclipse IDE then write <Resource>tag in server section of context.xml in eclipse IDE.

## Composite identity filed/composite identifier:

- If database having singular pk constraint then configure pk column related single property of domain class as singular identity filed using `<id>` tag.
- If Database table having composite primary key constraint applied on multiple columns then configure those multiple columns related multiple properties of domain class as composite identity filed using `<composite-id>` tag. We can't apply pk constraint on single column of Database table in that situation we check the possibility of applying pk constraint on multiple columns of Database table as shown below that maintains programmers and projects info

### Programmers Projects

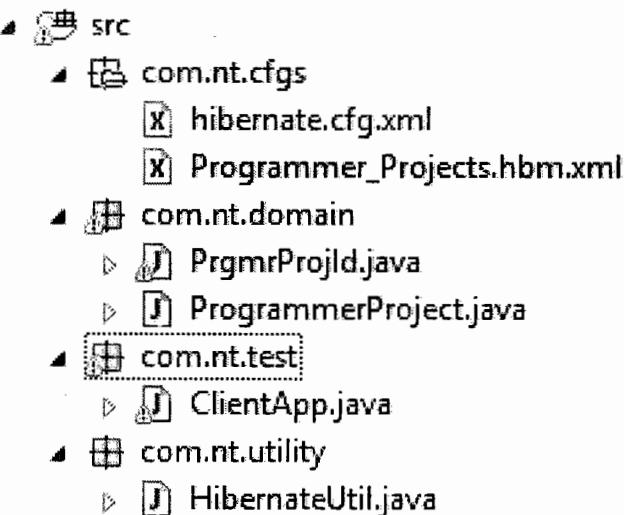
<u>Proj_id</u>	<u>proj_id</u>	<u>progmr_name</u>	<u>proj_name</u>	<u>salary</u>
101	1001	raja	openfx	9000
102	1002	rani	emarfn	8000
101	1001	raja	openfx	8500

We can't apply single pk constraint both individual column but we can apply composite pk constraint on both column together.

### Programmers Project

```
Create table programmers_project(programmer_id number(5), proj_id number(5), progmr_name
varchar(20), progmr_name varchar2(20), porj_name varchar(20), salary number(20), primary key
(programmer_id, proj_id));
```

### HBProj13(Composite Id Field)



### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

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

```

<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">scott</property>
<property name="connection.password">tiger</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/department.hbm.xml"/>
<mapping resource="com/nt/cfgs/empdetails.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

**Programmer Projects.hbm.xml**

```

<!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.domain.Projects_Programmers">
<composite-id name="id" class="com.nt.domain.PrgmrProjId">
<key-property name="prgmrlId" column="prgmr_id"/>
<key-property name="projName" column="proj_name"/>
</composite-id>
<property name="tl" column="tl_name"/>
</class>
</hibernate-mapping>

```

**PrgmrProjId.java**

```

package com.nt.domain;

import java.io.Serializable;

public class PrgmrProjId implements Serializable {
private int prgmrlId;
private String projName;

public PrgmrProjId() {
}

public PrgmrProjId(int prgmrlId, String projName) {
this.prgmrlId = prgmrlId;
this.projName = projName;
}

public int getPrgmrlId() {
return prgmrlId;
}
}

```

```
public void setPrgmrId(int prgmrlId) {  
    this.prgmrId = prgmrlId;  
}  
  
public String getProjName() {  
    return projName;  
}  
  
public void setProjName(String projName) {  
    this.projName = projName;  
}  
  
@Override  
public String toString() {  
    return "PrgmrProjId [prgmrlId=" + prgmrlId + ", projName=" + projName  
    + "]";  
}  
}
```

**ProgrammersProject.java**

```
package com.nt.domain;  
  
public class ProgrammersProject.java  
{  
    private PrgmrProjId id;  
    private String tl;  
    public Projects_Programmers() {  
    }  
  
    public Projects_Programmers(PrgmrProjId id, String tl) {  
        this.id = id;  
        this.tl = tl;  
    }  
  
    public PrgmrProjId getId() {  
        return id;  
    }  
    public void setId(PrgmrProjId id) {  
        this.id = id;  
    }  
    public String getTl() {  
        return tl;  
    }  
    public void setTl(String tl) {  
        this.tl = tl;  
    }  
}
```

```
@Override  
public String toString() {  
    return "Projects_Programmers [id=" + id + ", tl=" + tl + "]";  
}
```

**HibernateUtil.java**

```
//same as above
```

**ClientApp.java**

```
package com.nt.test;  
  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import com.nt.domain.PrgmrProjId;  
import com.nt.domain.Projects_Programmers;  
import com.nt.utility.HibernateUtil;  
  
public class ClientApp {  
  
    public static void main(String[] args) {  
        //Get Session  
        Session ses=HibernateUtil.getSession();  
  
        //Save Obj  
        PrgmrProjId id=new PrgmrProjId(1001,"proj1");  
        Projects_Programmers pp=new Projects_Programmers();  
        pp.setId(id);  
        pp.setTl("raja");  
        Transaction tx=null;  
        try{  
            tx=ses.beginTransaction();  
            PrgmrProjId cldVal=(PrgmrProjId)ses.save(pp);  
            System.out.println("Obj saved with compiste Id"+cldVal);  
            tx.commit();  
        } //try  
        catch(Exception e){  
            tx.rollback();  
        }  
  
        //Load object  
        /* PrgmrProjId id=new PrgmrProjId(1001,"proj1");  
         * Projects_Programmers  
         * pp=(Projects_Programmers)ses.get(Projects_Programmers.class,id);  
         * System.out.println(pp);  
         */  
    }  
}
```

While working with composite id there is no possibility of configuring any generator.

**PostgreSQL:**

Type: Database software(multi user)  
 Version: 9.x  
 Vendor: Enterprise DATABASE  
     Open source  
 Default port no: 5432  
 Default Username: postgres  
 Default password: tiger  
 Default logical Database: postgres  
 Allows to create more new logical dbs.  
 To download software: [www.postgresql.org](http://www.postgresql.org)

**Procedure to create logical Database having Database table with records in postgresql Database.**

Start → all program → PostgreSQL → postgres admin3 → right click on PostgreSQL → connect → right click on database → new data base → name → NTHB57DB → ok.  
 Go to NTHB57DB → schema → public → right click on table → new table → (name of table = emp) → column tab → add → name of the column → .....

**The enterprise Database company supplied type4 driver mechanism based JDBC driver for PostgreSQL is called PostgreSQL thin driver the details are:**

Driver class: org.postgresql.Driver  
 url: Jdbc:postgresql://<host>:<port>/<logical Database>  
 jar file: postgresql-8.4-701.jdbc.jar  
 (download from [www.postgresql.org](http://www.postgresql.org))

**Hibernate dialect for postgresql:**

org.hibernate.dialect.PostgreSQL9Dialect

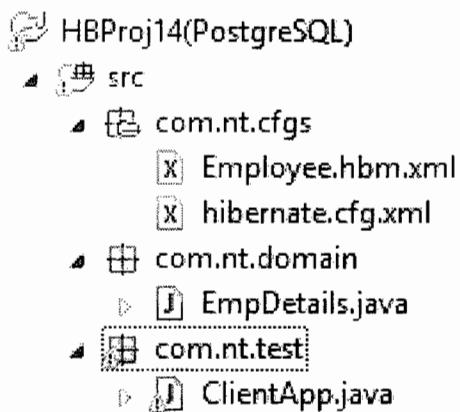
**Note:**

To make any hibernate application talking postgresql Database software Add postgresql-8.4-701.jdbc4.jar file to build path.

**Develop hibernate.cfg.xml as shown below**

```
<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">org.postgresql.Driver</property>
    <property name="connection.url">jdbc:postgresql:NTHB47DB</property>
    <property name="connection.username">postgres</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="dialect">org.hibernate.dialect.PostgreSQL9Dialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**Note:** in Postgresql logical Database name is case sensitive and column names and table names must be designed with lower case letters.



### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <property name="connection.driver_class">org.postgresql.Driver</property>
        <property name="connection.url">jdbc:postgresql:NTHB47DB</property>
        <property name="connection.username">postgres</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="dialect">org.hibernate.dialect.PostgreSQL9Dialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

### Employee.hbm.xml

```

<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID"/><!-- Singular Id field -->

        <property name="fname" >
            <column name="FIRSTNAME"/>
        </property>
        <property name="lname" >
            <column name="LASTNAME"/>
        </property>
        <property name="mail" >
            <column name="EMAIL"/>

```

```
</property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
```

```
this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}

}
```

**ClientApp.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.domain.EmpDetails;

public class ClientApp {

public static void main(String[] args) {
//Activate Hibernate f/w
Configuration cfg=new Configuration();
//Read and store HB cfg file , mapping file data into Configuration obj
cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
//create SessionFactory obj
SessionFactory factory=cfg.buildSessionFactory();

// get Session obj
Session ses=factory.openSession();

        //Save obj (insert record)
// create Domain class obj with data
EmpDetails ed=new EmpDetails();
ed.setNo(1021); ed.setFname("raja");
ed.setLname("rao"); ed.setMail("raja@x.com");

Transaction tx=null;
try{
tx=ses.beginTransaction();
int idval=(Integer)ses.save(ed); //saves obj (inserts the record)
System.out.println("Generated Id value"+idval);
tx.commit();
System.out.println("Object Saved /Record inserted");
}
catch(Exception e){
```

```

tx.rollback();
}

//close objs
ses.close();
factory.close();
}//main
}//class

```

### **Annotation Basics:**

- Annotation can be used for code declaration
  - @Override: makes the java method as overridden method
  - @Entity: makes the java bean as domain class /entity class of hibernate
- Annotation can be used to make underlying container or framework to perform dynamic code generation
  - eg: @Webservice: when this annotation is placed on the top of the class the underlying container/framework generates 1 dynamic sub class having the logic of web service.
  - @Ejb similar
- Annotation for documentation:
  - @Document
- Annotation for type conversion(from java 8)

If both annotation and xml file having the same configuration then the configuration done in xml file will override the configuration done through annotation.

While developing annotation based hibernate application we place mapping annotation in domain class mapping class in hibernate configuration file:

<mapping class="pkg.<class Name>" />

The mapping annotation configuration done in domain class be overridden by using the configuration done in hibernate mapping xml file (if placed)

### **What is JPA?**

JPA is software specification that gives set of rules and guideline to develop one or more ORM software while annotation it's recommended to use more JPA annotation in domain class to make domain portable to use in multiple ORM framework based application development.

In annotation driven hibernate programming we can use two type annotation

a) JPA annotation

eg:@Table ,@Id ,@column etc...

b) Hibernate annotation

Use these annotations when JPA annotations are not available

@Filter, @FilterDef and etc...

### **Simple domain class with annotation:**

@Entity // to make java bean as domain class

@Table(name="student\_tab") // to map domain class with Database table

Public class Student{

```

@Id
@Column(name="stno")
Private int sno;
@Column(name="stname")
Private String sname
@Column(name="stadd")
Private String address
//setter and getter
}

```

**Note:**

We can apply mapping annotations either on the top of domain class property or domain class getter method if domain class name as same table name and domain class property name same as column name then specify @Table, @Column annotation optional

 HBProj15(Basic App with Annotations)

```

└── src
    ├── com.nt.cfgs
    │   └── hibernate.cfg.xml
    ├── com.nt.domain
    │   └── Student.java
    ├── com.nt.test
    │   └── ClientApp.java
    └── com.nt.utility
        └── HibernateUtil.java

```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property
            name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

        <mapping class="com.nt.domain.Student"/>

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

```

Student.java

```
package com.nt.domain;

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

@Entity
@Table(name="Student_Tab")
public class Student {
    @Id
    @Column(name="stno")
    private int sno;
    @Column(name="stname")
    private String sname;
    @Transient
    private String address;

    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getName() {
        return sname;
    }
    public void setName(String sname) {
        this.sname = sname;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    @Override
    public String toString() {
        return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";
    }
}
```

**HibernateUtil.java**

Same as above

**ClientApp.java**

```
package com.nt.test;

import org.hibernate.Transaction;

import org.hibernate.Session;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class ClientApp {
    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();
        //Save object
        Student st=new Student();
        st.setSno(1045);
        st.setSname("raja2");
        st.setAddress("hyd2");
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            int idVal=(Integer)ses.save(st);
            System.out.println("Generated Id value"+idVal);
            tx.commit();
            System.out.println("Object Saved");
        } //try
        catch(Exception e){
            tx.rollback();
        }
        /* //Load obj operation
        Student st=(Student)ses.get(Student.class,1001);
        if(st!=null)
            System.out.println(st); */

        //close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

**Note:**

There are no annotation given for alternate for hibernate configuration because by changing that xml file configure we can make our hibernate persistence logic, portable across the multiple Database software.

**Internal flow:**

When annotation driven base Hibernate application will executed

*Configuration cfg = new Configuration();*

Activates the hibernate framework by using the jar file that are added to build path

*cfg=cfg.configure("/hibernate.cfg.xml");*

Hibernate framework load hibernate configuration file and checks whether that file is well formed or valid file or not. If not exception will be thrown otherwise reads the configuration file data and prepares in memory meta data it read in memory meta data and check for mapping tag having whether class attribute or resource attribute if class attribute available load the specify domain class and gets all the annotation detail by using reflection api and prepare in memory data as extension to existing in memory metadata based on the mapping annotation and lastly return configuration object representing that in memory meta data.

**@Transient:**

To make contain property of domain class not participating any kind of persistence operation we can annotate the property by using @Transient annotation:

**@Column(name="stadd")**

**@Transient**

Private String address;

**Generators configuration using annotations:**

**@Id**

**@GenericGenerator**

**@GeneratedValue**

**@Column**

**@parameter:**

**@GenericGenerator:**

Hibernate annotation that allows configuring increment generator having logical name

**@GeneratedValue:**

Assign the generator generated identity value as the identity filed property value.

**@Id:**

Configure sno property as identify filed property.

**@parameter:**

It useful to supply parameter name and value that is required for various operation of hibernate we can pass annotation as the value of parameter belonging to another annotations.

**Sequence:**

Use sequence of underlying DATABASE software like oracle to generate the identity value.

Create sequence oracle:

Create sequence student\_no\_sequence increment by 5 starts with 100;

**Sample Code:**

@Id

```
@GenericGenerator(name = "gen1", strategy = "sequence",
parameters = @Parameter(name = "sequence", value = "student_no_sequence"))
@GeneratedValue(generator = "gen1")
@Column(name = "stno")
```

 HBProj16(Anno-Generators)

- ▶  src
  - ▶  com.nt.cfgs
    - ☒ hibernate.cfg.xml
  - ▶  com.nt.domain
    - ▷ J Student.java
  - ▶  com.nt.test
    - ▷ J ClientApp.java
  - ▶  com.nt.utility
    - ▷ J HibernateUtil.java

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property
name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

        <mapping class="com.nt.domain.Student"/>
    </session-factory>
</hibernate-configuration>
```

Student.java

```
package com.nt.domain;

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

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name = "student_Tab")
public class Student {
    @Id
    @GenericGenerator(name = "gen1", strategy = "sequence",
        parameters = @Parameter(name = "sequence", value = "student_no_sequence"))
    @GeneratedValue(generator = "gen1")
    @Column(name = "stno")
    private int sno;

    @Column(name = "stname")
    private String sname;
    @Column(name = "stadd")
    private String address;

    // write setters and getters

    public int getSno() {
        return sno;
    }

    public void setSno(int sno) {
        this.sno = sno;
    }

    public String getName() {
        return sname;
    }

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

    public String getAddress() {
        return address;
    }
}
```

```
public void setAddress(String address) {
this.address = address;
}

@Override
public String toString() {
return "Student [sno=" + sno + ", sname=" + sname + ", address="
+ address + "]";
}

}
```

**HibernateUtil.java**

```
//same as previous
```

**ClientApp.java**

```
package com.nt.test;

import org.hibernate.Transaction;

import org.hibernate.Session;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class ClientApp {
public static void main(String[] args) {
// get Session
Session ses=HibernateUtil.getSession();
//Save object
Student st=new Student();
st.setSno(1045);
st.setSname("raja2");
st.setAddress("hyd2");
Transaction tx=null;
try{
tx=ses.beginTransaction();
int idVal=(Integer)ses.save(st);
System.out.println("Generated Id value"+idVal);
tx.commit();
System.out.println("Object Saved");
}//try
catch(Exception e){
tx.rollback();
}
/* //Load obj operation
Student st=(Student)ses.get(Student.class,1001);
```

```

if(st!=null)
System.out.println(st); */

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}
}

```

## **Annotation driven JPA generators:**

JPA Generators are useful to generate id values for domain class objects providing portability across multiple ORM S/W.

The Generators are Identity, Sequence, Table and Auto

### **Identity:**

Works in DB S/w that support identity columns (auto increment columns) use <Previous value+1> formula.

Example:

- Make application pointing to MySQL DB s/w (Annotation driven example)
- Create db table "student\_tab" in MySQL having "PK" and "autoincrement" (identity column) constraints on db table column "stno".
- Configure "IDENTITY" generator in Domain class as shown below.

### **In Domain class:**

```

@Id
@Column (name="stno")
@GeneratedValue (strategy=Generation type. IDENTITY)
private int sno;

```

### **Sequence:**

Create sequence with given name & parameters to generate the identity value, works in that DB S/W which support sequences.

Example:

- make sure that APP is pointing to oracle DB S/W.
- Configure "sequence" generator as shown below

### **In Domain class:**

```

@Id
@Column (name="stno")
@Sequence Generator (name="gen1", sequenceName="stno_seq1", initialValue=10,
allocationSize=20)
@GeneratedValue (generator="gen1", strategy=generation type. SEQUENCE)
private int sno;
➤ The generated ID values are 10, 31, 51, 71, 91 ...

```

### **Table:**

Uses given table info & parameters to generate the identity value

We need to configure helper table name, PK column name, initial value & allocation size and etc, parameters to generate the identity values.

Example:

- a. point to any database software.
- b. Configure table generator as shown below in domain class

**In domain class:**

```

@ID
@Column (name="stno")
@Table Generator (name="gen1", table="ID-GEN", PK column Name="PK_col", value column
Name="PK_VAL", PK column value= "STNO", initial value=100, allocation size=5)
@GeneratedValue (generator="gen1", strategy=Generation type. TABLE) Private int sno;
➤ The generated values are 101 107,112,117,122.....
```

**Auto:**

Pics up the generator based on the capabilities of underlying DB S/w. In case of Oracle it uses sequence (if no sequence is configured it takes hibernate-sequence as default sequence).  
in case of MySQL it uses Hilo generator.

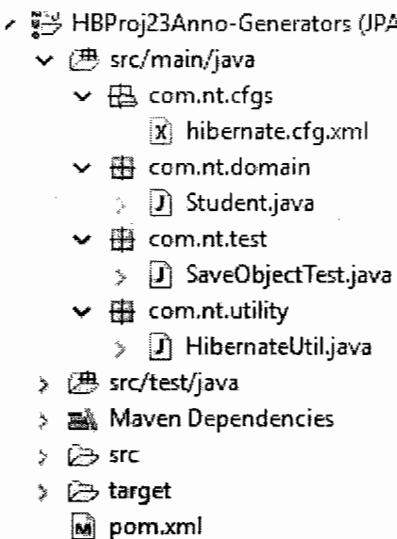
Example:

```

@Id
@Column (name="stno")
@GeneratedValue (Strategy=Generation type . AUTO) private int sno;
```

**Note:**

While working with sequence, Table, Auto JPA generator configure to "hbm2.ddl.auto" property with value "update" to create given sequences, tables dynamically.

**hibernate.cfg.xml:-**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <!-- <session-factory>
        connection 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>
<property name="hbm2ddl.auto">update</property>
Dialect cfg
<property
name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
mapping file
<mapping class="com.nt.domain.Student"/>
</session-factory> -->

<session-factory>
<!-- connection properties -->
<property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="connection.url">jdbc:mysql://nthb98db</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>
<!-- Dialect cfg -->
<property
name="dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hbm2ddl.auto">update</property>
<!-- mapping file -->
<mapping class="com.nt.domain.Student"/>
</session-factory>
</hibernate-configuration>
```

**Student.java:-**

```
package com.nt.domain;

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

@Entity
@Table(name = "Student_Tab")
public class Student {

    /*@Id
    @Column(name="stno")
    @GeneratedValue(strategy=GenerationType.IDENTITY)*/

    /*@Id
    @Column(name="stno")
```

```
@SequenceGenerator(name="gen1",
                     sequenceName="stno_seq1",
                     initialValue=10,
                     allocationSize=20
)
@GeneratedValue(generator="gen1",
               strategy=GenerationType.SEQUENCE)*/

/*
@Id
@Column(name="stno")
@TableGenerator(name="gen1",
               table="ID_GEN",
               pkColumnName="PK_COL",
               valueColumnName="PK_VAL",
               pkColumnValue="STNO",
               initialValue=100,
               allocationSize=5
)
@GeneratedValue(generator="gen1",
               strategy=GenerationType.TABLE)*/

@Id
@Column(name="stno")
@GeneratedValue(strategy=GenerationType.AUTO)
private int sno;

@Column(name = "stname")
private String sname;
@Column(name = "stadd")
// @Transient
private String sadd;

public int getSno() {
    return sno;
}

public void setSno(int sno) {
    this.sno = sno;
}

public String getSname() {
    return sname;
}

public void setSname(String sname) {
    this.sname = sname;
}

public String getSadd() {
    return sadd;
}
```

```
public void setSadd(String sadd) {
    this.sadd = sadd;
}

@Override
public String toString() {
    return "Student [sno=" + sno + ", sname=" + sname + ", sadd="
+ sadd + "]";
}

}
```

**SaveObjectTest.java:-**

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

    public static void main(String[] args) {
        Session ses=null;
        Transaction tx=null;
        Student st=null;
        //get Session
        ses=HibernateUtil.getSession();
        //prepare Domain class object
        //save object
        st=new Student();
        st.setSno(904);st.setSname("rajesh");st.setSadd("vizag");
        try{
            tx=ses.beginTransaction();
            int idVal=(int)ses.save(st);
            System.out.println("id value::"+idVal);
            tx.commit();

        }//try
        catch(Exception e){
            tx.rollback();
        }
        //close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```

**HibernateUtil.java:-**

```
package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        System.out.println("Session obj hashCode:::"+ses.hashCode());
        return ses;
    }//getSession()

    public static void closeSession(){
        Session ses=null;

        ses=threadLocal.get();
        if(ses!=null){
            ses.close();
            threadLocal.remove();
        }
    }

    public static void closeSessionFactory(){
        factory.close();
    }

}//class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
<groupId>nit</groupId>
<artifactId>HBProj12-Generator</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>HBProj12-Generator</name>
<url>http://maven.apache.org</url>

<properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>
    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.1.0.Final</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-
java -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.36</version>
        </dependency>

    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-proxool -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-proxool</artifactId>
            <version>5.1.0.Final</version>
        </dependency>

    </dependencies>
</project>
```

**Versioning in hibernate using annotation:**

Use @Version annotation to configure special property in domain class that keeps track versioning

**Example:**

Add additional column in Database table to keep track of versioning of type number.

Sql> alter table student\_tab add ver\_col number (5);

Add extra property in domain having @Version annotation to keep track of versioning of the object.

In Student.java

```
@Version  
@Column(name="ver_col")  
private int ver;
```

**Time stamping using annotations:**

This feature keeps the track of the last updated date and time object using hibernate persistence logic use @Temporal annotation mark the property that holds time stamp information.

**Note:**

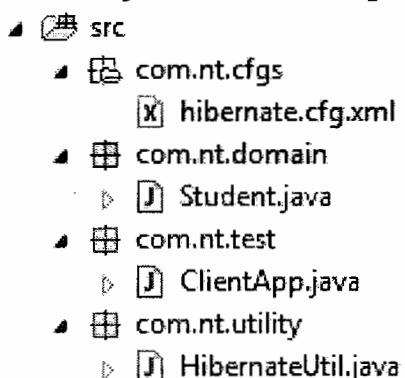
While working with xml configuration we can't apply both versioning and timestamp feature on single object at a time but this is possible while working with annotation configuration.

- Add Additional extra column in Database table of type timestamp

Sql> alter table add student\_tab last\_updated timestamp;

- Add extra property in domain class of type java.util.Date having

```
@Temporal(value=TemporalType.DATE)  
@Column (name="last_updated")  
private java.util.Date lstUpd=new Date();
```

**Example Application on Versioning and Time Stamping using annotation:**

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property
            name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.Student"/>
    </session-factory>
</hibernate-configuration>
```

**Student.java**

```
package com.nt.domain;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;

@Entity
@Table(name="student_Tab")
public class Student {
    @Id
    @Column(name="stno")
    private int sno;
    @Column(name="stname")
    private String sname;
    @Column(name="stadd")
    private String address;
    @Version
    @Column(name="ver_col")
    private int ver;
```

```
@Temporal(value=TemporalType.DATE)
@Column(name="last_updated")
private java.util.Date lstUpd=new Date();

public java.util.Date getLstUpd() {
return lstUpd;
}

public void setLstUpd(java.util.Date lstUpd) {
this.lstUpd = lstUpd;
}

public int getVer() {
return ver;
}

public void setVer(int ver) {
this.ver = ver;
}

//write setters and getters
public int getSno() {
return sno;
}

public void setSno(int sno) {
this.sno = sno;
}

public String getSname() {
return sname;
}

public void setSname(String sname) {
this.sname = sname;
}

public String getAddress() {
return address;
}

public void setAddress(String address) {
this.address = address;
}

@Override
public String toString() {
return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";
}

}
```

HibernateUtil.java

```
package com.nt.utility;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.hibernate.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal<Session>();
    private static SessionFactory factory;
    static{
        Configuration cfg=null;
        StandardServiceRegistryBuilder builder=null;
        ServiceRegistry registry=null;
        //Create SessionFactory obj
        cfg=new Configuration();
        cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        builder=new StandardServiceRegistryBuilder();
        registry=builder.applySettings(cfg.getProperties()).build();
        factory=cfg.buildSessionFactory(registry);
        System.out.println("SessionFactory:"+factory.hashCode());
    }

    public static Session getSession(){
        Session session=null;
        if(threadLocal.get()==null){
            session=factory.openSession();
            threadLocal.set(session);
        }
        session=threadLocal.get();
        System.out.println(Thread.currentThread().hashCode()+"---→"+session.hashCode());
        return session;
    } //getSession()

    public static void closeSession(){
        Session session=null;
        session=threadLocal.get();
        session.close();
        threadLocal.remove();
    }

    public static void closeSessionFactory(){
        factory.close();
    }
}
```

HibernateUtil.java

same as previous

ClientApp.java

```
package com.nt.test;

import org.hibernate.Transaction;
import org.hibernate.Session;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class ClientApp {
    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();
        //Save object
        Student st=new Student();
        st.setSno(1048);
        st.setSname("raja2");
        st.setAddress("hyd2");
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            int idVal=(Integer)ses.save(st);
            System.out.println("Generated Id value"+idVal);
            tx.commit();
            System.out.println("Object Saved");
        } //try
        catch(Exception e){
            tx.rollback();
        }
        /* //Load obj operation
        Student st=(Student)ses.get(Student.class,1046);
        if(st!=null)
            System.out.println(st); */

        /* // Load and modify the obj
        Student st=(Student) ses.get(Student.class,1046);
        if(st!=null){
            Transaction tx=null;
            try{
                tx=ses.beginTransaction();
                st.setAddress("vizag");
                tx.commit();
            } 
```

```

    catch(Exception e){
        tx.rollback();
    }
}//if */

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}
}
}

```

▼ HBProj22Anno-@Version+@Temporal-Timestamp

- ▼ src/main/java
  - ▼ com.nt.cfgs
    - hibernate.cfg.xml
  - ▼ com.nt.domain
    - Student.java
  - ▼ com.nt.test
    - SaveObjectTest.java
    - UpdateObjectTest.java
  - ▼ com.nt.utility
    - HibernateUtil.java
- > src/test/java
- > Maven Dependencies
- > Referenced Libraries
- > JRE System Library [jre1.8.0\_92]
- > src
- > target
- pom.xml

**hibernate.cfg.xml:-**

```

<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- connection 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>
        <!-- Dialect cfg -->
        <property
name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

```

```
<!-- mapping file -->
<mapping class="com.nt.domain.Student"/>
</session-factory>
</hibernate-configuration>
```

**Student.java:-**

```
package com.nt.domain;

import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;

@Entity
@Table(name = "Student_Tab")
public class Student {
    @Id
    @Column(name = "stno")
    private int sno;
    @Column(name = "stname")
    private String sname;
    @Column(name = "stadd")
    // @Transient
    private String sadd;

    @Version
    @Column(name = "ver_col")
    private int ver;
    @Column(name="lastUpdated")
    @Temporal(value=TemporalType.DATE)
    private Date lastUpdated=new Date();

    public Date getLastUpdated() {
        return lastUpdated;
    }

    public void setLastUpdated(Date lastUpdated) {
        this.lastUpdated = lastUpdated;
    }

    public int getVer() {
        return ver;
    }

    public void setVer(int ver) {
        this.ver = ver;
    }
}
```

```
public int getSno() {
    return sno;
}

public void setSno(int sno) {
    this.sno = sno;
}

public String getSname() {
    return sname;
}

public void setSname(String sname) {
    this.sname = sname;
}

public String getSadd() {
    return sadd;
}

public void setSadd(String sadd) {
    this.sadd = sadd;
}

@Override
public String toString() {
    return "Student [sno=" + sno + ", sname=" + sname + ", sadd=" +
+ sadd + "]";
}

}
```

**SaveObjectTest.java:-**

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class SaveObjectTest {

    public static void main(String[] args) {
        Session ses=null;
        Transaction tx=null;
        Student st=null;
        //get Session
        ses=HibernateUtil.getSession();
        //prepare Domain class object
    }
}
```

```
//save object
st=new Student();
st.setSno(904);st.setSname("rajesh");st.setSadd("vizag");
try{
    tx=ses.beginTransaction();
    ses.save(st);
    tx.commit();
    System.out.println("object saved---");
>ver::"+st.getVer()+"Date:"+

st.getLastUpdated());
    }//try
catch(Exception e){
    tx.rollback();
}
//close objects
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class
```

**UpdateObjectTest.java:-**

```
package com.nt.test;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class UpdateObjectTest {

    public static void main(String[] args) {
        Session ses=null;
        Transaction tx=null;
        Student st=null;
        //get Session
        ses=HibernateUtil.getSession();
        //load object
        st=ses.get(Student.class,904);
        //update object
        try{
            tx=ses.beginTransaction();
            st.setSadd("mumbai5");
            ses.update(st);
            tx.commit();
            System.out.println("object is updated---");
>ver::"+st.getVer()+"on date"+

st.getLastUpdated());
        }//try
    }
}
```

```

        catch(Exception e){
            tx.rollback();
        }
        //close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class

```

**HibernateUtil.java:-**

```

package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").

buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        System.out.println("Session obj hashCode:::"+ses.hashCode());
        return ses;
    }//getSession()

    public static void closeSession(){
        Session ses=null;

        ses=threadLocal.get();
        if(ses!=null){
            ses.close();
            threadLocal.remove();
        }
    }

    public static void closeSessionFactory(){
        factory.close();
    }
}

```

```
 }  
}//class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0  
    http://maven.apache.org/xsd/maven-4.0.0.xsd">  
    <modelVersion>4.0.0</modelVersion>  
  
    <groupId>nit</groupId>  
    <artifactId>HBProj12-Generator</artifactId>  
    <version>0.0.1-SNAPSHOT</version>  
    <packaging>jar</packaging>  
  
    <name>HBProj12-Generator</name>  
    <url>http://maven.apache.org</url>  
  
    <properties>  
        <project.build.sourceEncoding>UTF-  
8</project.build.sourceEncoding>  
    </properties>  
  
    <dependencies>  
        <dependency>  
            <groupId>junit</groupId>  
            <artifactId>junit</artifactId>  
            <version>3.8.1</version>  
            <scope>test</scope>  
        </dependency>  
        <!--  
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->  
        <!--  
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->  
        <dependency>  
            <groupId>org.hibernate</groupId>  
            <artifactId>hibernate-core</artifactId>  
            <version>5.1.0.Final</version>  
        </dependency>  
        <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-  
java -->  
        <dependency>  
            <groupId>mysql</groupId>  
            <artifactId>mysql-connector-java</artifactId>  
            <version>5.1.36</version>  
        </dependency>  
  
        <!--  
https://mvnrepository.com/artifact/org.hibernate/hibernate-proxool -->
```

```

<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-proxool</artifactId>
    <version>5.1.0.Final</version>
</dependency>

</dependencies>
</project>

```

**Q. Why annotation are not given as alternate to hibernate configuration file?**

Ans: Hibernate configuration file contains JDBC properties files that are required for DATABASE connectivity and these can't be placed in multiple domain, DAO class so annotation are not given as alternate to hibernate configuration file in hibernate more ever through xml file we will get the flexibility of modification to change from one Database table software to another DATABASE software.

**Composite id filed configuration using annotation:**

@Embeddable: to mark class that contains multiple properties to combine as composite identity filed.

@Embedded: to mark embeddable class reference variable as composite identity filed in main domain class.

HBProj18(Anno-Composite Id Field)

```

▲ □ src
  ▲ □ com.nt.cfgs
    □ hibernate.cfg.xml
    □ programmer.hbm.xml
  ▲ □ com.nt.domain
    ▶ □ PrgmrProjId.java
    ▶ □ ProgrammerProject.java
  ▲ □ com.nt.test
    ▶ □ CPKClientApp.java
  ▲ □ com.nt.utility
    ▶ □ HibernateUtil.java

```

**hibernate.cfg.xml**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
<property name="connection.password">tiger</property>
<property name="connection.pool_size">15</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping class="com.nt.domain.ProgrammerProject"/>
</session-factory>
</hibernate-configuration>
```

**PrgmrProjId.java**

```
package com.nt.domain;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
public class PrgmrProjId implements Serializable {
    @Column(name="prgmr_id")
    private int prgmrlId;
    @Column(name="proj_id")
    private int projId;
    //write setters and getters
    public int getPrgmrlId() {
        return prgmrlId;
    }
    public void setPrgmrlId(int prgmrlId) {
        this.prgmrlId = prgmrlId;
    }
    public int getProjId() {
        return projId;
    }
    public void setProjId(int projId) {
        this.projId = projId;
    }
    @Override
    public String toString() {
        return "PrgmrProjId [prgmrlId=" + prgmrlId + ", projId=" + projId + "]";
    }
}
```

**ProgrammerProject.java**

```
package com.nt.domain;
import javax.persistence.Column;
import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import javax.persistence.Table;

@Entity
@Table(name="programmers_projects")
public class ProgrammerProject {
    @EmbeddedId
    private PrgmrProjId id;
    @Column(name="prgmr_name")
    private String prgmrName;
    @Column(name="proj_name")
    private String projName;
    private int salary;

    //setters and getters
    public PrgmrProjId getId() {
        return id;
    }
    public void setId(PrgmrProjId id) {
        this.id = id;
    }
    public String getPrgmrName() {
        return prgmrName;
    }
    public void setPrgmrName(String prgmrName) {
        this.prgmrName = prgmrName;
    }
    public String getProjName() {
        return projName;
    }
    public void setProjName(String projName) {
        this.projName = projName;
    }
    public int getSalary() {
        return salary;
    }
    public void setSalary(int salary) {
        this.salary = salary;
    }
    @Override
    public String toString() {
        return "ProgrammerProject [id=" + id + ", prgmrName=" + prgmrName +
               ", projName=" + projName + ", salary=" + salary +
               "]";
    }
}
```

```
}
```

**HibernateUtil.java**

```
Same as previous
```

**CPKClientApp.java**

```
package com.nt.test;

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

import com.nt.domain.PrgmrProjId;
import com.nt.domain.ProgrammerProject;
import com.nt.utility.HibernateUtil;

public class CPKClientApp {
    public static void main(String[] args) throws Exception {
        // get Session
        Session ses = HibernateUtil.getSession();
        // Save Obj

        PrgmrProjId id = new PrgmrProjId();
        id.setProjId(121);
        id.setPrgmrid(1221);

        ProgrammerProject pp = new ProgrammerProject();
        pp.setId(id);
        pp.setPrgmrName("karan");
        pp.setProjName("proj1");
        pp.setSalary(5000);
        Transaction tx = null;
        try {
            tx = ses.beginTransaction();
            PrgmrProjId idVal = (PrgmrProjId) ses.save(pp);
            System.out.println(idVal);
            tx.commit();
        } catch (Exception e) {
            tx.rollback();
        }

        /*
        // Load obj
        // prpare id value
        PrgmrProjId idVal = new PrgmrProjId();
        idVal.setPrgmrid(1221);
        idVal.setProjId(121);
        // Load obj
    }
}
```

```

ProgrammerProject pp = (ProgrammerProject) ses.load(ProgrammerProject.class, idVal);
System.out.println(pp);

*/
// close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}

}

```

**Note:**

There is no concept of taking on composite identity filed with generator

### **Bulk persistence operations in Hibernate**

To perform more than one record manipulation by using our choice value as the criteria value use following techniques.

1. HQL
2. Native SQL
3. Criteria API/QBC (Query By Criteria)

#### **HQL (Hibernate Query Language)**

- It is object based SQL.
- SQL queries will be written based on Database table and its columns whereas HQL Queries will be written based on domain class and its properties.
- Every HQL query will be converted into SQL query by using ASTQueryTranslator of hibernate
- It can be used for executing both select and non select operations.
- HQL query syntax is very similar to SQL query syntax.
- HQL query are DATABASE software independent queries i.e. their persistence logic is DATABASE software independent.
- HQL supports two types of parameters:
  1. Positional parameters (?)
  2. Named parameters (:<name>)
- HQL supports aggregate functions, condition clauses, relational operators, joins, sub queries and etc...
- Each HQL query will be represented by **Query object**( it is the object of java class that implements java.sql.Query(!))
- To execute HQL select query, use list() or iterate() method.
- To execute HQL non select query use executeUpdate() method.
- HQL insert query can't be used to insert record in DB table with direct values.
- HQL based DDL operations are not possible.
- HQL based PL/SQL programming is not possible

**Understanding the syntax of HQL Queries:**

```
SQL> select * from employee
HQL>from EmpDetails
      from EmpDetails as ed
      from EmpDetails ed
      select ed from EmpDetails ed
```

```
SQL> select * from employee where eid>=100 and eid<=200
HQL>from EmpDetails ed where ed.no>100 and ed.no<=200
      from EmpDetails where no>=100 and no<=200
```

```
SQL>select firstname, lastname form employee
HQL>select fname, lname from EmpDetails
HQL> select ed.fname, ed.lname from EmpDetails ed
```

→ If you are writing HQL select query to get all the property values from objects (all the column values from table) then select word in HQL Query is optional otherwise mandatory.

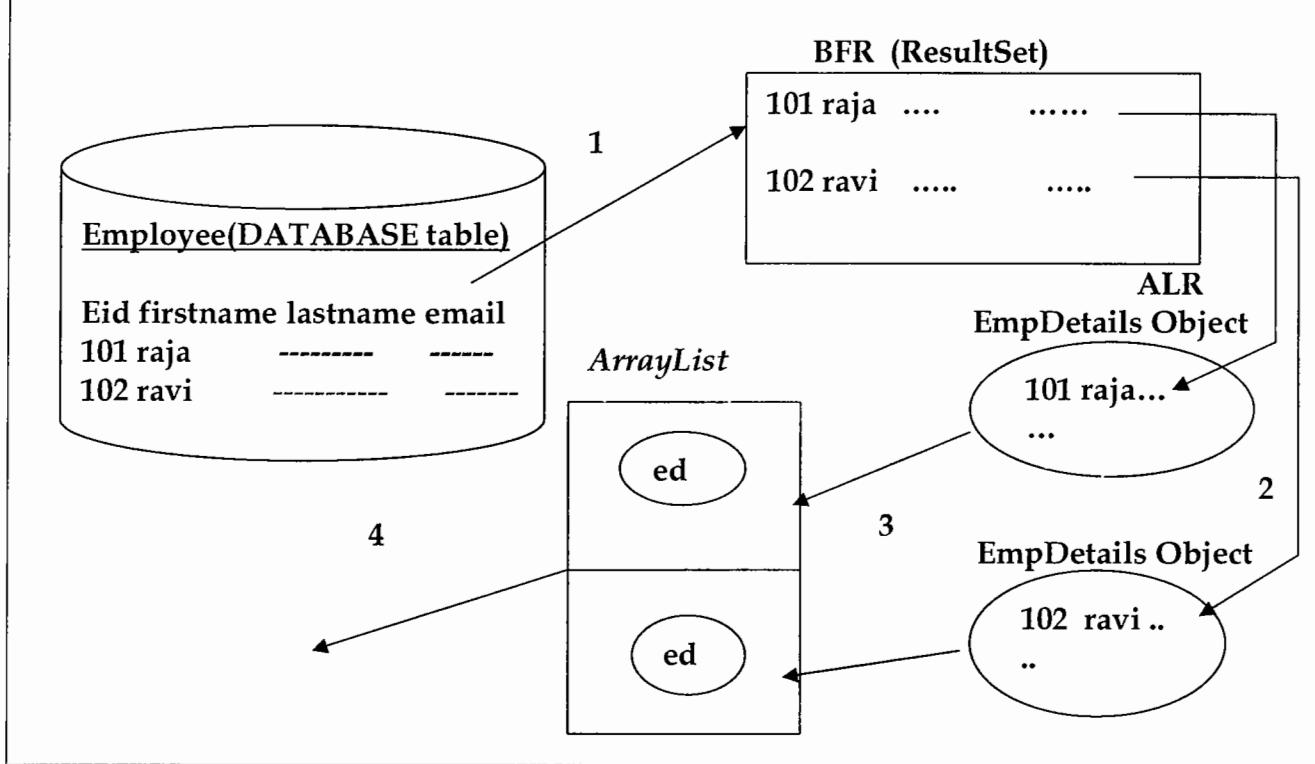
```
SQL> delete from employee where eid=100
HQL> delete from EmpDetails where no = 100;
HQL> delete from EmpDetails ed where ed.no = 100;
```

**Note:**

→ HQL keywords are not case sensitive, but domain class and properties are case sensitive.

**Create Query object representing HQL**

```
Query query=ses.createQuery("from EmpDetails");
//execute HQL
List<EmpDetails> list=query.list(); //List collection having objs of Domain class
//process List collection
for(EmpDetails ed:list){
    System.out.println(ed);
}
```



- Hibernate framework converts HQL query to Sql query to execute in DATABASE software and gets **ResultSet** object.
- Hibernate framework copies the record values of ResultSet object to the object of domain class.
- Framework copies the domain class objects to List collection and sends that **collection** to client application.

#### Executing HQL select query with positional params(?)

```
//(0 based index)
Query query = ses.createQuery("from EmpDetails where no>=? and no<=?");
//set param values
query.setInteger(0,100);
query.setInteger(1,200);

//execute Query
List<EmpDetails> list=query.list();
```

→ Positional parameter is **deprecated** in hibernate4.x, because there is a chance of getting confusion towards identifying based on indexes, so use either named parameters or JPA style possible parameters as alternate.

#### Named parameters:

Named parameters will have self description ,so setting values to parameters becomes easy.

Syntax:-> :<name>

```
Query query = ses.createQuery("from EmpDetails where no>=:min and no<=:max");
//set param values
query.setInteger("min",100);
```

```
query.setInteger("max",200);
//execute Query
List<EmpDetails> list=query.list();
```

**Executing HQL select query with JPA style named parameters:**

```
Query query= ses.createQuery("from EmpDetails where no>=?0 and no<=?1");
//set param values
query.setInteger("0",200);
query.setInteger("1",300);
//execute Query
List<EmpDetails> list=query.list();
//process the result
....
```

**Note:**We can't place positional parameters in HQL query after any named parameters have been defined.

*from EmpDetails where no>=? and no<=max(valid)  
from EmpDetails where no>=min and no<=? (invalid)*

**HQL query with both named and positional parameters**

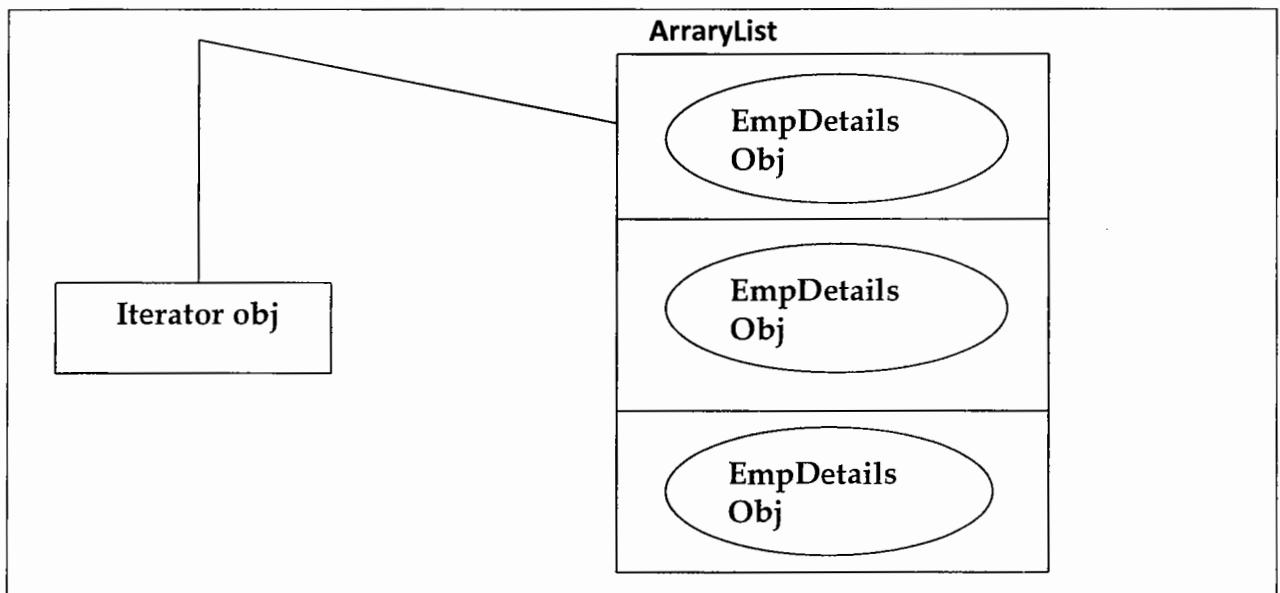
```
Query query = ses.createQuery("from EmpDetails where ? in(?,:name1,:name2)");
//set Param values
query.setString(0,"fname");
query.setString(1,"raja");
query.setString("name1","ravi");
query.setString("name2","mahesh");
//execute the Query
List<EmpDetails> list=query.list();
//process the result
.....
```

→We can place parameters in HQL query just representing input values that means you can't place parameters in HQL query representing domain class names HQL keywords, domain class property names.

- *from EmpDetails where no>=? (valid)*
- *from ? where ? >=? (invalid)*
- *from EmpDetails ? no>=? (invalid)*
- *? EmpDetails ? no>=? (invalid)*

**We can use the select queries with iterate () method.**

```
Query query=ses.createQuery("from EmpDetails where no>=:min");
query.setInteger("min",200);
//execute HQL select Query
Iterator<EmpDetails> it=query.iterate();
//process the results
while(it.hasNext()){
    EmpDetails ed=it.next();
    System.out.println(ed);
}
```



→**iterate()** method performs lazy loading of object/record ,first it generates select query to get only identity values and generates multiple select queries to get multiple records by submitting the above given multiple identify values.

**Note:**

Iterate () method performs lazy loading objects/records only when **HQL select query** retrieves all column values otherwise it performs eager loading becoz lazy loading is possible only for domain class objects and not possible for other objects.

**Q. What is the differences between list() and iterate () towards executing HQL select query that gives all column values of DATABASE table?**

list()	Iterate()
<ul style="list-style-type: none"> <li>1. Performs <b>eager loading</b>.</li> <li>2. Does not generates proxy object.</li> <li>3. To retrieve “n” records from DATABASE table it creates “n” objects for domain class</li> <li>4. Generates 1 select query to get “n” records.</li> <li>5. Directly returns List collection having columns values(In the form of domain class objects)</li> </ul> <p>Usecase:</p> <ul style="list-style-type: none"> <li>6. Useful to retrieve and display facebook Postings.</li> </ul>	<ul style="list-style-type: none"> <li>1. Performs <b>lazy loading</b>.</li> <li>2. Generates proxy object.</li> <li>3. To retrieve ” n” records from DATABASE table creates <math>2 \times n</math> objects . For dynamically created proxy class.</li> <li>4. Generate 1+n select query to get n record from database table.</li> <li>5. Returns iterator object that points the list collection having domain class object</li> </ul> <p>Usecase:</p> <ul style="list-style-type: none"> <li>6. Useful to retrieve and display facebook post commands.</li> </ul>

**Note:**

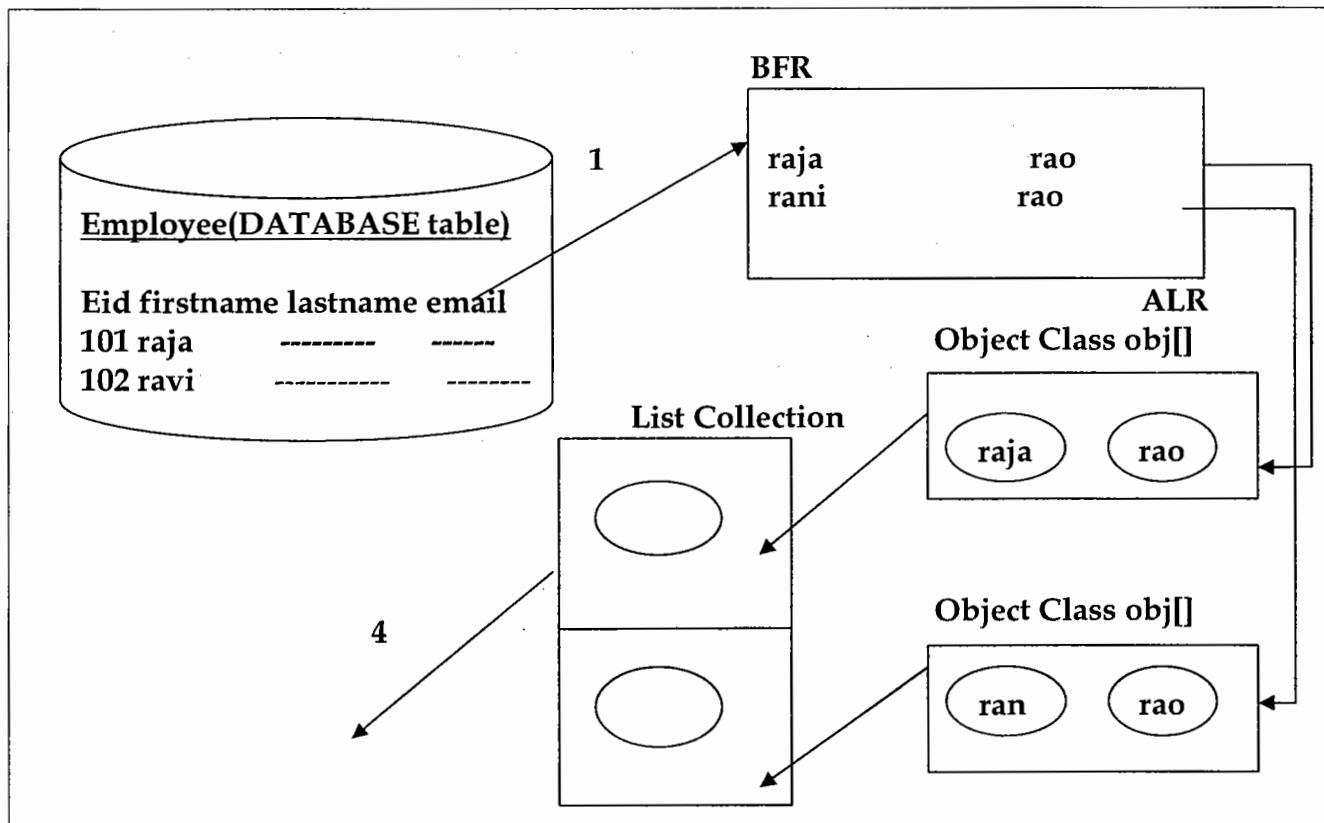
→ Use iterate() to execute select query when there is no guarantee that fetched data will be processed or not (facebook post related comments) use list() method when there is a guarantee of processing the fetched/retrieved data.

**Retrieving specific multiple column values from Database table**

```
Query query= ses.createQuery("select fname,lname from EmpDetails where mail like :domain");
//set param value
query.setString("domain","%star.com");
//execute HQL
List <Object[]>list=query.list();
//process the ResultSet
for(Object []row:list){
    for(Object val:row){
        System.out.print(val+" ");
    }
}
System.out.println();
}
```

→ Array is an object in java ,so it can be placed as the element value of the List collection.

→ when we execute HQL select query that retrieve multiple specific column values, we get List collection having arrays of type java.lang.Object as the element values. if HQL select query retrieves all the column values of DATABASE table then we get List collection having domain class objects as the element values.

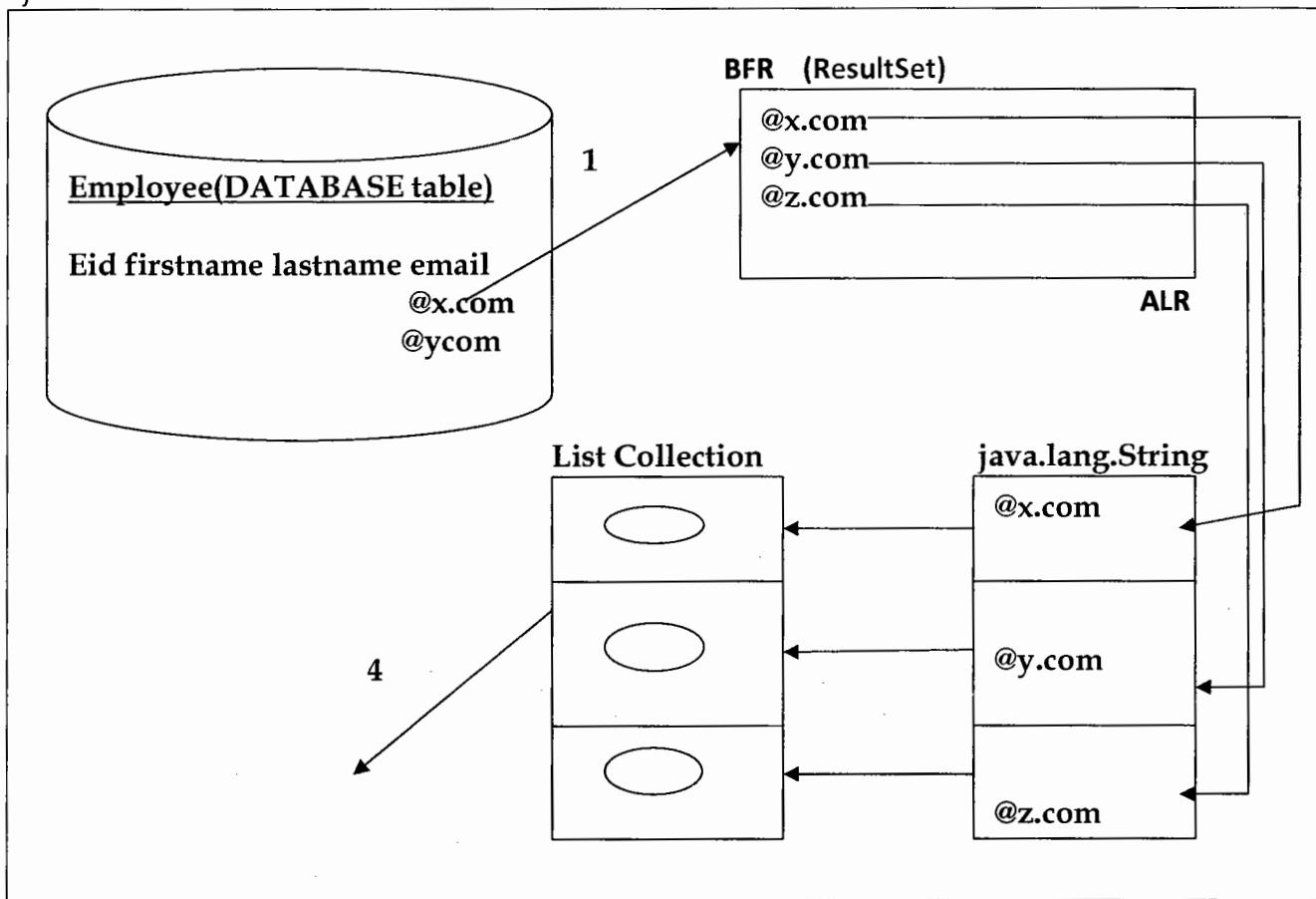


Retrieving specific single column value form DATABASE table using HQL select query

```

Query query=ses.createQuery("select mail from EmpDetails");
//execute HQL
List<String> list=query.list();
//process the List
for(String mail:list){
    System.out.println(mail+" ");
}

```



- When HQL select query retrieves specific single column value then column related property type objects will come as elements of the List collection.
- iterate() method performs lazy loading of object only HQL select query retrieves all the column values of DATABASE table i.e. when HQL select query retrieves all column values.If multiple specific column values are selected, then it doesn't perform lazy loading, because no domain class object will be handled there.

Executing HQL aggregate functions

```

Query query=ses.createQuery("select count(*) from EmpDetails");
//execute HQL
List <Long> list=query.list();
long count=(Long)list.get(0);
System.out.println("Records count"+count);

```

In hibernate mapping file you can specify hibernate data types as bridge data types to convert database notation data to java notation data.

**In mapping file:**

```
<id name="no" column="EID" type="int" />
<property name="fname" column="FIRSTNAME" type="string"/>
```

**Assignment :**

- write HQL query that gives highest emp number based EmpDetails
- Write HQL query that gives EmpDetails whose having 5<sup>th</sup> highest number.

**Example:**

```
Query query = ses.createQuery("from EmpDetails where no=(select max(no)from EmpDetails)");
List<EmpDetails>list=query.list();
EmpDetails ed=list.get(0);
System.out.println(ed);
```

We need to execute HQL non select SQL query by calling execute Update() this method return numeric value representing no of record that are affected.

**Executing HQL non-select queries:**

→ Non select HQL queries must be executed in transactional environment

```
Query query=ses.createQuery("delete from EmpDetails where mail like :domain");
```

```
//set param value
```

```
query.setString("domain","%star.com");
```

```
Transaction tx=null;
```

```
try{
```

```
tx=ses.beginTransaction();
```

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

```
tx.commit();
```

```
System.out.println("No.of records that are effected"+result);
```

```
}//try
```

```
catch(Exception e){
```

```
tx.rollback();
```

```
}
```

-> Execute update() method returns numeric value representing number of records that are effected becoz of non select query execution.

**Example application**

HBProj19(HQL Basic & Named HQL Queries)

```
▶ ⚙ src
  ▶ com.nt.cfgs
    ▶ Employee.hbm.xml
    ▶ hibernate.cfg.xml
  ▶ com.nt.domain
    ▶ EmpDetails.java
  ▶ com.nt.test
    ▶ HQLClientApp.java
    ▶ NamedHQLTest.java
  ▶ com.nt.utility
    ▶ HibernateUtil.java
```

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID" /><!-- Singular Id field -->
        <property name="fname" column="FIRSTNAME" length="20" not-null="true"/>
        <property name="lname" column="LASTNAME" length="20"/>
        <property name="mail" column="EMAIL" length="20" unique="true"/>
    </class>

</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }
}
```

```
public int getNo() {  
  
    return no;  
}  
public void setNo(int no) {  
  
    this.no = no;  
}  
public String getFname() {  
  
    return fname;  
}  
public void setFname(String fname) {  
  
    this.fname = fname;  
}  
public String getLname() {  
  
    return lname;  
}  
public void setLname(String lname) {  
  
    this.lname = lname;  
}  
public String getMail() {  
  
    return mail;  
}  
public void setMail(String mail) {  
  
    this.mail = mail;  
}  
  
@Override  
public String toString() {  
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

**HibernateUtil.java**

Same as previous

**HQLClientApp.java**

```
package com.nt.test;
```

```
import java.util.List;
```

```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class HQLClientApp{
public static void main(String[] args) {
//Get Session obj
Session ses=HibernateUtil.getSession();
/* //Create Query obj representing HQL
Query query=ses.createQuery("from EmpDetails");
//execute HQL
List<EmpDetails> list=query.list(); //List collection having objs of Domain class
//process List collection
for(EmpDetails ed:list){
System.out.println(ed);
} */

/* //Executing HQL Select Query with positional params (?)
//(0 based index)
Query query=
ses.createQuery("from EmpDetails where no>=?");
//set param values
query.setInteger(0,200);

//execute Query
List<EmpDetails> list=query.list();
//process the Result
for(EmpDetails ed:list){
System.out.println(ed);
} */

/* //HQL select Query with named parameters
Query query=
ses.createQuery("from EmpDetails where ? in (?,:name1,:name2)");
//set Param values
query.setString(0,"fname");
query.setString(1,"raja");
query.setString("name1","ravi");
query.setString("name2","mahesh");
//execute the Query
List<EmpDetails> list=query.list();
//process the result
for(EmpDetails ed:list){
System.out.println(ed);
```

```
}*/  
  
/*// Executing HQL select Query using iterate() method  
Query query=ses.createQuery("from EmpDetails where no>=:min");  
query.setInteger("min",200);  
//execute HQL select Query  
Iterator<EmpDetails> it=query.iterate();  
//process the results  
while(it.hasNext()){  
    EmpDetails ed=it.next();  
    ed.getFname();  
}  
*/  
/* //Retrieving specific multiple col values of DATABASE table using HQL select Query  
Query query=  
    ses.createQuery("select fname,iname from EmpDetails where mail like :domain");  
//set param value  
query.setString("domain","%star.com");  
//execute HQL  
List <Object[]>list=query.list();  
//process the ResultSet  
for(Object []row:list){  
    for(Object val:row){  
        System.out.print(val+" ");  
    }//for  
    System.out.println();  
}//for  
*/  
  
/* //Retrieving specific single col value from DATABASE table using HQL Select Query  
Query query=ses.createQuery("select mail from EmpDetails");  
//execute HQL  
List<String> list=query.list();  
//process the List  
for(String mail:list){  
    System.out.println(mail+" ");  
}  
*/  
/*//Retrieving specific single col value from DATABASE table using HQL Select Query  
Query query=ses.createQuery("select no from EmpDetails");  
//execute HQL  
Iterator<Integer> it=query.iterate();  
//process the List  
while(it.hasNext()){  
    int no=it.next();  
    System.out.println(no);  
}*/
```

```
/* //executing HQL aggregate function
Query query=ses.createQuery("select count(*) from EmpDetails");
//execute HQL
List <Long> list=query.list();
long count=(Long)list.get(0);
System.out.println("Records count"+count);
*/
// Executing HQL sub Queries
/* Query query=
ses.createQuery("from EmpDetails where no=(select max(no)from EmpDetails)");
List<EmpDetails>list=query.list();
EmpDetails ed=list.get(0);
System.out.println(ed);*/
/*
// Executing HQL non-select Query
Query query=
ses.createQuery("delete from EmpDetails where mail like :domain");
//set param value
query.setString("domain","%star.com");
Transaction tx=null;
try{
tx=ses.beginTransaction();
int result=query.executeUpdate();
tx.commit();
System.out.println("No.of records that are effected"+result);
}//try
catch(Exception e){
tx.rollback();
}
*/
/* Query query=
ses.createQuery("insert into EmpDetails values(:val1,:val2,:val3,:val4)");
query.setInteger("val1",4567);
query.setString("val2","raja");
query.setString("val3","rao");
Transaction tx=null;
try{
tx=ses.beginTransaction();
int result=query.executeUpdate();
tx.commit();
}//try
catch(Exception e){
tx.rollback();
}*/
//close session,SessionFactory
```

```
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
```

### HQL insert query:

→ There is no HQL insert query to insert record values directly to Database table, but HQL insert query is given to insert record into one Database table by selecting records from another DATABASE table.

→ **Insert into..... values.....** form of HQL insert query is not available but  
**insert into .....select from.....** form of HQL query is available.

We can use ses.save(-) method to insert a record to DATABASE table with explicitly supplied data.

### Wrong HQL insert query:

```
Query query = ses.createQuery("insert into EmpDetails values(:val1,:val2,:val3,:val4)");
```

HQL insert actually supports bulk insertion i.e. we can select records from 1 DATABASE table and we can insert them to another DATABASE table.

### Use case1:

→ If certain working employees want to become NGO members then we need to insert bulk records in **NGOMember** table by collecting records from **Employee** table.

### Use case 2:

In order to make old customers as the premium customers we need to insert records into **premium\_customer** DATABASE table by collecting them from Regular **customer** DATABASE table.

### HBProj20(HQLInsert)

```

    ▲ src
      ▲ com.nt.cfgs
        □ Employee.hbm.xml
        □ hibernate.cfg.xml
        □ NGOMember.hbm.xml
      ▲ com.nt.dao
        ▷ TransferEmpsDao.java
        ▷ TransferEmpsDaoFactory.java
        ▷ TransferEmpsDaolmpl.java
      ▲ com.nt.domain
        ▷ EmpDetails.java
        ▷ NGOMember.java
      ▲ com.nt.test
        ▷ ClientApp.java
      ▲ com.nt.utility
        ▷ HibernateUtil.java
  
```

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
        <mapping resource="com/nt/cfgs/NGOMember.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID" /><!-- Singular Id field -->
        <property name="fname" column="FIRSTNAME" />
        <property name="lname" column="LASTNAME" />
        <property name="mail" column="EMAIL" />
    </class>
</hibernate-mapping>
```

**NGOMember.hbm.xml**

```
<!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.domain.NGOMember" table="NGOMember">
        <id name="no" column="MID" /><!-- Singular Id field -->
        <property name="fname" column="FIRSTNAME" />
        <property name="lname" column="LASTNAME" />
        <property name="mail" column="EMAIL" />
    </class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
        this.mail = mail;
    }
}
```

```
@Override  
public String toString() {  
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

### NGOMember.java

```
package com.nt.domain;  
  
public class NGOMember {  
//domain class properties  
private int no;  
private String fname;  
private String lname;  
private String mail;  
  
public NGOMember(){  
System.out.println("NGOMember:0-param constructor");  
}  
  
public int getNo() {  
  
return no;  
}  
public void setNo(int no) {  
  
this.no = no;  
}  
public String getFname() {  
  
return fname;  
}  
public void setFname(String fname) {  
  
this.fname = fname;  
}  
public String getLname() {  
  
return lname;  
}  
public void setLname(String lname) {  
  
this.lname = lname;  
}
```

```
}

public String getMail() {

    return mail;
}

public void setMail(String mail) {

    this.mail = mail;
}

@Override
public String toString() {
    return "NGOMember [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
```

### TransferEmpsDao.java

```
package com.nt.dao;

public interface TransferEmpsDao {
    public int transferByEmailDomain(String domain);
}
```

### TransferEmpsDaoImpl.java

```
package com.nt.dao;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.utility.HibernateUtil;

public class TransferEmpsDaoImpl implements TransferEmpsDao {
    private static final String HQL_TRANSFER_EMPS="INSERT INTO
NGOMember(no,fname,lname,mail) SELECT no,fname,lname,mail FROM "+
            " EmpDetails WHERE mail LIKE :domain";

    @Override
    public int transferByEmailDomain(String domain) {
        Session ses=null;
        int result=0;
        Transaction tx=null;
        Query query=null;
        //get Session
        ses=HibernateUtil.getSession();
        // execute bulk insert Query
        query=ses.createQuery(HQL_TRANSFER_EMPS);
        query.setString("domain", "%" + domain);
        try{
```

```
tx=ses.beginTransaction();
result=query.executeUpdate();
tx.commit();
}//try
catch(Exception e){
tx.rollback();
}
return result;
}//method

}//class
```

```
package com.nt.dao;

public class TransferEmpsDaoFactory {

    public static TransferEmpsDao getInstance(){
        return new TransferEmpsDaoImpl();
    }
}
```

#### HibernateUtil.java

```
package com.nt.utility;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;
import org.cfg.Configuration;
import org.hibernate.service.ServiceRegistry;

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal<Session>();
    private static SessionFactory factory;

    static{
        Configuration cfg=null;
        StandardServiceRegistryBuilder builder=null;
        ServiceRegistry registry=null;
        //Create SessionFactory obj
        cfg=new Configuration();
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
        builder=new StandardServiceRegistryBuilder();
        registry=builder.applySettings(cfg.getProperties()).build();
        factory=cfg.buildSessionFactory(registry);
    }
}
```

```

public static Session getSession(){

Session session=null;
if(threadLocal.get()==null){
session=factory.openSession();
threadLocal.set(session);
}
session=threadLocal.get();
return session;
}//getSession()

public static void closeSession(){

Session session=null;
session=threadLocal.get();
session.close();
threadLocal.remove();
}

public static void closeSessionFactory(){
factory.close();
}
}

```

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.TransferEmpsDao;
import com.nt.dao.TransferEmpsDaoFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {
public static void main(String[] args) {
// get DAO
TransferEmpsDao dao=TransferEmpsDaoFactory.getInstance();
// call the method
int cnt=dao.transferByEmailDomain("gmail.com");
System.out.println("No.of records that are copied"+cnt);
//close HB Session
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

**Q. Why HQL insert query is not given insert record with direct values?**

Ans: Since `session.save(-)` is given doing that work, so there is no need of separate HQL insert query for the same.

## Named HQL queries

- So far we created **Query object** based on one Hibernate Session object i.e. HQL query is very much specific to one hibernate Session object i.e. we can't use that HQL query in multiple Session objects of multiple client applications or **multiple DAO classes**.
- To make HQL query visible in multiple Session objects of 1 or more client applications or 1 or more DAO classes, we need to use named HQL query placed in hibernate mapping file. Named HQL queries also gives the flexibility of modification with out touching the java code.
- Named HQL query will be placed in mapping file having logical name by using <query> tag.

eg: 1. Place named HQL query in mapping file

```
<query name="GET_EMPS_BY_NAMES">
    from EmpDetails where mail like:domain
</query>
```

2. Access the Named HQL query either in client application or in DAO class and execute that query

```
//get Access to Named HQL Query
Query query=ses.getNamedQuery("GET_EMPS_BY_DOMAIN");
//set Parameter value
query.setString("domain","%yahoo.com");
//execute HQL
List<EmpDetails> list=query.list();
//process List
for(EmpDetails ed:list){
    System.out.println(ed)
}
```

->Generally we write HQL queries in DAO class at the top as string constant values, but in hibernate application we prefer writing them in mapping file (or) in the domain class by taking them as Named HQL queries.

### Use cases for Named HQL Query:

→ Authentication logic is required in multiple modules of the project instead of writing HQL query of that logic in DAO class of every module, it is recommended to write in hibernate mapping file only for one time as **named HQL query**, so that we can use that HQL query in multiple modules of project(or in multiple DAOs).

→ While placing **named HQL query** in mapping file, we must aware about less than symbol (<) because we used in it conditional operator but it will be taken as "<" tag symbol(start of sub tag) to resolve this problem we can use either &lt; or we can use <![CDATA[ .....]]>

```
<query name="DELETE_EMPS_BY_RANGE" >
<![CDATA[
    from EmpDetails where no>=:start and no <=:end
  ]]>
</query>
(Or)
<query name="DELETE_EMPS_BY_RANGE">
  Delete from EmpDetails where no>=:start and no &lt;=:end
</query>
```

ExampleApplication:

```
<!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.domain.EmpDetails" table="Employee">
    <id name="no" column="EID"/><!-- Singular Identifier/identity field -->
    <property name="fname" column="FIRSTNAME" length="20" type="string" />
    <property name="lname" column="LASTNAME" length="20" type="string" />
    <property name="mail" column="EMAIL" length="20" type="string" unique="true" not-null="true"/>
  </class>
  <query name="DELETE_EMPS_BY_RANGE ">
    <![CDATA[
      Delete from EmpDetails where no>=:start and no <=:end
    ]]>
  </query>
</hibernate-mapping>
```

ClientApp.java

```
package com.nt.test;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class ClientApp {

  public static void main(String[] args) {
    // get Session
    Session ses=HibernateUtil.getSession();

    // get Named HQL Query
    Query query=ses.getNamedQuery("DELETE_EMPS_BY_RANGE ");
    //set Param values
    query.setInteger("start",200);
    query.setInteger("end",500);
    // Execute the Query
    Transaction tx = null;
    try{
      tx = ses.beginTransaction();
      int result=query.executeUpdate();
```

```

        tx.commit();
        System.out.println("record:::"+result);
    }
    catch(Exception e){
        tx.rollback();
    }
//close Session
HibernateUtil.closeSession();
//close SessionFactory
HibernateUtil.closeSessionFactory();
}

}

```

✓ HBProj27-NamedHQL Queries  
 ✓ src  
   ✓ com.nt.cfgs  
     ✗ Employee.hbm.xml  
     ✗ hibernate.cfg.xml  
   ✓ com.nt.domain  
     > EmpDetails.java  
   ✓ com.nt.test  
     > NamedHQLTest.java  
       > NamedHQLTest  
   ✓ com.nt.utility  
     > HibernateUtil.java  
 > JRE System Library [jre1.8.0\_92]  
 > Referenced Libraries

**Employee.hbm.xml:-**

```

<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
  <!-- O-R mapping cfgs -->
  <class name="com.nt.domain.EmpDetails" table="Employee">
    <id name="no" column="EID" type="int"/>
    <property name="fname">
      <column name="FIRSTNAME"/>
    </property>
    <property name="lname" column="LASTNAME"/>
    <property name="mail" column="EMAIL"/>
  </class>
  <query name="GET_EMPS_By_DOMAIN">
    from EmpDetails where mail like :domain
  </query>
  <query name="GET_EMPS_BY_RANGE">
    <!-- from EmpDetails where no>=:min and no<=:max -->
    <![CDATA[
      from EmpDetails where no>=:min and no<=:max
    ]]>
  </query>
</hibernate-mapping>

```

```
</query>
<query name="UPDATE_EMP_LASTNAME">
    update EmpDetails set mail=:newMail where no=:id
</query>
</hibernate-mapping>
```

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/EmpLOYEE.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

**EmpDetails.java:-**

```
package com.nt.domain;

public class EmpDetails {
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
}
```

```
public void setFname(String fname) {
    this.fname = fname;
}
public String getLname() {
    return lname;
}
public void setLname(String lname) {
    this.lname = lname;
}
public String getMail() {
    return mail;
}
public void setMail(String mail) {
    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ",
lname=" + lname + ", mail=" + mail + "]";
}
}
```

**NamedHQLTest.java:-**

```
package com.nt.test;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class NamedHQLTest {

    public static void main(String[] args) {
        Session ses=null;
        Query query=null;
        List<EmpDetails> list=null;
        EmpDetails details=null;
        Transaction tx=null;
        int result=0;
        // get Session
        ses=HibernateUtil.getSession();

        /* //get Access to Named HQL Query
        query=ses.getNamedQuery("GET_EMPS_By_DOMAIN");
        //set values to query params
        query.setString("domain","%hotmail.com");
```

```

//execute Query
list=query.list();
//process the Result
for(EmpDetails details:list){
    System.out.println(details);
}*/



/* //get Access to Named HQL Query
query=ses.getNamedQuery("GET_EMPS_BY_RANGE");
//set values to query params
query.setInteger("min",100);
query.setInteger("max",200);
//execute Query
list=query.list();
//process the Result
for(EmpDetails details:list){
    System.out.println(details);
}
*/
//load object
query= ses.createQuery("from EmpDetails where no=:id");
query.setInteger("id",103);
details=(EmpDetails) query.list().get(0);
//get Access to Named HQL Query
query=ses.getNamedQuery("UPDATE_EMP_LASTNAME");

query.setString("newMail",details.getName()+"@gmail.com");
query.setInteger("id",103);
try{
    tx=ses.beginTransaction();
    result=query.executeUpdate();
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}
System.out.println("no.of records that are
effected:"+result);
//close objects
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

**HibernateUtil.java:-**

```

package com.nt.utility;

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

```

```

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new
ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        System.out.println("Session obj hashCode:::"+ses.hashCode());
        return ses;
    }//getSession()

    public static void closeSession(){
        Session ses=null;

        ses=threadLocal.get();
        if(ses!=null){
            ses.close();
            threadLocal.remove();
        }
    }

    public static void closeSessionFactory(){
        factory.close();
    }
}//class

```

## Annotation driven Named HQL Queries

Instead of placing HQL query in DAO class they will be taken named HQL queries either in mapping file using <query> tag or in hibernate domain class using mapping annotations

Mapping annotation for named HQL queries are:

- @NamedQueries → To group multiple named HQL queries
- @NamedQuery → To place each named HQL Query

- Up to java7 we can not repeat Annotation at any level of java code from java8 annotation can be repeated but in the designing of the annotation @Repeatable should be there at the @NamedQuery annotation is not repeatable annotation so we can not repeat that annotation at any level to overcome this problem we need to use @NamedQueries annotation is given where multiple annotations can be placed in the value param whose type is array[].

### HBProj21(Anno-Named HQL )

```

    ▲ src
      ▲ com.nt.cfgs
        x hibernate.cfg.xml
      ▲ com.nt.domain
        D Student.java
      ▲ com.nt.test
        D ClientApp.java
      ▲ com.nt.utility
        D HibernateUtil.java
  
```

#### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property
      name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
      <mapping class="com.nt.domain.Student"/>
  </session-factory>
</hibernate-configuration>
  
```

#### Student.java

```

package com.nt.domain;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
  
```

```
import javax.persistence.Transient;

@Entity
@Table(name="student_Tab")
@NamedQueries(value={@NamedQuery(name="GET_ALL_STUDENTS",query="from Student"),
                     @NamedQuery(name="UPDATE_ADDRS_BY_NO",query="update Student
set address=:newAddrs where sno=:no")
}
)
public class Student {

private int sno;

private String sname;

private String address;
//write setters and getters
@Id
@Column(name="stno")
public int getSno() {
return sno;
}

public void setSno(int sno) {
this.sno = sno;
}
@Column(name="stname")
public String getName() {
return sname;
}
public void setName(String sname) {
this.sname = sname;
}
@Column(name="stadd")
/*@Transient*/
public String getAddress() {
return address;
}
public void setAddress(String address) {
this.address = address;
}
@Override
public String toString() {
return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";
}
}
```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```
package com.nt.test;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class ClientApp {
    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();
        //Access Named Query1 and execute it
        Query query1=ses.getNamedQuery("GET_ALL_STUDENTS");
        List<Student> list=query1.list();
        for(Student st:list){
            System.out.println(st);
        }
        //Access Named Query2 and execute it
        Query query2=ses.getNamedQuery("UPDATE_ADDRS_BY_NO");
        query2.setString("newAddrs", "new Delhi");
        query2.setInteger("no",1004);
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            int result=query2.executeUpdate();
            tx.commit();
            System.out.println("No.of records that are effected"+result);
        } //try
        catch(Exception e){
            tx.rollback();
        }
        //close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

## Native SQL

If certain operations are not possible with HQL and we want to use DATABASE software specific feature in hibernate application then we need to go for **NATIVE SQL**.

### The following operations are not possible with HQL:

- Insert record with direct values
- DDL operations
- Calling pl/sql procedure and functions
- Some underlying software specific features.

### Native SQL queries

- These are underlying DATABASE software specific SQL queries
- These queries are DATABASE software dependent Sql queries, so this persistence logic is DATABASE software dependent
- These queries will be written by using DATABASE table name and column names.
- Supports both named (<:name>) and positional parameters(?) .
- Mainly given to call **PL/SQL procedures and functions**.
- **SQLQuery** object represents each native Sql query. Sql query object means it is the object of java class that implements **org.hibernate.SQLQuery(I)** and this interface extends from **org.hibernate.Query(I)**.
- Gives better performance compared to HQL becoz the queries will go and execute in database softwares with out any transulation.
- Use native SQL for any operation that is not possible with HQL, QBC.

### There are two types Native SQL select queries:

#### **1) Entity Queries:**

Selects all column values from Database table and records will be mapped to domain class objects.

Eg:- select \* from student;

#### **2) Scalar Queries**

Select specific multiple column values or aggregate values.

Eg:- select sno,sname from student;

Select count(\*) from student;

- Supports DDL operations and Single record insertion with direct values
- Use list() method to execute native Sql select query and executeUpdate() method to execute native Sql non select query
- Does not support iterate()method i.e. there is no lazy loading

**Example1:**

```
//prepare Native SQL Query
SQLQuery query=ses.createSQLQuery("select * from Employee");
// execute the SQL Query
List <Object[]> list=query.list();
//process the result
for(Object[] row:list){
    for(Object val:row){
        System.out.print(val+"   ");
    }
    System.out.println();
}
}//for
```

->Since domain class is not specified anywhere in code the generated list collection maintains object class object array.

**Example2:**

```
//prepare Native SQL Entity Query with named params (Mapping Domain class)
SQLQuery query=
ses.createSQLQuery("select * from Employee where email like :domain");
//set param value
query.setString("domain","%gmail.com");
// map Domain /Entity class
query.addEntity(EmpDetails.class);
// execute the SQL Query
List <EmpDetails> list=query.list();
//process the result
for(EmpDetails ed:list){
    System.out.println(ed);
}
```

->to get Native SQL select query result in the form of list collection having domain class object we need to map domainclass with SQL Query object by using addEntity() as shown below.

**Example3:**

```
//prepare Native SQL scalar query that selects multiple specific col values
SQLQuery query=
ses.createSQLQuery("select EID,EMAIL from Employee where lastname=:lname");
//set param value
query.setString("lname","bachan");
//map scalar query comps with Hibernate data types
query.addScalar("EID",StandardBasicTypes.INTEGER);
query.addScalar("EMAIL",StandardBasicTypes.STRING);
//execute the Query
List<Object[]> list=query.list();
//process the result
for(Object[] row:list){
    for(Object val:row){
        System.out.print(val+" "+val.getClass());
    }
}
System.out.println();
}//for
```

->Mapping columns with hibernate datatypes due to this the element values in our object class object[] will be in our control.

**Example4:**

```
//prepare scalar query having aggregate function
SQLQuery query=
ses.createSQLQuery("select count(*) from Employee");
query.addScalar("count(*)", StandardBasicTypes.INTEGER);

List <Integer> list=query.list();
int cnt=list.get(0);
System.out.println("Records count"+cnt);
```

->while working with native SQL select queries it is recommended to map hibernate domain class with SQL Query object to get list collection with domain class objects.Similary it is recommended to map specific column names,aggregate function names with hibernate data types to get results as per our choice data type values.

**Note:-**

While working with Native SQL queries we can not call iterate method on SQL query object becoz, given SQL query goes to database software with out any conversion so lazy loading can not be done by fetching records by record.

Eg:- SQLQuery query = ses.createSQLQuery("select \* from Employee");

Iterator it = query.iterate();

->HQL insert query for direct values insertion is not possible, but we can do the same work by using native SQL insert query.

->All Native SQL non-select queries but be executed as transaction statements.

->While working with HQL,NativeSQL queries the named parameters will be converted to positional parameters in the generated SQL query that goes to database software.

**Example5:**

```
//prepare Native SQL non-select Query
SQLQuery query=
ses.createSQLQuery("insert into Employee values(:val1,:val2,:val3,:val4)");
//set param values
query.setInteger("val1", 1111);
query.setString("val2","rajesh");
query.setString("val3","rao");
query.setString("val4","rao@jani.com");
//execute the SQL Query
Transaction tx=null;
try{
tx=ses.beginTransaction();
int count=query.executeUpdate();
tx.commit();
if(count==1)
System.out.println("Record inserted");
else
System.out.println("Record not inserted");
} //try
catch(Exception e){
tx.rollback();
}
}
```

**Named Native SQL Queries:**

In order to make native Sql query executing in 1 of or more Session object of 1 or more client applications or DAO classes we need to use **named native Sql queries**.For this place Named Native SQL Query in mapping file having logical name using **<sql-query>** tag (after **</class>** tag).

**Note:**

We use **<Query>** tag to place named SQL query and **<SQL-Query>** to place named native SQL queries.

**Example1:**

- a) Placed NamedSQL query in mapping file

```
<sql-query name="GET_EMP_DETAILS_BY_RANAGE" >
<return class="com.nt.domain.EmpDetails"/>
<![CDATA[
select * from Employee where EID>=:min and EID<=:max
]]
</sql-query>
```

- b) Access named native sql query and execute it in DAO or Client App.

```
Query query=ses.getNamedQuery("GET_EMP_DETAILS_BY_RANAGE");
query.setInteger("min",100);
query.setInteger("max",200);
List<EmpDetails> list=query.list();
```

**Note:**

Even though we place named parameters in native sql query ,the framework converts them into positional parameters internally.

**Note:**

We use <return>tag to map entity query records with domain class objects, it is alternate to add entity () method, we can <return-scalar>tags to map scalar query column results,aggregate results with hibernate datatypes.

**Example2:**

- a) Placed NamedSQL query in mapping file

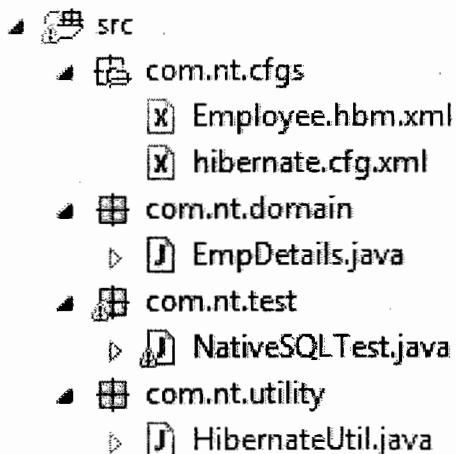
```
<sql-query name="GET_EMP_DETAILS_BY_MAIL">
<return-scalar column="firstname" type="string"/>
<return-scalar column="lastname" type="string"/>
    select firstname, lastname from Employee where email like :domain
</sql-query>
```

- b) Access named native sql query and execute it.

```
Query query=ses.getNamedQuery("GET_EMP_DETAILS_BY_MAIL");
query.setString("domain","%gmail.com");
List<Object[]> list=query.list();
```

**Note:**

- In one mapping file we can place multiple named native Sql queries and named HQL queries
- The results of non-select native Sql queries need not be mapped with hibernate data types.

 HBProj22(NativeSQL & Named Native SQL)
**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
<session-factory>
    <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">scott</property>
```

```

<property name="connection.password">tiger</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

```

<!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.domain.EmpDetails" table="Employee">
<id name="no" column="EID" type="int" /><!-- Singular Id field -->
<property name="fname" column="FIRSTNAME" length="20" not-null="true"/>
<property name="lname" column="LASTNAME" length="20"/>
<property name="mail" column="EMAIL" type="string" length="20" unique="true"/>
</class>
<sql-query name="GET_EMP_DETAILS_BY_RANAGE" >
<return class="com.nt.domain.EmpDetails"/>
<![CDATA[
    select * from Employee where EID>=:min and EID<=:max
  ]]>
</sql-query>

<sql-query name="GET_EMP_DETAILS_BY_MAIL" >
<return-scalar column="firstname" type="string"/>
<return-scalar column="lastname" type="string"/>
    select firstname, lastname from Employee where email like :domain
</sql-query>

</hibernate-mapping>

```

### EmpDetails.java

```

package com.nt.domain;

public class EmpDetails {
//domain class properties
private int no;
private String fname;
private String lname;
private String mail;

public EmpDetails(){
System.out.println("EmpDetails:0-param constructor");
}

```

```
public int getNo() {  
  
    return no;  
}  
public void setNo(int no) {  
  
    this.no = no;  
}  
public String getFname() {  
  
    return fname;  
}  
public void setFname(String fname) {  
  
    this.fname = fname;  
}  
public String getLname() {  
  
    return lname;  
}  
public void setLname(String lname) {  
  
    this.lname = lname;  
}  
public String getMail() {  
  
    return mail;  
}  
public void setMail(String mail) {  
  
    this.mail = mail;  
}  
  
@Override  
public String toString() {  
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

**HibernateUtil.java**

Same as previous

**NativeSQLTest.java**

```
package com.nt.test;  
  
import java.util.List;
```

```
import org.hibernate.Query;
import org.hibernate.Session;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class NativeSQLTest {
    public static void main(String[] args) {
        //Get Session
        Session ses=HibernateUtil.getSession();
        /* //prepare Native SQL Query
        SQLQuery query=ses.createSQLQuery("select * from Employee");
        // execute the SQL Query
        List <Object[]> list=query.list();
        //process the result
        for(Object[] row:list){
            for(Object val:row){
                System.out.print(val+"   ");
            }
            System.out.println();
        }
        */
        /* //prepare Native SQL Entity Query with named params (Mapping Domain class)
        SQLQuery query=
        ses.createSQLQuery("select * from Employee where email like :domain");
        //set param value
        query.setString("domain","%gmail.com");
        // map Domain /Entity class
        query.addEntity(EmpDetails.class);
        // execute the SQL Query
        List <EmpDetails> list=query.list();
        //process the result
        for(EmpDetails ed:list){
            System.out.println(ed);
        }
        */

        /*//prepare Native SQL scalar query that selects multiple specific col values
        SQLQuery query=
        ses.createSQLQuery("select EID,EMAIL from Employee where lastname=:lname");
        //set param value
        query.setString("lname","bachan");
        //map scalar query comps with Hibernate data types
        query.addScalar("EID",StandardBasicTypes.INTEGER);
        query.addScalar("EMAIL",StandardBasicTypes.STRING);
        //execute the Query
        List<Object[]> list=query.list();
        //process the result
        for(Object[] row:list){
            for(Object val:row){
```

```
System.out.print(val+" "+val.getClass());
}//for
System.out.println();
}//for
*/
/* //prepare scalar query having aggregate function
SQLQuery query=
ses.createSQLQuery("select count(*) from Employee");
query.addScalar("count(*)", StandardBasicTypes.INTEGER);

List <Integer> list=query.list();
int cnt=list.get(0);
System.out.println("Records count"+cnt);
*/
/* //prepare Native SQL non-select Query
SQLQuery query=
ses.createSQLQuery("insert into Employee values(:val1,:val2,:val3,:val4)");
//set param values
query.setInteger("val1", 1111);
query.setString("val2", "rajesh");
query.setString("val3", "rao");
query.setString("val4", "rao@jani.com");
//execute the SQL Query
Transaction tx=null;
try{
tx=ses.beginTransaction();
int count=query.executeUpdate();
tx.commit();
if(count==1)
System.out.println("Record inserted");
else
System.out.println("Record not inserted");
} //try
catch(Exception e){
tx.rollback();
}
*/
/* // Access Named Native SQL Query and execute it
Query query=ses.getNamedQuery("GET_EMP_DETAILS_BY_RANAGE");
query.setInteger("min",100);
query.setInteger("max",200);
List<EmpDetails> list=query.list();
for(EmpDetails ed:list){
System.out.println(ed);
}
*/
/* Query query=ses.getNamedQuery("GET_EMP_DETAILS_BY_MAIL");
query.setString("domain", "%gmail.com");
```

```
List<Object[]> list=query.list();
for(Object row[]:list){
    for(Object val:row){
        System.out.print(val+" ");
    }
    System.out.println();
}*/



//close Session
HibernateUtil.closeSession();
    HibernateUtil.closeSessionFactory();
}//main
}//class
```

### Annotation driven Named Native SQL Queries:

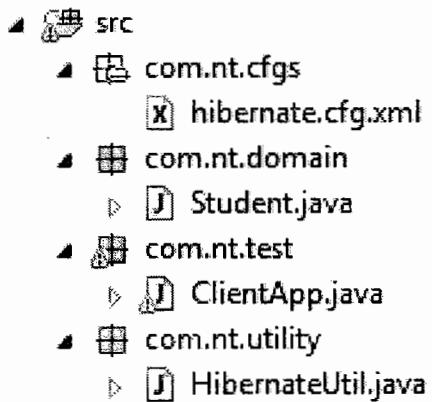
Use the following JPA annotations to place Named Native SQL Queries on the top of domain class.

- **@NamedNativeQueries:** To hold multiple named native queries
- **@NamedNativeQuery:** To place each named native SQL query

->In Hibernate 4.x we can not execute Named Native SQL scalar select Queries and non select queries using Annotations. But in hibernate 5.x we can make any kind of Named Native SQL Queries executing through annotations.

->Since Query interface is super interface of SQL query interface the **session.getNamedQuery ()** method returned object (query object) can represent both named HQL query, Named Native SQL query.

 HBProj23(Anno-Named Native SQL )



**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property
            name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

        <mapping class="com.nt.domain.Student"/>
    </session-factory>
</hibernate-configuration>
```

**Student.java**

```
package com.nt.domain;

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

@Entity
@Table(name="student_Tab")
@NamedNativeQueries(value={@NamedNativeQuery(name="GET_ALL_STUDENTS",
        query="select * from student_tab",
        resultClass=Student.class),
        @NamedNativeQuery(name="DELETE_STUDENTS_BY_ADDRS",
        query="delete from student_Tab where stadd=:addrs"
    )
})
public class Student {

    private int sno;

    private String sname;

    private String address;
```

```
//write setters and getters
@Id
@Column(name="stno")
public int getSno() {
    return sno;
}

public void setSno(int sno) {
    this.sno = sno;
}

@Column(name="stname")
public String getName() {
    return sname;
}

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

@Column(name="stadd")
/*@Transient*/
public String getAddress() {
    return address;
}

public void setAddress(String address) {
    this.address = address;
}

@Override
public String toString() {
    return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";
}

}
```

#### HibernateUtil.java

Same as previous

#### ClientApp.java

```
package com.nt.test;
import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;
```

```

public class ClientApp {
    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();

        // get Access to Named Native SQL Query1 and execute it.
        Query query=ses.getNamedQuery("GET_ALL_STUDENTS");
        List <Student> list=query.list();
        for(Student st:list){
            System.out.println(st);
        }

        // get Access to Named Native SQL Query2 and execute it.
        Query query1=ses.getNamedQuery("DELETE_STUDENTS_BY_ADDRS");
        query1.setString("addrs","hyd2");
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            int result=query1.executeUpdate();
            tx.commit();
            System.out.println("Deleted records count"+result);
        }/try
        catch(Exception e){
            tx.rollback();
        }
        //close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}

```

### Calling PL/SQL procedures and functions from hibernate App:

- In order to make persistence logic as reusable logic in multiple applications of a module or multiple modules of project we need to work with PL/SQL stored procedures or functions which will reside and execute in DATABASE software by centralizing persistence logic.  
eg:Authentication logic is required in multiple modules of project, so place in DATABASE software as PL/SQL procedure or function and use it for multiple times.
- **Cursor** is special in-memory variable of Oracle PL/SQL programming of oracle having the capacity of holding bunch of records. It is like ResultSet object of JDBC programming.
- From hibernate3 , we can call PL/SQL procedure or function of DATABASE software, for that we need to follow set of rules while creating those PL/SQL procedures and functions, these rules will change based on the DATABASE software we use, **the commons rules are:**

- a) Use SQL92 syntax to call PL/SQL procedures or functions:  
`{call<procedure name>(<params>)}  
{?=call<function name>(<params>)}`
- b) Don't apply pagination on results of PL/SQL procedure or functions (We can not call set FirstResult (-), set MaxResult (-) methods).

### Oracle specific rules are

- For a procedure the first parameter must be an **out parameter** of type cursor. Cursor returns set of records by taking the parameter type as **sys\_refcursor**.
- For a function it must return ResultSet by taking cursor as the return type i.e. function return type must be **sys\_refcursor**.

#### *Create a PL/SQL function in oracle*

```
create or replace function GET_EMP_DETAILS(stno in number , endno in number) return
sys_refcursor as
details sys_refcursor
begin
open detail for
select * from employee where eid<=stno and eid>= endno
return details;
end;
```

#### *Use case:*

In university project multiple departments will be developed as multiple modules instead of writing attendances calculation logic in every module it is recommended to write only for one time in database software as PL/SQL procedure (or) function and use it in multiple modules.

#### HBProj24(Calling PI-sql function (NativeSQL))

```
src
└── com.nt.cfgs
    ├── Employee.hbm.xml
    └── hibernate.cfg.xml
└── com.nt.domain
    └── EmpDetails.java
└── com.nt.test
    └── FunctionTest.java
└── com.nt.utility
    └── HibernateUtil.java
```

#### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
```

```

<session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

**Employee.hbm.xml**

```

<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID" type="int" /><!-- Singular Id field -->
        <property name="fname" column="FIRSTNAME" length="20" not-null="true"/>
        <property name="lname" column="LASTNAME" length="20"/>
        <property name="mail" column="EMAIL" type="string" length="20" unique="true"/>
    </class>
    <sql-query name="FX_TEST" callable="true">
        <return class="com.nt.domain.EmpDetails"/>
        { ?= call GET_EMP_DETAILS(?,?) }
    </sql-query>
</hibernate-mapping>

```

**EmpDetails.java**

```

package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails(){}
    System.out.println("EmpDetails:0-param constructor");
}

public int getNo() {

    return no;
}

```

```
public void setNo(int no) {  
  
    this.no = no;  
}  
public String getFname() {  
  
    return fname;  
}  
public void setFname(String fname) {  
  
    this.fname = fname;  
}  
public String getLname() {  
  
    return lname;  
}  
public void setLname(String lname) {  
  
    this.lname = lname;  
}  
public String getMail() {  
  
    return mail;  
}  
public void setMail(String mail) {  
  
    this.mail = mail;  
}  
  
@Override  
public String toString() {  
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

**HibernateUtil.java**

Same as previous

**FunctionTest.java**

```
package com.nt.test;  
import java.util.List;  
  
import org.hibernate.Query;  
import org.hibernate.Session;  
  
import com.nt.domain.EmpDetails;  
import com.nt.utility.HibernateUtil;
```

```

public class FunctionTest {
    public static void main(String[] args) {
        //Get Session
        Session ses=HibernateUtil.getSession();
        // get Access to Named Native SQL Query and execute it to call
        //pl/sql function
        Query query=ses.getNamedQuery("FX_TEST");
        query.setInteger(0,100);
        query.setInteger(1,200);
        List <EmpDetails> list=query.list();
        for(EmpDetails ed: list){
            System.out.println(ed);
        }

        //close Session
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class

```

**Example2:****PL/SQL procedure in Oracle:**

```

Create or replace procedure
GET_EMP_DETAILS_FOR_DOMAIN(details out sys_refcursor, domain in varchar2)
as begin
open detail for
select firstname, lastname from employee where email like domain;
end;

```

**HBProj25(Calling PL-sql procedure (NativeSQL))**

- ▶ **src**
  - ▶ **com.nt.cfgs**
    - Employee.hbm.xml
    - hibernate.cfg.xml
  - ▶ **com.nt.domain**
    - ▶ EmpDetails.java
  - ▶ **com.nt.test**
    - ▶ ProcedureTest.java
  - ▶ **com.nt.utility**
    - ▶ HibernateUtil.java

**hibernate.cfg.xml**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

```

```
<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
    <id name="no" column="EID" type="int" /> <!-- Singular Id field -->
    <property name="fname" column="FIRSTNAME" length="20" not-null="true"/>
    <property name="lname" column="LASTNAME" length="20"/>
    <property name="mail" column="EMAIL" type="string" length="20" unique="true"/>
  </class>
  <sql-query name="PROCEDURE_TEST" callable="true">
    <return-scalar column="firstname" type="string"/>
    <return-scalar column="lastname" type="string"/>
    { call GET_EMP_DETAILS_FOR_DOMAIN(?, :dmn) }
  </sql-query>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private int no;
  private String fname;
  private String lname;
  private String mail;

  public EmpDetails(){}
  System.out.println("EmpDetails:0-param constructor");
}
```

```
public int getNo() {  
  
    return no;  
}  
public void setNo(int no) {  
  
    this.no = no;  
}  
public String getFname() {  
  
    return fname;  
}  
public void setFname(String fname) {  
  
    this.fname = fname;  
}  
public String getLname() {  
  
    return lname;  
}  
public void setLname(String lname) {  
  
    this.lname = lname;  
}  
public String getMail() {  
  
    return mail;  
}  
public void setMail(String mail) {  
  
    this.mail = mail;  
}  
  
@Override  
public String toString() {  
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";  
}  
}
```

**HibernateUtil.java**

Same as previous

**ProcedureTest.java**

```
package com.nt.test;  
  
import java.util.List;
```

```

import org.hibernate.Query;
import org.hibernate.Session;

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class ProcedureTest {
    public static void main(String[] args) {
        //Get Session
        Session ses=HibernateUtil.getSession();
        // get Access to Named Native SQL Query and execute it to call
        //pl /sql function
        Query query=ses.getNamedQuery("PROCEDURE_TEST");
        query.setString("dmn", "%gmail.com");
        List <Object[]> list=query.list();
        for(Object row[]: list){
            for(Object val: row){
                System.out.print(val+" ");
            }
            System.out.println();
        }

        //close Session
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class

```

### **We can call procedure that does not follow hibernate rules:**

- The method that will be called automatically by underlying container or framework is called callback method.
- The interface that contains such callback method is called callback interface, the underlying framework or container automatically calls the implementation class method of callback interface.
- Generally framework supplied callback interface with callback method exposes the object of underlying technology to write that technology specific code in the framework application
- If PL/SQL procedure or function is not designed as per hibernate rules then we need to take two callback interface given in hibernate API.
  1. **Work(I) : Does not allow to return the value**
  2. **ReturningWork(I) : Allows to return the value**

→ This callback interface provides callback methods execute (-) exposing JDBC Connection object as the parameter and we can use this JDBC Connection object to create other JDBC objects and to develop persistence logic like creating CallableStatement object to call PL/SQL procedures or functions

Create Database table in oracle

<u>userlist</u>	
<u>uname(varchar2)pk</u>	<u>pwd(vc2)</u>
raja	rani
king	kingdom

Create pl/sql procedure in oracle for authentication (no hibernate rules are following here)

```
create or replace procedure
AUTHENTICATION_PRO(user in varchar2, pass in varchar2, result out varchar2)
as
cnt number;
begin
select count(*) into cnt from userlist where uname=user and pwd=pass;
if(cnt<>0)then
result:='valid credential';
else
result:='invalid credential';
end if;
end;
/
```

->In any framework Callback interfaces and callback methods are given to access underlying technology object and to create customized logic having capability to generate customized results.

#### Note:

If you want to use callback interface implementing class logic in multiple DAO class then go for separate outerclass, if same logic is required in multiple methods of DAO class then go for separate innerclass inside the DAO class, if same logic is required only in one method of DAO class then go for anonymous inner class (or) method level innerclass.

#### Anonymous innerclass through Lamida expression:-

```
String result = ses.doReturningWork (con -> {
```

```
.....
```

```
.....
```

```
.....
```

```
return result;
```

```
}
```

#### Note:-

We can go for Lamida Expression based Anonymous innerclass only when callback interface contains one method with one param otherwise not possible.

 HBProj26(Callback interfaces based pl-sql procedure call)

- ▶  src
  - ▶  com.nt.cfgs
    - ☒ hibernate.cfg.xml
  - ▶  com.nt.test
    - ▷ J ProcedureTest.java
    - ▷ J ProcedureTest1.java
  - ▶  com.nt.utility
    - ▷ J HibernateUtil.java

[hibernate.cfg.xml](#)

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    </session-factory>
</hibernate-configuration>
```

[HibernateUtil.java](#)

Same as previous

[ProcedureTest.java](#)

```
package com.nt.test;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;

import org.hibernate.Session;
import org.hibernate.jdbc.ReturningWork;

import com.nt.utility.HibernateUtil;

public class ProcedureTest {
    public static void main(String[] args) {
        // Get Session
```

```

Session ses=HibernateUtil.getSession();
// Use callback interface to call pl/sql procedure
String result=ses.doReturningWork(new InvokeProcedure());
System.out.println("Result :::"+result);
//close HB objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main(-)
private static class InvokeProcedure implements ReturningWork<String>{
@Override
public String execute(Connection con) throws SQLException {
CallableStatement cs=null;
String result=null;
//create Callable Statement obj
cs=con.prepareCall("{call Authetication_pro(?, ?, ?)}");
//register OUT params with JDBC types
cs.registerOutParameter(3,Types.VARCHAR);
//set Values to IN Params
cs.setString(1,"king");
cs.setString(2,"kingdom");
//execute pl/sql procedure
cs.execute();
//Gather results from OUT param
result=cs.getString(3);
return result;
}//execute(-)
}//inner class
}//outer class

```

**ProcedureTest1.java**

```

package com.nt.test;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;
import org.hibernate.Session;
import org.hibernate.jdbc.ReturningWork;
import com.nt.utility.HibernateUtil;

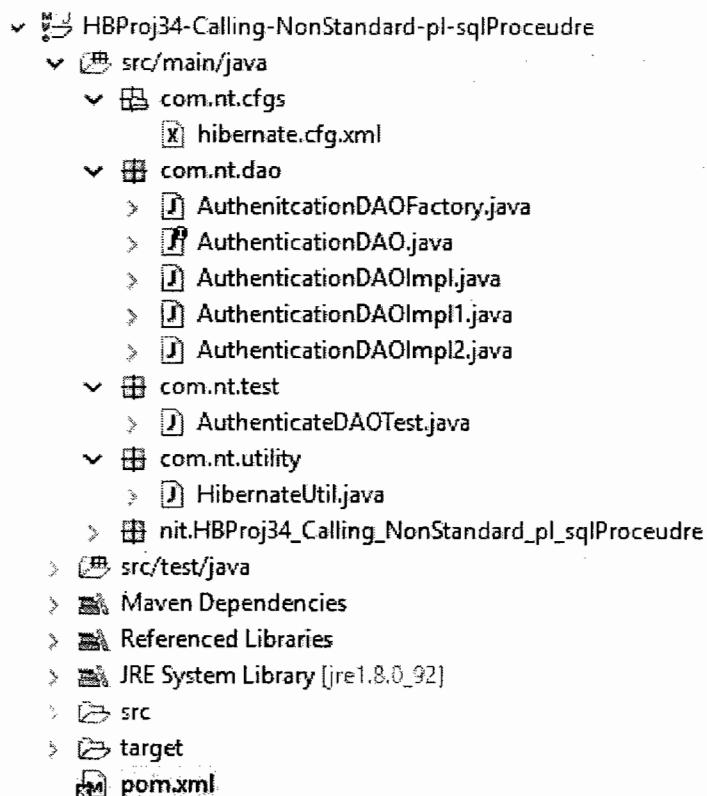
public class ProcedureTest1 {
public static void main(String[] args) {
// Get Session
Session ses=HibernateUtil.getSession();
// Use callback interface to call pl/sql procedure
String result=ses.doReturningWork(new ReturningWork<String>(){
@Override

```

```

public String execute(Connection con) throws SQLException {
    CallableStatement cs=null;
    String result=null;
    //create Callable Statement obj
    cs=con.prepareCall("{call Authetincation_pro(?, ?, ?)}");
    //register OUT params with JDBC types
    cs.registerOutParameter(3,Types.VARCHAR);
    //set Values to IN Params
    cs.setString(1,"king");
    cs.setString(2,"kingdom1");
    //execute pl/sql procedure
    cs.execute();
    //Gather results from OUT param
    result=cs.getString(3);
    return result;
} //execute(-)
} //inner class
);
System.out.println("Result :::"+result);
//close HB objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
} //main(-)
} //inner class
} //outer class

```



hibernate.cfg.xml:-

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
            name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
    </session-factory>
</hibernate-configuration>
```

AuthenitcationDAOFactory.java:-

```
package com.nt.dao;

public class AuthenitcationDAOFactory {
    public static AuthenticationDAO getInstance(){
        /*return new AuthenticationDAOImpl();*/
        return new AuthenticationDAOImpl1();
    }
}
```

AuthenticationDAO.java:-

```
package com.nt.dao;

public interface AuthenticationDAO {
    public String authenticate(String user, String pwd);
}
```

AuthenticationDAOImpl.java:-

```
package com.nt.dao;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;
import org.hibernate.Session;
import org.hibernate.jdbc.ReturningWork;
import com.nt.utility.HibernateUtil;

public class AuthenticationDAOImpl implements AuthenticationDAO {
    private static final String CALL_PRO_AUTHENTICATION = "{ call
        PRO_AUTHENTICATION(?, ?, ?)}";
```

```

@Override
public String authenticate( String user, String pwd) {
    String result=null;
    // get Session
    Session ses=HibernateUtil.getSession();
    //get result
    result=ses.doReturningWork(new AuthenticateWork(user,pwd));
    return result;
}//method

private class AuthenticateWork implements ReturningWork<String>{
    private String user,pwd;
    public AuthenticateWork(String user, String pwd) {
        this.user=user;
        this.pwd=pwd;
    }
    @Override
    public String execute(Connection con) throws SQLException {
        CallableStatement cs=null;
        String result=null;
        //create CallableStatement object
        cs=con.prepareCall(CALL_PRO_AUTHENTICATION);
        //register OUT parms with JDBC types
        cs.registerOutParameter(3,Types.VARCHAR);
        //set values to IN Params
        cs.setString(1,user);
        cs.setString(2,pwd);
        //execute pl/sql procedure
        cs.execute();
        //gather results from OUT param
        result=cs.getString(3);
        return result;
    }
}//inner class
}//outer class

```

AuthenticationDAOImpl1.java:-

```

package com.nt.dao;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;
import org.hibernate.Session;
import org.hibernate.jdbc.ReturningWork;
import com.nt.utility.HibernateUtil;

public class AuthenticationDAOImpl1 implements AuthenticationDAO {
    private static final String CALL_PRO_AUTHENTICATION="{ call

```

```

PRO_AUTHENTICATION(?, ?, ?)}";
@Override
public String authenticate( String user, String pwd) {
    String result=null;
    // get Session
    Session ses=HibernateUtil.getSession();
    //get result
    result=ses.doReturningWork(new ReturningWork<String>(){
        @Override
        public String execute(Connection con) throws
SQLException {
            CallableStatement cs=null;
            String result=null;
            //create CallableStatement object
            cs=con.prepareCall(CALL_PRO_AUTHENTICATION);
            //register OUT parms with JDBC types
            cs.registerOutParameter(3,Types.VARCHAR);
            //set values to IN Params
            cs.setString(1,user);
            cs.setString(2,pwd);
            //execute pl/sql procedure
            cs.execute();
            //gather results from OUT param
            result=cs.getString(3);
            return result;
        } //execute(-)
    });
    return result;
} //method

}// class

```

**AuthenticationDAOImpl2.java:-**

```

package com.nt.dao;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;
import org.hibernate.Session;
import org.hibernate.jdbc.ReturningWork;
import com.nt.utility.HibernateUtil;

public class AuthenticationDAOImpl2 implements AuthenticationDAO {
    private static final String CALL_PRO_AUTHENTICATION="{ call
PRO_AUTHENTICATION(?, ?, ?)}";
    @Override
    public String authenticate( String user, String pwd) {
        String result=null;
        // get Session

```

```
Session ses=HibernateUtil.getSession();
//get result
result=ses.doReturningWork(con->{
    CallableStatement cs=null;
    String resultMsg=null;
    //create CallableStatement object
    cs=con.prepareCall(CALL_PRO_AUTHENTICATION);
    //register OUT parms with JDBC types
    cs.registerOutParameter(3,Types.VARCHAR);
    //set values to IN Params
    cs.setString(1,user);
    cs.setString(2,pwd);
    //execute pl/sql procedure
    cs.execute();
    //gather results from OUT param
    resultMsg=cs.getString(3);
    return resultMsg;
});//execute(-)
);
return result;
};//method

}// class
```

**AuthenticateDAOTest.java:-**

```
package com.nt.test;

import com.nt.dao.AuthenitcationDAOFactory;
import com.nt.dao.AuthenticationDAO;
import com.nt.utility.HibernateUtil;

public class AuthenticateDAOTest {

    public static void main(String[] args) {
        AuthenticationDAO dao=null;
        String result=null;
        //get DAO
        dao=AuthenitcationDAOFactory.getInstance();
        //use dAO
        result=dao.authenticate("raja","rani");
        System.out.println("Result=====>" +result);
        //close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    };//method
};//class
```

HibernateUtil.java:-

```

package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new
ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml")
                .buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        System.out.println("Session obj hashCode:::"+ses.hashCode());
        return ses;
    }//getSession()

    public static void closeSession(){
        Session ses=null;

        ses=threadLocal.get();
        if(ses!=null){
            ses.close();
            threadLocal.remove();
        }
    }

    public static void closeSessionFactory(){
        factory.close();
    }
}//class

```

pom.xml:-

```

<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

```

```
<groupId>nit</groupId>
<artifactId>HBProj17-CompositeID</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>HBProj17-CompositeID</name>
<url>http://maven.apache.org</url>

<properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>

    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.1.0.Final</version>
    </dependency>

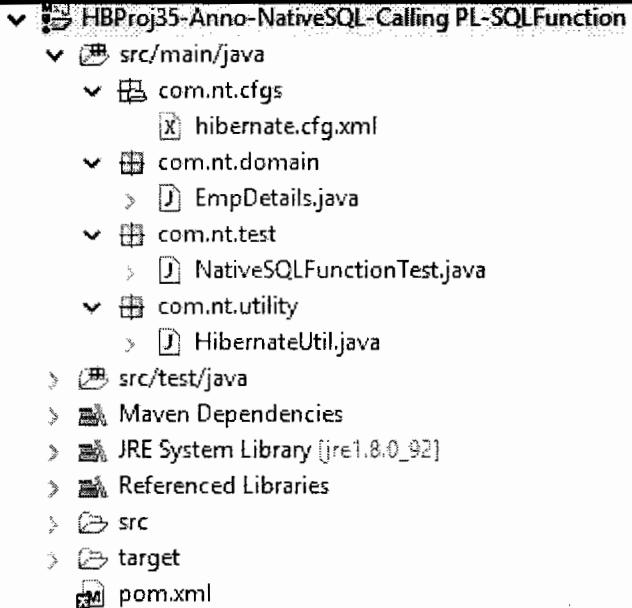
    <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc6</artifactId>
        <version>11.2.0</version>
    </dependency>
</dependencies>
</project>
```

**Important Note:-**

→ The JPA supplied named native query are not suitable to call PL/SQL procedure or function because there is no callable="true" column param in those annotations. So use hibernate supplied annotations for the same which callable param.

-> @NamedNativeQuery (JPA) is not suitable for calling PL/SQL procedure or function becoz callable = "true" becoz callable param is not there.

@NamedNativeQuery (Hibernate) is suitable for calling PL/SQL procedure or function becoz callable="true" is there.

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.EmpDetails"/>
    </session-factory>

</hibernate-configuration>
```

**EmpDetails.java:-**

```
package com.nt.domain;

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

//import org.hibernate.annotations.NamedNativeQuery;
```

```
@Entity
@Table(name="Employee")
/*@NamedNativeQuery(name="CALL_Fx",
query="{ ?=call
GET_EMP_DETAILS_RANGE(:min,:max)}",
callable=true,
resultClass=EmpDetails.class)*/
@NamedNativeQuery(name="CALL_Fx",
query="{ ?=call GET_EMP_DETAILS_RANGE(:min,:max)}",
hints=@QueryHint(name="org.hibernate.callable",value="true"),
resultClass=EmpDetails.class)

public class EmpDetails {
    @Id
    @Column(name="EID")
    private int no;
    @Column(name="FIRSTNAME")
    private String fname;
    @Column(name="LASTNAME")
    private String lname;
    @Column(name="EMAIL")
    private String mail;

    public EmpDetails() {
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
        this.lname = lname;
    }
    public String getMail() {
        return mail;
    }
    public void setMail(String mail) {
```

```
        this.mail = mail;
    }

    @Override
    public String toString() {
        return "EmpDetails [no=" + no + ", fname=" + fname + ",
lname=" + lname + ", mail=" + mail + "]";
    }
}
```

**NativeSQLFunctionTest.java:-**

```
package com.nt.test;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.SQLQuery;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class NativeSQLFunctionTest {

    public static void main(String[] args) {
        Session ses=null;
        Query query=null;
        List<EmpDetails> list=null;
        // get Session
        ses=HibernateUtil.getSession();
        //get Access Named Native SQL query
        query=ses.getNamedQuery("CALL_Fx");
        //set values to Params
        query.setInteger("min",100);
        query.setInteger("max",200);
        //excute Named Native SQL Query/calling PL/SQL Function
        list=query.list();
        //process the Reuslts
        for(EmpDetails ed:list){
            System.out.println(ed);
        }

        //close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();

    }//main
}//class
```

**HibernateUtil.java:-**

```
package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new
ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
                buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        System.out.println("Session obj hashCode:::"+ses.hashCode());
        return ses;
    }//getSession()

    public static void closeSession(){
        Session ses=null;

        ses=threadLocal.get();
        if(ses!=null){
            ses.close();
            threadLocal.remove();
        }
    }

    public static void closeSessionFactory(){
        factory.close();
    }

}//class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
```

```
<modelVersion>4.0.0</modelVersion>

<groupId>nit</groupId>
<artifactId>HBProj17-CompositeID</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>HBProj17-CompositeID</name>
<url>http://maven.apache.org</url>

<properties>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
</properties>

<dependencies>
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
    </dependency>

    <!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-core</artifactId>
        <version>5.1.0.Final</version>
    </dependency>

    <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc6</artifactId>
        <version>11.2.0</version>
    </dependency>
</dependencies>
</project>
```

Simple application to call pl/sql procedure by using annotation driven Hibernate code by following the Hibernate rules.

```
Create or replace procedure GET_STUDENT_BY_ADDRS(details out sys_refcursor, address in
varchar2)as begin
Open details for
  Select sno,sname, sad from student where stadd=adrs;
End;
/
```

- Keep annotation driven example application ready
- Add the following annotation on the Student class(domain class)

```
@Entity
@Table(name="student_Tab")
@NamedNativeQuery(name = "TEST_PROC",
    query = "{ call GET_STUDENTS_BY_ADDRS(?:addr) }",
    hints=@QueryHint(name="org.hibernate.Callable",value="true" ),
    resultClass=Student.class
)
```

Get names native query in DAO class/ClientApp and to execute that query to call pl/sql procedure.

-> Using Hibernate annotation @NamedNativeQuery (name= "TEST\_proc", query= "{? = call GET\_STUDENTS\_BY\_ADDRS (?,: addr) }" callable= true,resultClass = students.class)

### HBProj27(Anno-Calling pl-sql procedure )

```
▶ src
  ▶ com.nt.cfgs
    X hibernate.cfg.xml
  ▶ com.nt.domain
    ▶ J Student.java
  ▶ com.nt.test
    ▶ J ClientApp.java
  ▶ com.nt.utility
    ▶ J HibernateUtil.java
```

#### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
  <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">scott</property>
  <property name="connection.password">tiger</property>
  <property
    name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

  <mapping class="com.nt.domain.Student"/>

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

**Student.java**

```
package com.nt.domain;

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

@Entity
@Table(name="student_Tab")
@NamedNativeQuery(name = "TEST_PROC",
    query = "{ call GET_STUDENTS_BY_ADDRS(?,:addrs) }",
    hints=@QueryHint(name="org.hibernate.callable",value="true" ),
    resultClass=Student.class
)
public class Student {

    private int sno;
    private String sname;
    private String address;
    //write setters and getters
    @Id
    @Column(name="stno")
    public int getSno() {
        return sno;
    }

    public void setSno(int sno) {
        this.sno = sno;
    }

    @Column(name="stname")
    public String getSname() {
        return sname;
    }

    public void setSname(String sname) {
        this.sname = sname;
    }

    @Column(name="stadd")
    /*@Transient*/
    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

```
@Override  
public String toString() {  
    return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";  
}  
  
}
```

### HibernateUtil.java

Same as previous

### ClientApp.java

```
package com.nt.test;  
  
import java.util.List;  
  
import org.hibernate.Query;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import com.nt.domain.Student;  
import com.nt.utility.HibernateUtil;  
  
public class ClientApp {  
    public static void main(String[] args) {  
        // get Session  
        Session ses=HibernateUtil.getSession();  
  
        // get Access to Named Native SQL Query  
        Query query=ses.getNamedQuery("TEST_PROC");  
        query.setString("addrs","vizag");  
        List<Student> list=query.list();  
        for(Student st:list){  
            System.out.println(st);  
        }  
  
        //close objs  
        HibernateUtil.closeSession();  
        HibernateUtil.closeSessionFactory();  
  
    }  
}
```

## Criteria API or (QBC: Query By Criteria)

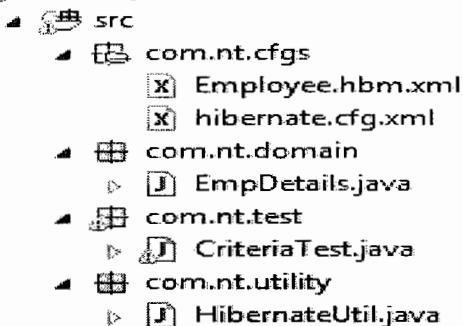
- This persistence logic will be written by using java statements without using queries
- This persistence logic generates optimized Sql query as required for underlying DATABASE software for better performance.
- QBC logic is developed using domain class and its properties so it is DATABASE software independent persistence logic.
- This logic is useful for pagination and report generation, this is 100% java code based persistence logic supporting **both single row and bulk operations**.
- This logic supports only select operations like getting all column values, specific multiple column values , specific single column values.
- QBC supports aggregate operations but does not support non-select operations
- Using QBC we can not perform non-select operations, DDL operations and calling PL/SQL procedures and functions.

### Criteria object:

→Represents whole QBC logic and it is the object of java class that implements **org.hibernate.Criteria (I)**

->To execute QBC logics just list method is available and iterate method is not available this indicates **only eager loading is possible and lazy loading is not possible**.

### HBProj28(Criteria API)



### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
  
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
<id name="no" column="EID"/>
<property name="fname" >
  <column name="FIRSTNAME"/>
</property>
<property name="lname" >
  <column name="LASTNAME"/>
</property>
<property name="mail" >
  <column name="EMAIL"/>
</property>
</class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private int no;
  private String fname;
  private String lname;
  private String mail;

  public EmpDetails(){
    System.out.println("EmpDetails:0-param constructor");
  }

  public int getNo() {
    return no;
  }
  public void setNo(int no) {
    this.no = no;
  }
  public String getFname() {
    return fname;
  }
  public void setFname(String fname) {
```

```
this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}
```

#### HibernateUtil.java

Same as previous

#### CriteriaTest.java

```
package com.nt.test;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.criterion.Criterion;
import org.hibernate.criterion.Order;
import org.hibernate.criterion.Projection;
import org.hibernate.criterion.ProjectionList;
import org.hibernate.criterion.Projections;
import org.hibernate.criterion.Restrictions;

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class CriteriaTest {
public static void main(String[] args) {
//Get Session
```

```
Session ses=HibernateUtil.getSession();
/* // create QBC logic
Criteria criteria=ses.createCriteria(EmpDetails.class);
//execute QBC logic
List <EmpDetails>list=criteria.list();
// DATABASE table ---> ResultSet --> Obs of Domain class--> List collection
//process the Result
for(EmpDetails ed:list)
System.out.println(ed);
*/
/* //create QBC logic with conditions
Criteria criteria=ses.createCriteria(EmpDetails.class);
//prepare condition (Criterion objs)
Criterion cond=Restrictions.between("no",100,400);
//add condition
criteria.add(cond);
//execute QBC
List<EmpDetails> list=criteria.list();
//process the results
for(EmpDetails ed:list){
System.out.println(ed);
}/*
/*// create QBC Logic with conditions
Criteria criteria=ses.createCriteria(EmpDetails.class);
//prepare conditions
Criterion cond1=Restrictions.ge("no",100);
Criterion cond2=Restrictions.le("no",200);
Criterion cond3=Restrictions.like("mail","%gmail.com");
//apply and clause on cond1,cond2
Criterion andCond=Restrictions.and(cond1,cond2);
//apply or clause on cond1+cond2 and cond3
Criterion finalCond=Restrictions.or(andCond,cond3);
//Add criterion objs
criteria.add(finalCond);
//execute QBC logic
List<EmpDetails> list=criteria.list();
//process the result
for(EmpDetails ed:list){
System.out.println(ed);
}/*
/* //Create Criteria obj
Criteria criteria=ses.createCriteria(EmpDetails.class);
//prepare SQL Query based condition
Criterion cond=Restrictions.sqlRestriction(" email like '%gmail.com%'");
//add condition
criteria.add(cond);
//execute QBC
```

```
List <EmpDetails> list=criteria.list();
//process the results
for(EmpDetails ed:list){
    System.out.println(ed);
}*/  
  
/* //QBC logic having Order,Criterion objs
Criteria criteria=ses.createCriteria(EmpDetails.class);
//prepare Criterion objs
Criterion criterion=Restrictions.in("lname",new String[]{"rao","rathod"});
// add Criterion obj
criteria.add(criterion);
//prpeare Order
Order order=Order.asc("fname");
//add ORder
criteria.addOrder(order);
//execute QBC logic
List <EmpDetails> list=criteria.list();
//process the result
for(EmpDetails ed:list){
    System.out.println(ed);
}*/  
/* //QBC logic to get specific multiple col values from DATABASE table
Criteria criteria=ses.createCriteria(EmpDetails.class);
//prepre conditions
Criterion cond=Restrictions.ilike("mail","%GMAIL.com");
// add conditions to QBC
criteria.add(cond);
//prepare Projections
Projection pFname=Projections.property("fname");
Projection pLname=Projections.property("lname");
//add Projections to ProjectionList
ProjectionList pList=Projections.projectionList();
pList.add(pFname); pList.add(pLname);
//set Projections(ProjectionList) to QBC logic
criteria.setProjection(pList);
//prepare and add Order
Order order=Order.asc("lname");
criteria.addOrder(order);
//Execute QBC logic
List <Object[]>list=criteria.list();
//process the List
for(Object row[]:list){
    for(Object val:row){
        System.out.print(val+" ");
    }
    System.out.println();
}
```

```
/*
/* //QBC logic performing aggregate operations
Criteria criteria=ses.createCriteria(EmpDetails.class);
// Prepare Projections
Projection rowCount=Projections.rowCount();
Projection noMax=Projections.max("no");
Projection noMin=Projections.min("no");
//Add Projections to ProjectionList
ProjectionList pList=Projections.projectionList();
pList.add(rowCount); pList.add(noMax); pList.add(noMin);
//add Projections to Criteria
criteria.setProjection(pList);
//execute QBC logic
List<Object[]>list=criteria.list();
//process the Result
Object result[]=list.get(0);
System.out.println("row count"+result[0]);
System.out.println("EID max value"+result[1]);
System.out.println("EID min value"+result[2]);
*/
//QBC logic for pagination
Criteria criteria=ses.createCriteria(EmpDetails.class);
//pagination settings
criteria.setFirstResult(2);
criteria.setMaxResults(2);
//execute QBC logic
List <EmpDetails>list=criteria.list();
//process the Result
for(EmpDetails ed:list)
System.out.println(ed);

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
}
```

**Note:**

- QBC logic takes given domain class name and its properties and prepares Metadata , uses that metadata against hibernate mapping file in memory metadata to get table name and column names and generates optimized Sql query using these table and column names.
- Condition in QBC logic will be represented by **Criterion object**. we can call various static methods of org.hibernate.criterion.Restriction(c) to get this Criterion object.The methods are like **Restrictions.and(-,-)**, **Restrictions.or(-,-)** **Restrictions.between(-,-)** and etc...
- When multiple Criterion objects are added to Criteria object ,they will be executing having **and clause between them**.

- We can use `restrictions.orMethod()` (or) `restrictions.andMethod()` to prepare criterian object having and, or clauses between conditions.

#### **Q. What is the different between Criteria object and Criterion object?**

Ans: Criteria object represents whole QBC logic, whereas Criterion object represents each condition of criteria object that should be added to Criteria object.

→ Criteria object means it is the object of that class that implements `org.hibernate.Criteria(i)`

→ Criterion object means it is the object the class that implements `org.hibernate.criterion.Criterion(i)`.

→ If you feel framing condition by using Restrictions class static methods is complex then u can frame the same conditions with the support of **SQL statement**. For this we need to use `Restriction.SqlRestriction(-)` as shown below.

#### **prepare SQL Query based condition**

```
Criterion cond=Restrictions.sqlRestriction(" email like '%gmail.com'");
//add condition
criteria.add(cond);
//execute QBC
List <EmpDetails> list=criteria.list();
```

#### **We can add the following objects on QBC Criteria object they are**

- Criteria object(for condition)
- Order object(for specifying asc and dsc order)
- Projection object(to get specific multiple or single column values or to get aggregate function results)

#### **QBC logic having Order,Criterion objs**

```
Criteria criteria=ses.createCriteria(EmpDetails.class);
//prepare Criterion objs
Criterion criterion=Restrictions.in("Iname",new String[]{"rao","rathod"});
// add Criterion obj
criteria.add(criterion);
//prpeare Order
Order order=Order.asc("fname");
//add ORder
criteria.addOrder(order);
//execute QBC logic
List <EmpDetails> list=criteria.list();
```

→ while working with Projections, if we want to retrieve specific single column value /multiple column values or if you want to perform aggregate operations on Database table you can use Projections.

#### **To work with projection we need to perform the following operations**

- Prepare set of Projection objects on 1 per property basis or 1 per aggregate function basis.
- Add all the Projection objects to `ProjectionList`.
- Set projectList to `Criteria` object.
- Execute QBC logic.

```
//QBC logic to get specific multiple col values from DATABASE table

Criteria criteria=ses.createCriteria(EmpDetails.class);

//prepre condtions

Criterion cond=Restrictions.ilike("mail","%GMAIL.com");

// add condtions to QBC

criteria.add(cond);

//prepare Projections

Projection pFname=Projections.property("fname");

Projection pLname=Projections.property("lname");

//add Projections to ProjectionList

ProjectionList pList=Projections.projectionList();

pList.add(pFname); pList.add(pLname);

//set Projections(ProjectionList) to QBC logic

criteria.setProjection(pList);

//prepare and add Order

Order order=Order.asc("lname");

criteria.addOrder(order);

//Execute QBC logic

List <Object[]>list=criteria.list();

//process the List

for(Object row[]:list){

    for(Object val:row){

System.out.print(val+" ");

    }

//for

    System.out.println();

}

//for
```

## Pagination:

→ The process of displaying huge amount of data part by part in multiple pages having next, previous hyperlinks or displaying page numbers to select a page is called pagination . This is very useful in report generation.

→ Generally we use the Pseudo column rownum support to perform pagination, but hibernate simplifies that process by using two methods they are:

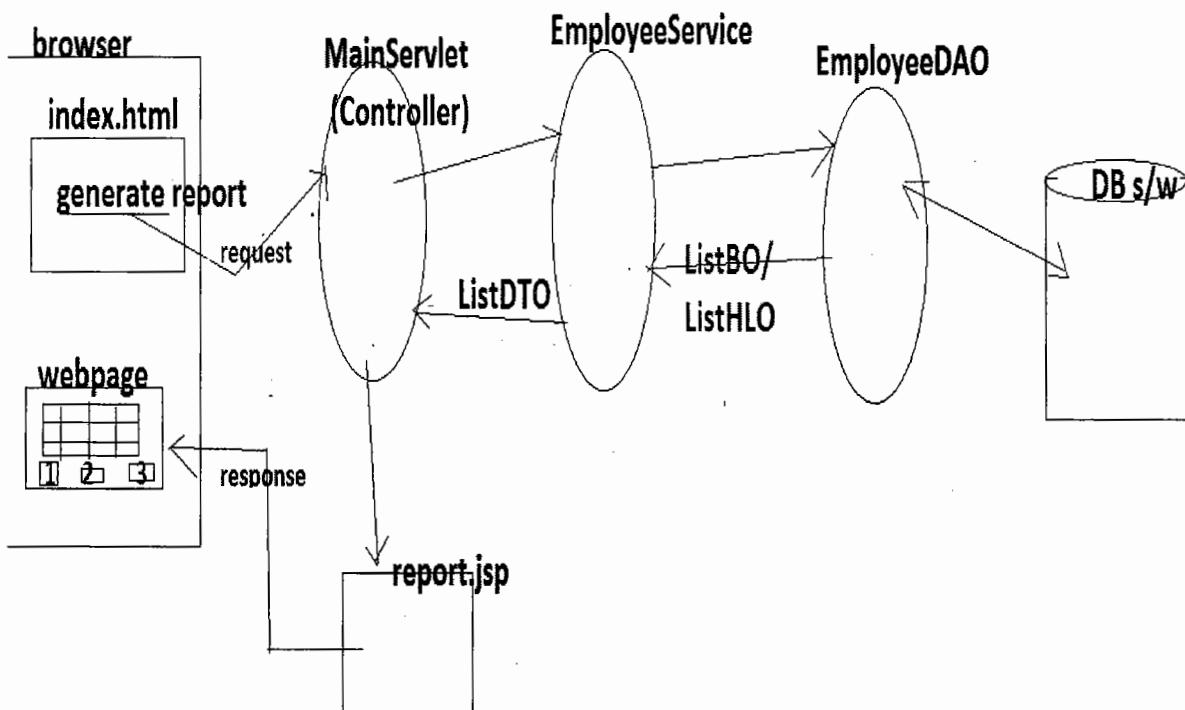
1. **setFirstResult(-)** → To specify the start position of record to display
2. **setMaxResults(-)** → To specify page size (to specify no of records to display from start position)

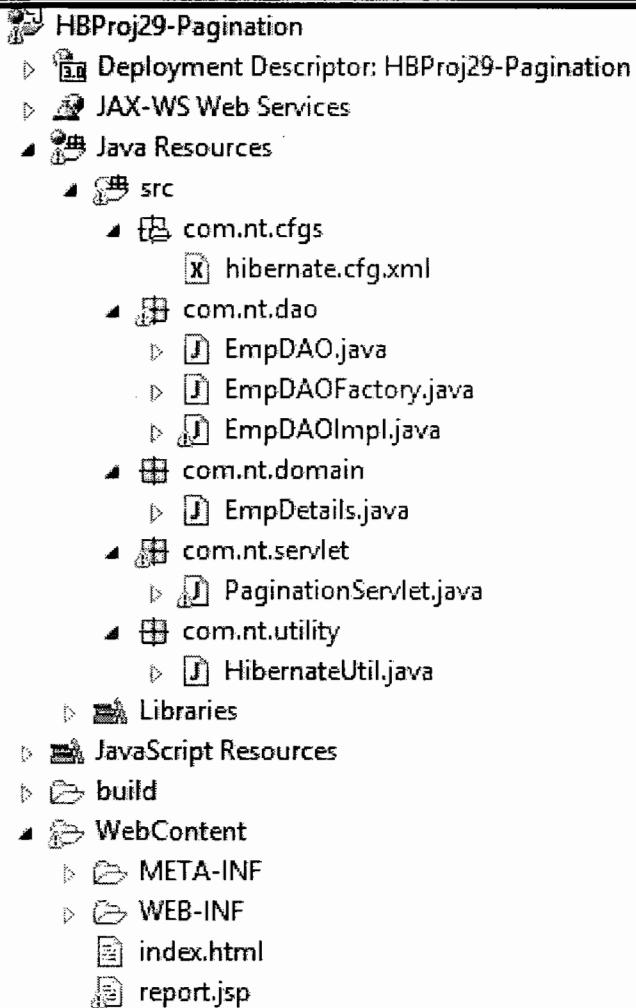
→ These two methods can be invoked on **Query object (HQL)** ,**SQLQuery object (native sql)** and **Criteria object(QBC)**. i.e all the 3 techniques support pagination.

//To get start position of record of each pages

Int start = (pageno\*pagesize)-pagesize

→ **PAGE\_SIZE** represents no.of records that should be displayed in a web page.





### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>

    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

    <mapping class="com.nt.domain.EmpDetails"/>
  </session-factory>
</hibernate-configuration>
```

**EmpDetails.java**

```
package com.nt.domain;

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

@Table(name="Employee")
@Entity
public class EmpDetails {
//domain class properties
@Id
@Column(name="EID")
private int no;
@Column(name="FIRSTNAME")
private String fname;
@Column(name="LASTNAME")
private String lname;
@Column(name="EMAIL")
private String mail;

public EmpDetails(){
System.out.println("EmpDetails:0-param constructor");
}

public int getNo() {
return no;
}
public void setNo(int no) {

this.no = no;
}
public String getFname() {
return fname;
}
public void setFname(String fname) {

this.fname = fname;
}
public String getLname() {
return lname;
}
public void setLname(String lname) {
```

```
this.lname = lname;
}
public String getMail() {

    return mail;
}
public void setMail(String mail) {

    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
```

**EmpDAO.java**

```
package com.nt.dao;

import java.util.List;

import com.nt.domain.EmpDetails;

public interface EmpDAO {
    public long getEmployeeCount();
    public List<EmpDetails> getEmployees(int startPos);
}
```

**EmpDAOImpl.java**

```
package com.nt.dao;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.criterion.Projection;
import org.hibernate.criterion.Projections;

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class EmpDAOImpl implements EmpDAO {
    private static final int PAGE_SIZE=3;

    @Override
    public long getEmployeeCount() {
        Session ses=null;
        Criteria criteria=null;
```

```
List<?> list=null;
Projection rowCount;
long recordsCount=0;
// get Session
ses=HibernateUtil.getSession();
//get RowCount using QBC logic + projections
criteria=ses.createCriteria(EmpDetails.class);
//Prepare Projection
rowCount=Projections.rowCount();
//add Projection
criteria.setProjection(rowCount);
//execute QBC
list=criteria.list();
recordsCount=(Long)list.get(0);
return recordsCount;
}

@Override
public List<EmpDetails> getEmployees(int startPos) {
Criteria criteria=null;
Session ses=null;
List<EmpDetails>list;
//get Session
ses=HibernateUtil.getSession();
//prepare Criteria obj
criteria=ses.createCriteria(EmpDetails.class);
//perform pagination
criteria.setFirstResult(startPos);
criteria.setMaxResults(PAGE_SIZE);
//execute QBC
list=criteria.list();
return list;
}//method

}//class
```

### **EmpDAOFactory.java**

```
package com.nt.dao;

public class EmpDAOFactory {
public static EmpDAO getInstance(){
    return new EmpDAOImpl();
}
}
```

PaginationServlet.java

```
package com.nt.servlet;
import java.io.IOException;
import java.util.List;

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.nt.dao.EmpDAO;
import com.nt.dao.EmpDAOFactory;
import com.nt.domain.EmpDetails;

public class PaginationServlet extends HttpServlet {
    private static final int PAGE_SIZE=3;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException,
    IOException {
        int pageNo=0;
        int startPos=0;
        List <EmpDetails> list;
        EmpDAO dao=null;
        int pagesCount=0;
        long rowCount=0;
        RequestDispatcher rd=null;
        //read page no
        pageNo=Integer.parseInt(req.getParameter("pageNo"));
        //get Start Position to begin the report
        startPos=(pageNo*PAGE_SIZE)-PAGE_SIZE;
        // get DAO obj
        dao=EmpDAOFactory.getInstance();
        //get Records to display
        list=dao.getEmployees(startPos);
        // get no.of pages display
        rowCount=dao.getEmployeeCount();
        pagesCount=(int)rowCount/PAGE_SIZE;
        if(pagesCount==0 || rowCount%PAGE_SIZE>0)
            pagesCount++;
        //keep List, pageCount in request attributes
        req.setAttribute("empList",list);
        req.setAttribute("pages",pagesCount);
        //forward request to report.jsp
        rd=req.getRequestDispatcher("report.jsp");
        rd.forward(req,res);
    } //doGet(-,-)
}
```

```

@Override
public void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException,
IOException {
    doGet(req,res);
}//doPost(-,-)

}//class

```

**HibernateUtil.java***Same as previous***web.xml**

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-
app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>HBProj29-Pagination</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
        <servlet-name>page</servlet-name>
        <servlet-class>com.nt.servlet.PaginationServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>page</servlet-name>
        <url-pattern>/pageurl</url-pattern>
    </servlet-mapping>
</web-app>

```

**result.jsp**

```

<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<h1><center> Employee Report</center></h1>
<c:choose>
    <c:when test="${empList ne null}">
        <table border="1">
            <tr><th>no</th> <th>first name</th><th>lastname </th><th>Email </th> </tr>
            <c:forEach var="ed" items="${empList}">
                <tr>
                    <td> <c:out value="${ed.no}"/></td>
                    <td> <c:out value="${ed.fname}"/></td>
                    <td> <c:out value="${ed.lname}"/></td>
                    <td> <c:out value="${ed.mail}"/></td>
                </tr>
            </c:forEach>
        </table>
    
```

```

</c:when>
<c:otherwise>
  <h3 style="color:red"> No Records found </h3>
</c:otherwise>
</c:choose>
<br><br><br>
<c:forEach var="i" begin="1" end="${pages}" step="1">
  <a href="pageurl?pageNo=<c:out value='${i}' />"><c:out
  value="${i}" /></a>&ampnbsp&ampnbsp&ampnbsp;
</c:forEach>

```

index.jsp

```
<a href="pageurl?pageNo=1">Generate Report</a>
```

**Note:**

While working with hibernate based pagination we use set first result, set maxresult() method these methods internally used the pseudo column rownum to complete pagination activites.

## Hibernate Filters

- These are no way related to servlet Filters.
- Hibernate filter is a named, global, parameterized condition that can be enabled or disabled in a hibernate session.
- Once the filter is enabled on hibernate Session all the HQL queries and QBC logics that are executed using in that Session will applied with hibernate filter condition. We can not apply filter on native SQL queries.
- To define hibernate filter parameters having name use `<filter-def>` tag. similarly to create hibernate filter condition use `<filter>` tag.
- To use enable hibernate filter on Session object use `ses.enableFilter(-)` method and similarly to disable hibernate filter on Session object use `ses.disableFilter(-)`.
- To provide global visibility to complex condition like filtering (avoid blocked account in the report generation/balance generation in the bank) we need to go for filter support.

**Example:**

- a) Keep any Hibernate application.
- b) Define hibernate filter in mapping file using `<filter-def>` tag.

In Employee.hbm.xml(after </class> tag)

```

<filter-def name="HB_ENO_RANGE_FILTER">
  <filter-param name="min" type="int"/>
  <filter-param name="max" type="int"/>
</filter-def>

```

- c) Prepare filter condition with respect to domain class using `<filter>` tag in mapping file.

In Employee.hbm.xml (before </class> tag)

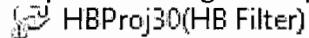
```
<filter name="HB_ENO_RANGE_FILTER" condition="EID>=:min and EID<=:max"/>
```

- d) Enable filter on Hibernate Session object and specify filter param to execute queries with filter condition in ClientApp or in DAO class.

- Hibernate filter condition is visible in multiple session objects of one or more client application.

### Note

Generally industry does not prefer working with filters becoz, it makes programmer to write queries and conditions separately due to this the maintenance of the application becoz very complex, so use hibernate filters only when dealing with complex and common conditions that are required through out applications (like not involving blocked accounts in report generation).



- ▶ src
  - ▶ com.nt.cfgs
    - ☒ Employee.hbm.xml
    - ☒ hibernate.cfg.xml
  - ▶ com.nt.domain
    - ▷ EmpDetails.java
  - ▶ com.nt.test
    - ▷ FilterTest.java
  - ▶ com.nt.utility
    - ▷ HibernateUtil.java

### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

### Employee.hbm.xml

```
<!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.domain.EmpDetails" table="Employee">
        <id name="no" column="EID"/>
        <property name="fname" >
            <column name="FIRSTNAME"/>
        </property>
    </class>
</hibernate-mapping>
```

```
</property>
<property name="lname" >
    <column name="LASTNAME"/>
</property>
<property name="mail" >
    <column name="EMAIL"/>
</property>
<filter name="HB_ENO_RANGE_FILTER" condition="EID>=:min and EID<=:max"/>
</class>
<filter-def name="HB_ENO_RANGE_FILTER">
    <filter-param name="min" type="int"/>
    <filter-param name="max" type="int"/>
</filter-def>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;

    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
}
```

```

}

public void setLname(String lname) {

    this.lname = lname;
}
public String getMail() {

    return mail;
}
public void setMail(String mail) {

    this.mail = mail;
}

@Override
public String toString() {
    return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}

```

**HibernateUtil.java****Same as previous****FilterTest.java**

```

package com.nt.test;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Filter;
import org.hibernate.Session;

import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class FilterTest {
    public static void main(String[] args) {
        //Get Session
        Session ses=HibernateUtil.getSession();

        //enable filter
        Filter filter=ses.enableFilter("HB_ENO_RANGE_FILTER");
        //set params value to filter
        filter.setParameter("min",100);
        filter.setParameter("max",200);
        /*// Exeecute HQL Query
        Query query=ses.createQuery("from EmpDetails");

```

```

List<EmpDetails>list=query.list();
for(EmpDetails ed:list){
    System.out.println(ed);
}
//disable filter
ses.disableFilter("HB_ENO_RANGE_FILTER");
Query query1=ses.createQuery("select count(*) from EmpDetails");
List<?> list1=query1.list();
System.out.println("Records count"+list1.get(0));
/*//Using Native SQL
SQLQuery query=ses.createSQLQuery("select * from Employee");
query.addEntity(EmpDetails.class);
List <EmpDetails>list=query.list();
for(EmpDetails ed:list){
    System.out.println(ed);
}*/



//Using QBC
Criteria criteria=ses.createCriteria(EmpDetails.class);
List<EmpDetails> list=criteria.list();
for(EmpDetails ed:list){
    System.out.println(ed);
}

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
}

```

- Hibernate conditions are globally visible in multiple Session objects that are created in multiple clients or multiple DAO classes.
- Hibernate filter can be enabled on HQL query logic and QBC logic, but can't be enabled on native SQL logic.

### Annotation driven hibernate filters:

→ Filters are hibernate specific features ,so there are no JPA annotations to work with hibernate filters i.e. we need to use hibernate specific annotations they are:

- @FilterDefs : To group multiple filter definitions.
- @FilterDef: To define each filter definitions.
- @ParamDef: To define each parameter hibernate definitions
- @Filters: To group multiple Hibernate filter conditions
- @Filter: To prepare each hibernate filter conditions
  
- We can place multiple filter definitions and conditions in one hibernate mapping file while working with xml based programming, similarly we can place multiple filters condition and definitions in the domain class while working with annotation driver hibernate programming.

- @Filter, @FilterDef annotations are not repeatable annotations so to group multiple filter definitions and conditions @FilterDefs, @Filters are given.

### HBProj31(Anno-HB Filters)

```

    ▲ src
      ▲ com.nt.cfgs
        [x] hibernate.cfg.xml
      ▲ com.nt.domain
        ▶ [J] Student.java
      ▲ com.nt.test
        ▶ [J] HBFilterTest.java
      ▲ com.nt.utility
        ▶ [J] HibernateUtil.java

```

#### hibernate.hbm.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property
      name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

    <mapping class="com.nt.domain.Student"/>

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

```

#### Student.java

```

package com.nt.domain;

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

import org.hibernate.annotations.Filter;
import org.hibernate.annotations.FilterDef;
import org.hibernate.annotations.ParamDef;

```

```
@Entity
@Table(name="student_Tab")
@FilterDef(name="SNO_RANGE",
    parameters={@ParamDef(name="min",type="int"),
                @ParamDef(name="max",type="int")}
)
@Filter(name="SNO_RANGE",
    condition="STNO>=:min and STNO<=:max")
public class Student {

    private int sno;

    private String sname;

    private String address;
    //write setters and getters
    @Id
    @Column(name="stno")
    public int getSno() {
        return sno;
    }

    public void setSno(int sno) {
        this.sno = sno;
    }

    @Column(name="stname")
    public String getName() {
        return sname;
    }

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

    @Column(name="stadd")
    @Transient
    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";
    }
}
```

**HibernateUtil.java**

Same as previous

**HBFilterTest.java**

```
package com.nt.test;

import java.util.List;

import org.hibernate.Filter;
import org.hibernate.Query;
import org.hibernate.Session;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class HBFilterTest {
    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();
        //Enable Filter
        Filter filter=ses.enableFilter("SNO_RANGE");
        //set filter params
        filter.setParameter("min",1000);
        filter.setParameter("max",1010);
        //Execute HQL logics
        Query query=ses.createQuery("from Student");
        List<Student> list=query.list();
        for(Student st:list){
            System.out.println(st);
        }
        //close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

**Hibernate supports different modes of O-R mapping they are:**

- a) **Basic O-R mapping:** mapping class with database table and properties with Database table columns.
- b) **Component mapping:** When domain classes are there in composition.
- c) **Inheritance mapping:** When domain classes are there in the inheritance.
- d) **Association mapping:** When domain classes taken in relationships like one to many, many to one, many to many, one to one and etc...
- e) **Collection mapping**

## Component Mapping

### Composition (HAS-A relationship)

```
class A{
private B b = new B();
}

class B{
.....}
.....
```

### Inheritance: (IS-A relationship)

```
class A extends B{
.....
.....
}

class B{
.....
.....
}
```

→ We use inheritance when one class wants to use all the functionality of another class. Using inheritance we can't make one class utilizing the functionalities of multiple classes.

→ We go for composition when one class wants to use the partial functionality of another one or more classes.

### Q. What is the problem of granularity in O-R mapping?

Ans: When multiple classes designed based on composition and they want to be mapped with single DATABASE table i.e. more granuales (classes) wants to map with single DATABASE table but we fail to do that mapping using **O-R mapping**, this is nothing but problem of granularity.

```
public class Person{
    private int id;
    private String pname;
    private JobType pjob;
    //setters and getters
}

public class JobType{
    private String job;
    private int department;
    private double salary;
    //setters and getters
}
```

- In the above designing **JobType** class is taken separately having properties to holds job details to use that class in multiple main classes like **Person**, **LoanInfo**, **PersonDetails** and etc... classes through composition.
- We can design hibernate domain classes in composition but we can't make DATABASE tables staying in composition. This is nothing but multiple classes that are in composition pointing to single table (**problem of granularity**). Use component mapping to solve that problem.
- If property is mapped with single column of DATABASE table then it is called **simple property**.
- If property is mapped with multiple columns of DATABASE table then it is called **component property**.
- Use **<property>** tag to configure simple property and use **<component>** tag to configure component property.

```
<component name="pjob" class="com.nt.domain.JobType">
  <property name="job"/>
  <property name="department"/>
  <property name="salary"/>
</component>
```

### HBProj32(Component mapping)

- ▶ src
  - ▶ com.nt.cfgs
    - ☒ hibernate.cfg.xml
    - ☒ person.hbm.xml
  - ▶ com.nt.domain
    - ▷ J JobType.java
    - ▷ J Person.java
  - ▶ com.nt.test
    - ▷ J InsertTest.java
    - ▷ J SelectTest.java
  - ▶ com.nt.utility
    - ▷ J HibernateUtil.java

### hibernate.hbm.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/person.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

```
<!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.domain.Person" table="person_tab">
    <id name="pid"/>
    <property name="pname"/>
    <component name="pjob" class="com.nt.domain.JobType">
```

```
<property name="job"/>
<property name="department"/>
<property name="salary"/>
</component>
</class>
</hibernate-mapping>
```

**JobType.java**

```
package com.nt.domain;

public class JobType {
    private String job;
    private int department;
    private double salary;

    public String getJob() {
        return job;
    }

    public void setJob(String job) {
        this.job = job;
    }

    public int getDepartment() {
        return department;
    }

    public void setDepartment(int department) {
        this.department = department;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

**Person.java**

```
package com.nt.domain;
public class Person {
    private int pid;
    private String pname;
    private JobType pjob;

    public int getPid() {
        return pid;
    }

    public void setPid(int pid) {
        this.pid = pid;
    }
}
```

```

}
public String getPname() {
return pname;
}
public void setPname(String pname) {
this.pname = pname;
}
public JobType getPjob() {
return pjob;
}
public void setPjob(JobType pjob) {
this.pjob = pjob;
}
}

```

**HibernateUtil.java**

Same as previous

**InsertTest.java**

```

package com.nt.test;

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

import com.nt.domain.JobType;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class InsertTest {

public static void main(String[] args) {
// get Session
Session ses=HibernateUtil.getSession();
//save objs
JobType job=new JobType();
job.setJob("clerk"); job.setDepartment(1001); job.setSalary(5000);
Person person=new Person();
person.setPid(111); person.setPname("raja"); person.setPjob(job);

Person person1=new Person();
person1.setPid(112); person1.setPname("ravi"); person1.setPjob(job);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(person);
ses.save(person1);
tx.commit();
}
}

```

```
//try
catch(Exception e){
    tx.rollback();
}
//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
}
```

### SelectTest.java

```
package com.nt.test;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;

import com.nt.domain.JobType;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class SelectTest {

    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();
        //Retrieve objs
        /*Query query=ses.createQuery("from Person");
        List<Person> list=query.list();
        for(Person per:list){
            System.out.println("Person details..."+per.getPid()+" "+per.getPname());
            JobType job=per.getPjob();
            System.out.println("Job details ..."+job.getJob()+" "+job.getDepartment()+" "+job.getSalary());
        }
        */
        //Retrieve objs
        Query query=ses.createQuery("from Person where pjob.department=:dept");
        query.setInteger("dept",1001);
        List<Person> list=query.list();
        for(Person per:list){
            System.out.println("Person details..."+per.getPid()+" "+per.getPname());
            JobType job=per.getPjob();
            System.out.println("Job details ..."+job.getJob()+" "+job.getDepartment()+" "+job.getSalary());
        }
        //Close objs
        HibernateUtil.closeSession();
    }
}
```

```

    HibernateUtil.closeSessionFactory();
}

}

```

## Annotation driven component mapping:

- **@EmbeddedId** : use to configure component property like pjob
- **@Embeddable**: use to configure helper class that acts component class having sub properties like **JobType**.
- **Note** : Both are JPA annotations.

### HBProj33(Anno-Component mapping)

```

    ▲ src
      ▲ com.nt.cfgs
        X hibernate.cfg.xml
      ▲ com.nt.domain
        J JobType.java
        J Person.java
      ▲ com.nt.test
        J InsertTest.java
        J SelectTest.java
      ▲ com.nt.utility
        J HibernateUtil.java

```

### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Person"/>
  </session-factory>
</hibernate-configuration>

```

JobType.java

```
package com.nt.domain;

import javax.persistence.Embeddable;

@Embeddable
public class JobType {
    private String job;
    private int department;
    private double salary;
    public String getJob() {
        return job;
    }
    public void setJob(String job) {
        this.job = job;
    }
    public int getDepartment() {
        return department;
    }
    public void setDepartment(int department) {
        this.department = department;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```

Person.java

```
package com.nt.domain;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="person_tab")
public class Person {
    @Id
    private int pid;
    private String pname;
    @Embedded
    private JobType pjob;

    public int getPid() {
```

```
return pid;
}
public void setPid(int pid) {
this.pid = pid;
}
public String getPname() {
return pname;
}
public void setPname(String pname) {
this.pname = pname;
}
public JobType getPjob() {
return pjob;
}
public void setPjob(JobType pjob) {
this.pjob = pjob;
}

}
```

**HibernateUtil.java**

Same as previous...

**InsertTest.java**

```
package com.nt.test;

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

import com.nt.domain.JobType;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class InsertTest {

public static void main(String[] args) {
// get Session
Session ses=HibernateUtil.getSession();
//save objs
JobType job=new JobType();
job.setJob("clerk"); job.setDepartment(1001); job.setSalary(5000);
Person person=new Person();
person.setPid(111); person.setPname("raja"); person.setPjob(job);

Person person1=new Person();
person1.setPid(112); person1.setPname("ravi"); person1.setPjob(job);
Transaction tx=null;
try{
tx=ses.beginTransaction();
```

```

ses.save(person);
ses.save(person1);
tx.commit();
}//try
catch(Exception e){
    tx.rollback();
}
//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
}

```

**SelectTest.java**

```

package com.nt.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import com.nt.domain.JobType;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;
public class SelectTest {
    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();
        //Retrieve objs
        /*Query query=ses.createQuery("from Person");
        List<Person> list=query.list();
        for(Person per:list){
            System.out.println("Person details... "+per.getPid()+" "+per.getPname());
        }
        JobType job=per.getPjob();
        System.out.println("Job details ... "+job.getJob()+" "+job.getDepartment()+" "+job.getSalary());
    }
    */
        //Retrieve objs
        Query query=ses.createQuery("from Person where pjob.department=:dept");
        query.setInteger("dept",1001);
        List<Person> list=query.list();
        for(Person per:list){
            System.out.println("Person details... "+per.getPid()+" "+per.getPname());
            JobType job=per.getPjob();
            System.out.println("Job details ... "+job.getJob()+" "+job.getDepartment()+" "+job.getSalary());
        }
        //Close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}

```

## Inheritance mapping

→ Since domain classes are POJO classes, they can be designed through inheritance to take the advantage of extensibility, but we can't keep Database tables of these domain classes participating in inheritance because there is no table level inheritance in Database. This is called **sub types problem** which is nothing but the one or more classes of inheritance hierarchy cannot be mapped with one or more DATABASE tables of DATABASE software.

→ To solve this **subtypes** problem use inheritance mapping i.e. we can guide hibernate framework to map the domain classes of inheritance with one or more DATABASE tables of DATABASE software.

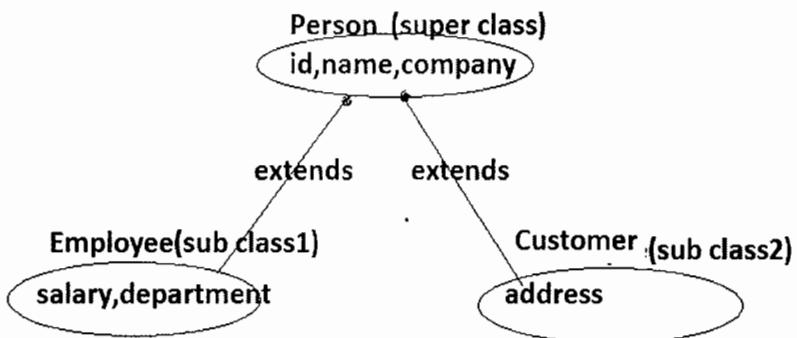
-> Since domain class or ordinary java classes they can use all java features like inheritance and composition, but database tables does not support some features so to guide hibernate for handling above situation we go for component mapping, inheritance mapping concepts.

**Hibernate supports inheritance mapping in 3 strategies /levels:**

1. Table per class hierarchy.
2. Table per sub classes
3. Table per concrete class hierarchy.

### Strategy1: Table per class hierarchy

- Here the classes of inheritance hierarchy will be mapped with single DATABASE table i.e object/record can be saved in DATABASE table by using any class of inheritance hierarchy.
- DATABASE table maintains one **discriminator column** holding values that indicates using which domain class of inheritance is used to save the record/object in DATABASE table.

**Table per class hierarchy**

All the classes of inheritance hierarchy will use single Db table

=> We need to place one desriminator column in Db table to maintain desriminator value indicating which domain class of inheritance hirerchy as inserted each record of DB table.

=>While cfg each domain class we need to specify desriminator value .This will be inserted as "value" of desriminator column every record is inserted.

inh_persons (db table)						(Desriminator col)
id(pk)	name	company	salary	department	address	person_type
101	raja	HCL	--	--	--	per
102	ravi	Wipro	90000	1001	....	emp
103	rakesh	TCS	---	---	Hyd	cus

=>Desriminator col will not be mapping with domain class properties..

=>Use `<class>` tag to cfg super class, use `<subclass>` tags to cfg sub classes , use `<desriminator>` tag to cfg desriminator col and "desriminator-value" of `<class>`, `<subclass>` tags to cfg desriminator values.

Mapping file

```

<class name="pkg.Person" table="in_persons4" desriminator-value="per">
  <id name="id"/>
  <desriminator column="person_type" type="string"/>
  <property name="name"/>
  <property name="company"/>
  <subclass name="pkg.Employee" desriminator-value="emp">
    <property name="salary"/>
    <property name="department"/>
  </subclass>
  <subclass name="pkg.Customer" desriminator-value="cus">
    <property name="address"/>
  </subclass>

```

- Discriminator column must be configured using `<desriminator>` tag.
- Use `<class>` tag to cfg super class and `<sub-class>` tags to configure sub classes inside `<class>` tag. Use **desriminator-value** attribute to specify desriminator value for each class of inheritance hierarchy.
- Discriminator column will not be mapped with any property of domain class.

Mapping file

```
<class name="com.nt.domain.Person"  table="in_persons"  discriminator-value="per">
    <id name="id"/>
    <discriminator column="person_type" type="string"/>
    <property name="name"/>
    <property name="company"/>
    <subclass name="com.nt.domain.Employee"  discriminator-value="emp">
        <property name="salary"/>
        <property name="department"/>
    </subclass>
    <subclass name="com.nt.domain.Customer"  discriminator-value="cust">
        <property name="address"/>
    </subclass>
</class>
```

Limitations:

- The DATABASE table has to hold multiple values like all the property values of all classes in inheritance hierarchy. Designing table with huge number of columns is a **bad practice**.
- We can't apply **not-null** constraint on the columns represented by sub class properties.
- To manipulate discriminator column values we need to use **native Sql**.

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

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

```

<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">scott</property>
<property name="connection.password">tiger</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/person.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

```

<!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.domain.Person"  table="in_persons"  discriminator-value="per">
  <id name="id"/>
  <discriminator column="person_type" type="string"/>
  <property name="name"/>
  <property name="company"/>
  <subclass name="com.nt.domain.Employee"  discriminator-value="emp">
    <property name="salary"/>
    <property name="department"/>
  </subclass>
  <subclass name="com.nt.domain.Customer"  discriminator-value="cust">
    <property name="address"/>
  </subclass>
</class>
</hibernate-mapping>

```

### Customer.java

```

package com.nt.domain;

public class Customer extends Person {
  private String address;
  public String getAddress() {
    return address;
  }
  public void setAddress(String address) {
    this.address = address;
  }
  @Override
  public String toString() {
    return "Customer [address=" + address + "]" + super.toString();
  }
}

```

**Employee.java**

```
package com.nt.domain;

public class Employee extends Person {
    private double salary;
    private int department;
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public int getDepartment() {
        return department;
    }
    public void setDepartment(int department) {
        this.department = department;
    }
    @Override
    public String toString() {
        return "Employee [salary=" + salary + ", department=" + department + "]";
    }
}
```

**Person.java**

```
package com.nt.domain;
public class Person {
    private int id;
    private String name;
    private String company;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
        this.company = company;
    }
}
```

```
@Override
public String toString() {
return "Person [id=" + id + ", name=" + name + ", company=" + company + "]";
}
}
/* create table in_persons (id number(5) primary key,
name varchar2(20),
company varchar2(20),
salary number(5),
department number(5),
address varchar2(20),
person_type varchar(10)
)
*/
```

#### InhDAO.java

```
package com.nt.dao;

public interface InhDAO {
public void saveData();
public void displayData();

}
```

#### InhDAOImpl.java

```
package com.nt.dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Customer;
import com.nt.domain.Employee;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class InhDAOImpl implements InhDAO {

@Override
public void saveData() {
//Get Session
Session ses=HibernateUtil.getSession();
//Save objs
Person person=new Person();
person.setId(1001); person.setName("raja"); person.setCompany("HCL");
}
```

```
Employee emp=new Employee();
emp.setId(1002); emp.setName("jani");emp.setCompany("InfoSys"); emp.setSalary(9000);
emp.setDepartment(1001);

Customer customer=new Customer();
customer.setId(1003); customer.setName("rani"); customer.setCompany("WIPRO");
customer.setAddress("hyd");
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(person);
    ses.save(emp);
    ses.save(customer);
    tx.commit();
    System.out.println(" Objects are Saved");
} //try
catch(Exception e){
    tx.rollback();
}
HibernateUtil.closeSession();
}//saveData()

@Override
public void displayData() {
    //Get Session
Session ses=HibernateUtil.getSession();
//Execute HQL Query
Query query1=ses.createQuery("from Person");
List<Person> list1=query1.list();
for(Person per:list1){
    System.out.println(per);
} //for

System.out.println("-----");
Query query2=ses.createQuery("from Employee");
List<Employee> list2=query2.list();
for(Employee emp:list2){
    System.out.println(emp);
} //for

System.out.println("-----");
Query query3=ses.createQuery("from Customer");
List<Customer> list3=query3.list();
for(Customer cust:list3){
    System.out.println(cust);
} //for
}//displayData

}//class
```

InhDAOFactory.java

```
package com.nt.dao;

public class InhDAOFactory {
    public static InhDAO getInstance(){
        return new InhDAOImpl();
    }
}
```

HibernateUtil.java

Same as above

ClientApp.java

```
package com.nt.test;

import com.nt.dao.InhDAO;
import com.nt.dao.InhDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {
    public static void main(String[] args) {
        // Access DAO
        InhDAO dao=InhDAOFactory.getInstance();
        //call b.method
        //dao.saveData();

        //call b.method
        dao.displayData();

        //Close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();

    }//main
}//class
```

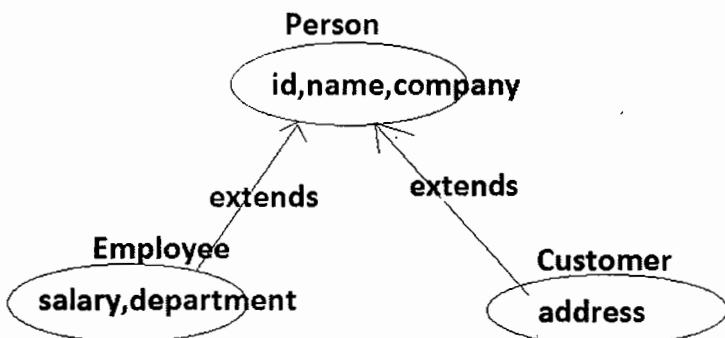
## Strategy2: (Table per sub class Inheritance mapping)

- Here every class of inheritance hierarchy will have its own DATABASE table and these DATABASE tables will be there in relationship.
- The DATABASE table mapped by super class act as **parent table** and the Database tables mapped by sub classes act as **child tables**.
- While saving data by using any class object of inheritance hierarchy the common properties data will be saved to parent table and sub classes properties data will be saved to child tables.

Table per sub class (Inheritance mapping : strategy no:2)

=> Every class of inheritance hierarchy will maintain its own db table having one to one relationship with parent table.

=> The db table of super class will maintain one to one association with the db tables of sub classes using FK column support.

in\_persons2( parent db table)

id(pk)	name	company	using
101	raja	HCL	Person
102	ramesh	Wipro	Employee
103	karan	POLARIS	

in\_employee2 (child table1)

department	salary	person_id(FK)
9999	9000	102

address	person_id(FK)
hyd	103

- Use **<class>** to configure super class and use **<joined-subclass>** tag to configure sub classes.
- Every configuration of domain class specifies its own related DATABASE table name.
- Since every class of inheritance hierarchy maintains its own DATABASE table there is no need of configuring **discriminator column name**.
- While configuring every sub class we need specify **FK column** of child DATABASE tables.

In mapping file:

```

<class name="com.nt.domain.Person"  table="in_persons2" >
    <id name="id"/>
    <property name="name"/>
    <property name="company"/>
    <joined-subclass name="com.nt.domain.Employee" table="in_employees2">
        <key column="person_id"/>
        <property name="salary"/>
        <property name="department"/>
    </joined-subclass>
    <joined-subclass name="com.nt.domain.Customer"  table="in_customers2" >
  
```

```

<key column="person_id"/>
<property name="address"/>
</joined-subclass>
</class>

```

### Advantages of strategy2:

- Instead of taking huge number of columns in one Database table, multiple tables are taken having relationship, which is good industry practice.
- No discriminator column is required.
- We can apply **not-null constraint** on the columns represented by **sub class properties**.

**Note:** Because of the entire above advantages and best table designing support, strategy no 2 is most recommend strategy of **inheritance mapping**.

**Note:** While working with inheritance mapping it is always suggested not to use super class of inheritance hierarchy in persistence operations. It is always recommended to use that class just containing common properties ,if possible it is suggested to take that class as **abstract class**.

### HBProj35(HibernateMapping2-TPSC)

```

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
    - person.hbm.xml
  - com.nt.dao
    - InhDAO.java
    - InhDAOFactory.java
    - InhDAOImpl.java
  - com.nt.domain
    - Customer.java
    - Employee.java
    - Person.java
  - com.nt.test
    - ClientApp.java
  - com.nt.utility
    - HibernateUtil.java

```

### Hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

```

```

<hibernate-configuration>
  <session-factory>
    <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">scott</property>

```

```

<property name="connection.password">tiger</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/person.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

```

<!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.domain.Person"  table="in_persons2" >
  <id name="id"/>
  <property name="name"/>
  <property name="company"/>
  <joined-subclass name="com.nt.domain.Employee" table="in_employees2">
    <key column="person_id"/>
    <property name="salary"/>
    <property name="department"/>
  </joined-subclass>
  <joined-subclass name="com.nt.domain.Customer"  table="in_customers2" >
    <key column="person_id"/>
    <property name="address"/>
  </joined-subclass>
</class>
</hibernate-mapping>

```

### ***Customer.java***

```

package com.nt.domain;

public class Customer extends Person {
  private String address;

  public String getAddress() {
    return address;
  }

  public void setAddress(String address) {
    this.address = address;
  }
  @Override
  public String toString() {
    return super.toString()+"Customer [address=" + address + "]";
  }
}

```

**Employee.java**

```
package com.nt.domain;

public class Employee extends Person {
    private double salary;
    private int department;
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public int getDepartment() {
        return department;
    }
    public void setDepartment(int department) {
        this.department = department;
    }
    @Override
    public String toString() {
        return super.toString()+"Employee [salary=" + salary + ", department=" + department + "]";
    }
}
```

**Person.java**

```
package com.nt.domain;

public abstract class Person {
    private int id;
    private String name;
    private String company;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
```

```

this.company = company;
}
@Override
public String toString() {
return "Person [id=" + id + ", name=" + name + ", company=" + company + "]";
}
}
/*
 * create table in_persons2(id number(5) primary key,
 *                         name varchar2(20),
 *                         company varchar2(20)
 * )
 * create table in_employee2(salary number(8),department number(8),
 *                           person_id number(5) references in_persons2(id)
 * )
 * create table in_customers2(address varchar2(20),
 *                            person_id number(5) references in_persons2(id)
 * )
 */

```

**InhDAO.java**

```

package com.nt.dao;

public interface InhDAO {

    public void saveData();
    public void displayData();

}

```

**InhDAOImpl.java**

```

package com.nt.dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Customer;
import com.nt.domain.Employee;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class InhDAOImpl implements InhDAO {

    @Override
    public void saveData() {
        //Get Session

```

```
Session ses=HibernateUtil.getSession();
//Save objs
/* Person person=new Person();
person.setId(1001); person.setName("raja"); person.setCompany("HCL"); */

Employee emp=new Employee();
emp.setId(1002); emp.setName("jani");emp.setCompany("InfoSys"); emp.setSalary(9000);
emp.setDepartment(1001);

Customer customer=new Customer();
customer.setId(1003); customer.setName("rani"); customer.setCompany("WIPRO");
customer.setAddress("hyd");
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(person);
ses.save(emp);
ses.save(customer);
tx.commit();
System.out.println(" Objects are Saved");
}//try
catch(Exception e){
tx.rollback();
}
HibernateUtil.closeSession();
}//saveData()

@Override
public void displayData() {
//Get Session
Session ses=HibernateUtil.getSession();
//Execute HQL Query
Query query1=ses.createQuery("from Person");
List<Person> list1=query1.list();
for(Person per:list1){
System.out.println(per);
}//for

System.out.println("-----");
Query query2=ses.createQuery("from Employee");
List<Employee> list2=query2.list();
for(Employee emp:list2){
System.out.println(emp);
}//for

System.out.println("-----");
Query query3=ses.createQuery("from Customer");
List<Customer> list3=query3.list();
```

```
for(Customer cust:list3){  
    System.out.println(cust);  
}//for  
}//displayData  
  
}//class
```

**InhDAOFactory.java**

```
package com.nt.dao;  
  
public class InhDAOFactory {  
  
    public static InhDAO getInstance(){  
        return new InhDAOImpl();  
    }  
  
}
```

**HibernateUtil.java**

```
Same
```

**ClientApp.java**

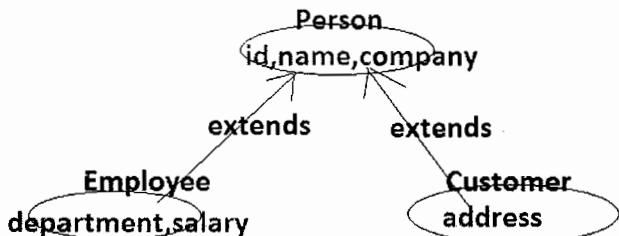
```
package com.nt.test;  
  
import com.nt.dao.InhDAO;  
import com.nt.dao.InhDAOFactory;  
import com.nt.utility.HibernateUtil;  
  
public class ClientApp {  
    public static void main(String[] args) {  
        // Access DAO  
        InhDAO dao=InhDAOFactory.getInstance();  
        //call b.method  
        dao.saveData();  
  
        //call b.method  
        //dao.displayData();  
  
        //Close SessionFactory  
        HibernateUtil.closeSessionFactory();  
  
    }//main  
}
```

## Strategy no 3 :( Table per concrete class or union class)

Here every class of inheritance hierarchy will have its own DATABASE table and these DATABASE tables do not stay in relationship, due to this multiple Database tables maintain duplicate columns.

### Table per concrete class (Inheritance mapping : strategy no:3)

=> Every class of inheritance hierarchy will have its own db table but these db tables will not be there in the relationship So db tables will endup with duplicate cols



in\_persons3 (table1)

<u>id(pk)</u>	<u>name</u>	<u>company</u>	
101	raja	HCL	using Person

in\_customers3(table2)

<u>id(pk)</u>	<u>name</u>	<u>company</u>	<u>address</u>
103	ramesh	TATA	hyd Using Customer

in\_employee2(table3)

<u>id(pk)</u>	<u>name</u>	<u>company</u>	<u>department</u>	<u>salary</u>	
102	karan	POLARIS	1001	9000	using Employee

=>Here table2,table3 are having duplicate cols that are already there in table , So t is not a good pratice.

=> Since every domain class inheritance hierarchy is having its own db table,So there is no need of desriminator column.

### Version1 Cfg:

->use multiple independent `<class>` tags to cfg super class and sub classes but while cfg sub class we need to both inherited and specific properties.

### Version2 Cfg:

use `<class>` tag to cfg super class and use `<union-subclass>` tags to cfg sub classes here there is no need of cfg iherited properties while cfg sub classes.

**In mapping file (version1)**

```
<class name="pkg.Person" table="in_person3">
.....
.... //super class 3 properties
.....
</class>

<class name="pkg.Employee" table="in_employees3">
.....
..... super class 3 properties
.....
.... sub class1 2 properties
.....
</class>

<class name="pkg.Customer" table="in_customers3">
.....
.... super class 3 properties
.....
... //sub class2 1 property
</class>
```

## In mappigle (version2)

```
<class name="pkg.Person" table="in_persons3">
    ...
    ... //super class properties
    ...
    <union-subclass name="pkg.Employee" table="in_employee3">
        ... //sub class1 2 properties
        ...
    </union-subclass>
    <union-subclass name="pkg.Customer" table="in_customers3">
        ...
        ... //sub class2 1 property
    </union-subclass>
</class>
```

- Since every class of inheritance hierarchy is having its own DATABASE table and there is no need of discriminator column.
- We can configure all the classes by using separate multiple `<class>` tags or we can use `<class>` tag to configure super class and `<union-subclass>` tags to configure sub classes.

## mapping file (version1)

```
<class name="com.nt.domain.Person" table="in_persons3" >
    <id name="id"/>
    <property name="name"/>
    <property name="company"/>
</class>
<class name="com.nt.domain.Employee" table="in_employees3">
    <id name="id"/>
    <property name="name"/>
    <property name="company"/>
    <property name="salary"/>
    <property name="department"/>
</class>
<class name="com.nt.domain.Customer" table="in_customers3" >
    <id name="id"/>
    <property name="name"/>
    <property name="company"/>
    <property name="address"/>
</class>
```

mapping file(Version2)

```
<class name="com.nt.domain.Person"  table="in_persons3">
    <id name="id"/>
    <property name="name"/>
    <property name="company"/>
<union-subclass name="com.nt.domain.Employee" table="in_employees3">
    <property name="salary"/>
    <property name="department"/>
</union-subclass>
<union-subclass name="com.nt.domain.Customer" table="in_customers3" >
    <property name="address"/>
</union-subclass>
</class>
```

Advantages:

- No need of configuring discriminator column.
- We can apply **not-null** constraint on the columns of sub class properties.

Limitations:

- DATABASE table of sub classes contains huge number of columns (both columns of common properties and uncommon properties),which is a bad practice.
- The columns of common properties will be repeated in multiple Database tables which is a **bad Database designing**.

## HBProj36(HnMapping3-TPCC)

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/person.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

```
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<!-- &lt;class name="com.nt.domain.Person"  table="in_persons3" &gt;
  &lt;id name="id"/&gt;
  &lt;property name="name"/&gt;
  &lt;property name="company"/&gt;
&lt;union-subclass name="com.nt.domain.Employee"  table="in_employees3"&gt;
  &lt;property name="salary"/&gt;
  &lt;property name="department"/&gt;
&lt;/union-subclass&gt;
&lt;union-subclass name="com.nt.domain.Customer"  table="in_customers3" &gt;
  &lt;property name="address"/&gt;
&lt;/union-subclass&gt;
&lt;/class&gt;--&gt;
&lt;class name="com.nt.domain.Person"  table="in_persons3" &gt;
  &lt;id name="id"/&gt;
  &lt;property name="name"/&gt;
  &lt;property name="company"/&gt;
&lt;/class&gt;
&lt;class name="com.nt.domain.Employee"  table="in_employees3"&gt;
  &lt;id name="id"/&gt;
  &lt;property name="name"/&gt;
  &lt;property name="company"/&gt;
  &lt;property name="salary"/&gt;
  &lt;property name="department"/&gt;
&lt;/class&gt;
&lt;class name="com.nt.domain.Customer"  table="in_customers3" &gt;
  &lt;id name="id"/&gt;
  &lt;property name="name"/&gt;
  &lt;property name="company"/&gt;
  &lt;property name="address"/&gt;
&lt;/class&gt;
&lt;/hibernate-mapping&gt;</pre>
```

Person.java

```
package com.nt.domain;

public class Person {
    private int id;
    private String name;
    private String company;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getCompany() {
        return company;
    }
    public void setCompany(String company) {
        this.company = company;
    }
    @Override
    public String toString() {
        return "Person [id=" + id + ", name=" + name + ", company=" + company + "]";
    }
}
/*
 * create table in_persons3(id number(5) primary key,
 *                         name varchar2(20),
 *                         company varchar2(20)
 * )
 *
 * create table in_employee3(id number(5) primary key,
 *                          name varchar2(20),
 *                          company varchar2(20),
 *                          salary number(8),department number(8)
 * )
 *
 * create table in_customers3(id number(5) primary key,
 *                           name varchar2(20),
 *                           company varchar2(20),
 *                           address varchar2(20)
 * )
 */

```

Customer.java

```
package com.nt.domain;

public class Customer extends Person {
    private String address;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return super.toString()+"Customer [address=" + address + "]";
    }
}
```

Employee.java

```
package com.nt.domain;

public class Employee extends Person {
    private double salary;
    private int department;

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public int getDepartment() {
        return department;
    }

    public void setDepartment(int department) {
        this.department = department;
    }

    @Override
    public String toString() {
        return super.toString()+"Employee [salary=" + salary + ", department=" + department + "]";
    }
}
```

InhDAO.java

```
package com.nt.dao;

public interface InhDAO {
```

```
public void saveData();
public void displayData();

}
```

**InhDAOImpl.java**

```
package com.nt.dao;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Customer;
import com.nt.domain.Employee;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class InhDAOImpl implements InhDAO {

    @Override
    public void saveData() {
        //Get Session
        Session ses=HibernateUtil.getSession();
        //Save objs
        Person person=new Person();
        person.setId(1001); person.setName("raja"); person.setCompany("HCL");

        Employee emp=new Employee();
        emp.setId(1002); emp.setName("jani");emp.setCompany("InfoSys"); emp.setSalary(9000);
        emp.setDepartment(1001);

        Customer customer=new Customer();
        customer.setId(1003); customer.setName("rani"); customer.setCompany("WIPRO");
        customer.setAddress("hyd");
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(person);
            ses.save(emp);
            ses.save(customer);
            tx.commit();
            System.out.println(" Objects are Saved");
        } //try
        catch(Exception e){
            tx.rollback();
        }
        HibernateUtil.closeSession();
    }
}
```

```
//saveData()

@Override
public void displayData() {
    //Get Session
    Session ses=HibernateUtil.getSession();
    //Execute HQL Query
    Query query1=ses.createQuery("from Person");
    List<Person> list1=query1.list();
    for(Person per:list1){
        System.out.println(per);
    }//for

    System.out.println("-----");
    Query query2=ses.createQuery("from Employee");
    List<Employee> list2=query2.list();
    for(Employee emp:list2){
        System.out.println(emp);
    }//for

    System.out.println("-----");
    Query query3=ses.createQuery("from Customer");
    List<Customer> list3=query3.list();
    for(Customer cust:list3){
        System.out.println(cust);
    }//for
    //displayData

}//class
```

#### InhDAOFactory.java

```
package com.nt.dao;

public class InhDAOFactory {

    public static InhDAO getInstance(){
        return new InhDAOImpl();
    }
}
```

#### HibernateUtil.java

Same as previous

#### ClientApp.java

```
package com.nt.test;

import com.nt.dao.InhDAO;
```

```
import com.nt.dao.InhDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {
    public static void main(String[] args) {
        // Access DAO
        InhDAO dao=InhDAOFactory.getInstance();
        //call b.method
        //dao.saveData();

        //call b.method
        dao.displayData();

        //Close SessionFactory
        HibernateUtil.closeSessionFactory();

    }//main
}//class
```

#### Q. What is implicit polymorphism in inheritance mapping?

Ans:Implicit polymorphism.

While working with inheritance mapping strategy no 3, if we use multiple individual class tags to configure multiple classes of inheritance hierarchy then that comes under **implicit polymorphism** because while configuring classes we does not highlight inheritance that means we configure them as individuals concrete classes by also configuring super class inherited properties along with sub class properties but when we use super class names in HQL query to retrieve the records we not only get super class objects/records we also get subclass object/records this indicates super class reference variable is referring super class object and sub class objects. This is nothing but **implicit polymorphism**.

#### Explicit polymorphism:

hibernate gives only that class objects whose class name is specified in the HQL query i.e. it does not gives related sub class objects in any angle.

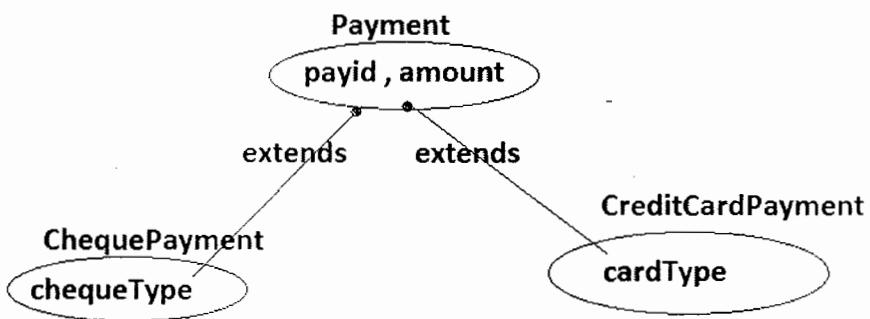
## Annotations driven inheritance mapping:

The JPA annotations for inheritance mapping are:

- **@Inheritance**- To specify inheritance mapping are
  - @Inheritance(strategy=InheritanceType.SINGLE\_TABLE)(Table per class hierarchy)
  - @Inheritance(strategy=InheritanceType.JOINED)(Table per sub class hierarchy)
  - @Inheritance(strategy=InheritanceType.TABLE\_PER\_CLASS)(Table per concrete class)
- **@DiscriminatorColumn**: To specify discriminator column in table per class hierarchy of Inheritance mapping.
- **@DiscriminatorValue**: To specify discriminator value for every class of inheritance.
- **@PrimaryKeyJoinColumn**: To specify the FK column.

### Note:

- **@Inheritance annotation** must be specified on the top root class belonging inheritance hierarchy of domain classes.
- If no **@Inheritance** is specified on the top of root class ,then table per class hierarchy (**SINGLE\_TABLE**) will be taken default inheritance mapping strategy.

Annotations based Table per class hierarchy Inheritance Mapping**Payment1(db table) (All the classes of inheritance hierarchy will use single DB table)**

payid(pk)	amount	chType	ccType	paytype (Discriminator col)
1	2000	Order	....	CHEQUE
2	5000	....	VISA	CREDIT

Payment.java (super class)

```

@Table("Payment1")
@Entity
@Inheritance(InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="paytype",discriminatorType=DiscriminatorType.STRING)
public class Payment{
    @Id
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int payid;
    private double amount;
    //write setXxx(-) and getXxx() methods
    ...
    ...
}
  
```

ChequePayment.java (Sub class1)

```

@Entity
@DiscriminatorValue("CHEQUE")
public class ChequePayment extends Payment{
    @Column(name="chType")
    private String chequeType;
    //setXxx(-) and getXxx()
    ...
    ...
}
  
```

CreditCardPayment.java (sub class2)

```

@Entity
@DiscriminatorValue("CREDIT")
public class CreditCardPayment
    extends Payment
    @Column(name="ccType")
    private String cardType;
    //setXxx(-) and getXxx() methods
    ....
    ....
}
  
```

 HBProj37(Anno-InhMapping1-TPCH)

```

    ▲ src
      ▲ com.nt.cfgs
        X hibernate.cfg.xml
      ▲ com.nt.dao
        J InhDAO.java
        J InhDAOFactory.java
        J InhDAOImpl.java
      ▲ com.nt.domain
        J ChequePayment.java
        J CreditCardPayment.java
        J Payment.java
      ▲ com.nt.test
        J ClientApp.java
      ▲ com.nt.utility
        J HibernateUtil.java

```

**hibernate.cfg.xml**

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Payment"/>
    <mapping class="com.nt.domain.ChequePayment"/>
    <mapping class="com.nt.domain.CreditCardPayment"/>
  </session-factory>
</hibernate-configuration>

```

**Payment.java**

```

package com.nt.domain;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

```

```
@Entity
@Table(name="Payment1")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="paytype",discriminatorType=DiscriminatorType.STRING)
public class Payment {
    @Id
    private int payld;
    private int amount;
    public int getPayld() {
        return payld;
    }
    public void setPayld(int payld) {
        this.payld = payld;
    }
    public int getAmount() {
        return amount;
    }
    public void setAmount(int amount) {
        this.amount = amount;
    }
    @Override
    public String toString() {
        return "[payld=" + payld + ", amount=" + amount + "]";
    }
}
```

#### ChequePayment.java

```
package com.nt.domain;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value="CHEQUE")
public class ChequePayment extends Payment {
    private String chType;

    public String getChType() {
        return chType;
    }
    public void setChType(String chType) {
        this.chType = chType;
    }
    @Override
    public String toString() {
        return "ChequePayment"+super.toString()+"[chType=" + chType + "]";
    }
}
```

**CreditCardPayment.java**

```
package com.nt.domain;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;

@Entity
@DiscriminatorValue(value="CREDIT")
public class CreditCardPayment extends Payment {
    private String ccType;

    public String getCcType() {
        return ccType;
    }

    public void setCcType(String ccType) {
        this.ccType = ccType;
    }

    @Override
    public String toString() {
        return "CreditCardPayment"+super.toString()+" [ccType=" + ccType + "]";
    }
}
```

**InhDAO.java**

```
package com.nt.dao;

public interface InhDAO {
    public void saveData();
    public void displayData();

}
```

**InhDAOImpl.java**

```
package com.nt.dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.ChequePayment;
import com.nt.domain.CreditCardPayment;
import com.nt.domain.Payment;
import com.nt.utility.HibernateUtil;
```

```
public class InhDAOImpl implements InhDAO {  
  
    @Override  
    public void saveData() {  
        //get Session  
        Session ses=HibernateUtil.getSession();  
        //prepare Objs  
        ChequePayment chPayment=new ChequePayment();  
        chPayment.setPayId(1001); chPayment.setAmount(30000);  
        chPayment.setChType("Self Paid");  
  
        CreditCardPayment ccPayment=new CreditCardPayment();  
        ccPayment.setPayId(1002); ccPayment.setAmount(40000);  
        ccPayment.setCcType("VISA");  
  
        //save objs  
        Transaction tx=null;  
        try{  
            tx=ses.beginTransaction();  
            ses.save(chPayment);  
            ses.save(ccPayment);  
            tx.commit();  
            System.out.println("Objs are saved");  
        } //try  
        catch(Exception e){  
            tx.rollback();  
        }  
        HibernateUtil.closeSession();  
    } //saveData()  
  
    @Override  
    public void displayData() {  
        //get Session  
        Session ses=HibernateUtil.getSession();  
        //display data  
        Query query1=ses.createQuery("from Payment");  
        List<Payment> list1=query1.list();  
        for(Payment payment:list1){  
            System.out.println(payment);  
        }  
        System.out.println("-----");  
        Query query2=ses.createQuery("from ChequePayment");  
        List<ChequePayment> list2=query2.list();  
        for(ChequePayment payment:list2){  
            System.out.println(payment);  
        }  
        System.out.println("-----");  
        Query query3=ses.createQuery("from CreditCardPayment");  
    }  
}
```

```
List<CreditCardPayment> list3=query3.list();
for(CreditCardPayment payment:list3){
    System.out.println(payment);
}

HibernateUtil.closeSession();
}

}
```

**InhDAOFactory.java**

```
package com.nt.dao;

public class InhDAOFactory {

    public static InhDAO getInstance(){
        return new InhDAOImpl();
    }

}
```

**HibernateUtil.java**

```
Same as previous
```

**ClientApp.java**

```
package com.nt.test;

import com.nt.dao.InhDAO;
import com.nt.dao.InhDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        InhDAO dao=InhDAOFactory.getInstance();
        // call method
        //dao.saveData();

        //display data
        dao.displayData();

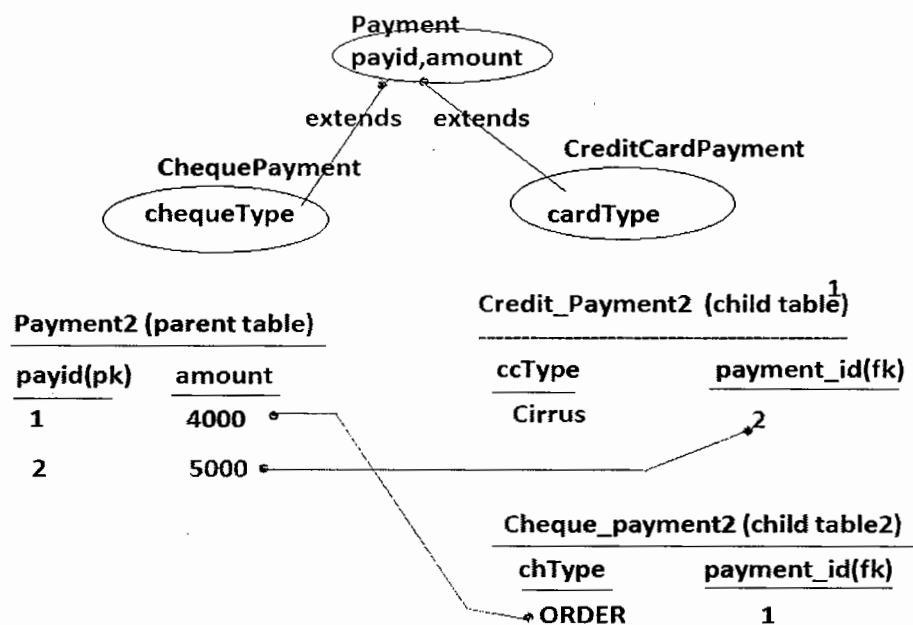
        //Close SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```

## Annotation driven inheritance mapping(Table Per Sub class)

- `@Inheritance(strategy=InheritanceType.JOINED)`
- `@PrimaryKeyJoinColumn` → To specify FK column

### Annotations based Table per sub class Inheritance Mapping

In this model every class of inheritance will have its own db table and all these db table will be in relationship.



### Payment.java (parent class)

```

@Entity
@Table(name="payment2")
@Inheritance(strategy=InheritanceType.JOINED)
public class Payment{
    @Id
    ...
    private int payid;
    private double amount;
    //setters and getters
    ...
}
  
```

### ChequePayment.java (sub class1)

```

@Entity
@Table(name="cheque_payment2")
@PrimaryKeyJoinColumn
(name="payment_id")
public class ChequePayment
    extends Payment{
    @Column(name="chType")
    private String chequeType;
    //setters and getters
    .....
}
  
```

### CreditCardPayment.java

```

@Entity
@Table(name="Credit_Payment2")
@PrimaryKeyJoinColumn(name="payment_id")
public class CreditCardPayment extends Payment{
    @Column(name="ccType")
    private String cardType;
    //setters and getters
    .....
}
  
```

//setters and getters

.....

}

HBProj38(Anno-InhMapping2-TPSC)

```

    ▲ src
      ▲ com.nt.cfgs
        X hibernate.cfg.xml
      ▲ com.nt.dao
        ▷ InhDAO.java
        ▷ InhDAOFactory.java
        ▷ InhDAOImpl.java
      ▲ com.nt.domain.
        ▷ ChequePayment.java
        ▷ CreditCardPayment.java
        ▷ Payment.java
      ▲ com.nt.test
        ▷ ClientApp.java
      ▲ com.nt.utility
        ▷ HibernateUtil.java
  
```

### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Payment"/>
    <mapping class="com.nt.domain.ChequePayment"/>
    <mapping class="com.nt.domain.CreditCardPayment"/>
  </session-factory>
</hibernate-configuration>
  
```

### Payment.java

```

package com.nt.domain;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
  
```

```
import javax.persistence.Table;

@Entity
@Table(name="Payment2")
@Inheritance(strategy=InheritanceType.JOINED)
public class Payment {
    @Id
    private int payId;
    private int amount;
    public int getPayId() {
        return payId;
    }
    public void setPayId(int payId) {
        this.payId = payId;
    }
    public int getAmount() {
        return amount;
    }
    public void setAmount(int amount) {
        this.amount = amount;
    }
    @Override
    public String toString() {
        return "[payId=" + payId + ", amount=" + amount + "]";
    }
}
```

#### ChequePayment.java

```
package com.nt.domain;

import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

@Entity
@Table(name="Cheque_payment2")
@PrimaryKeyJoinColumn(name="payment_id")
public class ChequePayment extends Payment {
    private String chType;

    public String getChType() {
        return chType;
    }

    public void setChType(String chType) {
        this.chType = chType;
    }
}
```

```
@Override  
public String toString() {  
return "ChequePayment"+super.toString()+"[chType=" + chType + "]";  
}  
}
```

#### CreditCardPayment.java

```
package com.nt.domain;  
  
import javax.persistence.DiscriminatorValue;  
import javax.persistence.Entity;  
import javax.persistence.PrimaryKeyJoinColumn;  
import javax.persistence.Table;  
  
@Entity  
@Table(name="Credit_payment2")  
@PrimaryKeyJoinColumn(name="payment_id")  
public class CreditCardPayment extends Payment {  
private String ccType;  
  
public String getCcType() {  
return ccType;  
}  
  
public void setCcType(String ccType) {  
this.ccType = ccType;  
}  
  
@Override  
public String toString() {  
return "CreditCardPayment"+super.toString()+" [ccType=" + ccType + "]";  
}  
}
```

#### InhDAO.java

```
package com.nt.dao;  
  
public interface InhDAO {  
public void saveData();  
public void displayData();  
}  
}
```

#### InhDAOImpl.java

```
package com.nt.dao;  
  
import java.util.List;
```

```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.ChequePayment;
import com.nt.domain.CreditCardPayment;
import com.nt.domain.Payment;
import com.nt.utility.HibernateUtil;

public class InhDAOImpl implements InhDAO {

    @Override
    public void saveData() {
        //get Session
        Session ses=HibernateUtil.getSession();
        //prepare Objs
        ChequePayment chPayment=new ChequePayment();
        chPayment.setPayId(1001); chPayment.setAmount(30000);
        chPayment.setChType("Self Paid");

        CreditCardPayment ccPayment=new CreditCardPayment();
        ccPayment.setPayId(1002); ccPayment.setAmount(40000);
        ccPayment.setCcType("VISA");

        //save objs
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(chPayment);
            ses.save(ccPayment);
            tx.commit();
            System.out.println("Objs are saved");
        } //try
        catch(Exception e){
            tx.rollback();
        }
        HibernateUtil.closeSession();
    } //saveData()

    @Override
    public void displayData() {
        //get Session
        Session ses=HibernateUtil.getSession();
        //display data
        Query query1=ses.createQuery("from Payment");
        List<Payment> list1=query1.list();
        for(Payment payment:list1){
            System.out.println(payment);
        }
    }
}
```

```

}

System.out.println("-----");
Query query2=ses.createQuery("from ChequePayment");
List<ChequePayment> list2=query2.list();
for(ChequePayment payment:list2){
    System.out.println(payment);
}
System.out.println("-----");
Query query3=ses.createQuery("from CreditCardPayment");
List<CreditCardPayment> list3=query3.list();
for(CreditCardPayment payment:list3){
    System.out.println(payment);
}

HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
}

```

**InhDAOFactory.java**

```

package com.nt.dao;

public class InhDAOFactory {

    public static InhDAO getInstance(){
        return new InhDAOImpl();
    }
}

```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.InhDAO;
import com.nt.dao.InhDAOFactory;
import com.nt.utility.HibernateUtil;
public class ClientApp {
    public static void main(String[] args) {
        // get DAO
        InhDAO dao=InhDAOFactory.getInstance();
        // call method
        //dao.saveData();

        //display data
        dao.displayData();
        //Close SessionFactory
        HibernateUtil.closeSession();
    }
}

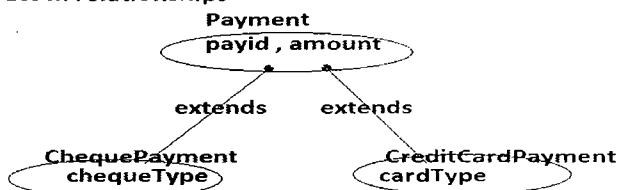
```

```
HibernateUtil.closeSessionFactory();
}//main
}//class
```

### Annotation driven inheritance mapping (table per Concrete class)

#### Annotations based Table per concrete class Inheritance Mapping

Here Every class inheritance hierarchy will have its own db table ,but those db table will not participate in relationships



**Payment3 (table1)**

payid(pk)	amount
2	5000

~~Table1 is optional~~

**Credit\_payment3(table2)**

payid	amont	ccType
2	5000	VISA

**Cheque\_payment(table3)**

payid	amount	chType
1	4000	ORDER

#### Payment.java(parent class)

```
@Entity
@Table(name="payment3")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Payment3{
    ....
    ....
}
```

#### CreditCardPayment.java (sub class1)

```
@Table(name="Credit_Payment3")
@Entity
public class CreditCardPayment extends Payment{
    ....
    ...
}
```

#### ChequePayment.java (sub class2)

```
@Table(name="Cheque_payment3")
@Entity
public class ChequePayment
    extends Payment{
    ...
    ...
}
```

### HBProj39(Anno-InhMapping3-TPCC)

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
  - com.nt.dao
    - InhDAO.java
    - InhDAOFactory.java
    - InhDAOImpl.java
  - com.nt.domain
    - ChequePayment.java
    - CreditCardPayment.java
    - Payment.java
  - com.nt.test
    - ClientApp.java
  - com.nt.utility
    - HibernateUtil.java

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Payment"/>
    <mapping class="com.nt.domain.ChequePayment"/>
    <mapping class="com.nt.domain.CreditCardPayment"/>

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

**Payment.java**

```
package com.nt.domain;

import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;

@Entity
@Table(name="Payment3")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public class Payment {
  @Id
  private int payId;
  private int amount;
  public int getPayId() {
    return payId;
  }
  public void setPayId(int payId) {
    this.payId = payId;
  }
  public int getAmount() {
    return amount;
  }
}
```

```
}
```

```
public void setAmount(int amount) {
```

```
this.amount = amount;
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
return "[payId=" + payId + ", amount=" + amount + "]";
```

```
}
```

```
}
```

### ChequePayment.java

```
package com.nt.domain;
```

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

```
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name="cheque_payment3")
```

```
public class ChequePayment extends Payment {
```

```
private String chType;
```

```
public String getChType() {
```

```
return chType;
```

```
}
```

```
public void setChType(String chType) {
```

```
this.chType = chType;
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
return "ChequePayment"+super.toString()+"[chType=" + chType + "]";
```

```
}
```

```
}
```

### CreditCardPayment.java

```
package com.nt.domain;
```

```
import javax.persistence.DiscriminatorValue;
```

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

```
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name="credit_payment3")
```

```
public class CreditCardPayment extends Payment {
```

```
private String ccType;
```

```
public String getCcType() {
    return ccType;
}

public void setCcType(String ccType) {
    this.ccType = ccType;
}

@Override
public String toString() {
    return "CreditCardPayment"+super.toString()+" [ccType=" + ccType + "]";
}
```

**InhDAO.java**

```
package com.nt.dao;

public interface InhDAO {
    public void saveData();
    public void displayData();
}
```

**InhDAOImpl.java**

```
package com.nt.dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.ChequePayment;
import com.nt.domain.CreditCardPayment;
import com.nt.domain.Payment;
import com.nt.utility.HibernateUtil;

public class InhDAOImpl implements InhDAO {

    @Override
    public void saveData() {
        //get Session
        Session ses=HibernateUtil.getSession();
        //prepare Objs
        ChequePayment chPayment=new ChequePayment();
        chPayment.setPayId(1001); chPayment.setAmount(30000);
        chPayment.setChType("Self Paid");

        CreditCardPayment ccPayment=new CreditCardPayment();
```

```
ccPayment.setPayId(1002); ccPayment.setAmount(40000);
ccPayment.setCcType("VISA");

//save objs
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(chPayment);
ses.save(ccPayment);
tx.commit();
System.out.println("Objs are saved");
}//try
catch(Exception e){
tx.rollback();
}
HibernateUtil.closeSession();
}//saveData()

@Override
public void displayData() {
//get Session
Session ses=HibernateUtil.getSession();
//display data
Query query1=ses.createQuery("from Payment");
List<Payment> list1=query1.list();
for(Payment payment:list1){
System.out.println(payment);
}
System.out.println("-----");
Query query2=ses.createQuery("from ChequePayment");
List<ChequePayment> list2=query2.list();
for(ChequePayment payment:list2){
System.out.println(payment);
}
System.out.println("-----");
Query query3=ses.createQuery("from CreditCardPayment");
List<CreditCardPayment> list3=query3.list();
for(CreditCardPayment payment:list3){
System.out.println(payment);
}
}
```

**InhDAOFactory.java**

```
package com.nt.dao;
```

```
public class InhDAOFactory {
```

```
public static InhDAO getInstance(){
    return new InhDAOImpl();
}
```

### HibernateUtil.java

Same as previous

### ClientApp.java

```
package com.nt.test;

import com.nt.dao.InhDAO;
import com.nt.dao.InhDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        InhDAO dao=InhDAOFactory.getInstance();
        // call method
        dao.saveData();
        //display data
        //dao.displayData();

        //Close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

## Working with Large Objects

There are two types of large objects (files).

### **1.BLOB(Binary Large Objects)**

\* audio files, video files, audio video files and etc..

### **2.CLOB(Character large objects)**

- Text files, rich text files, doc files and etc...

→ While developing matrimony application, profile management application, job portals applications we need to deal with large objects.

→ In hibernate ,there is no direct support to work with large objects. so we need to use jdbc and I/O streams together with hibernate code.

### To insert BLOB value:

BLOB file (image file) → byte[] → java.sql.Blob object → set to domain class object → ses.save(-) → insert into database table column.

**To insert CLOB value:**

CLOB file (text file) → char[] / (String) → java.sql.Clob object → set to domain class object → ses.save() → insert into database table column.

**To retrieve BLOB value:**

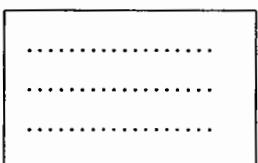
Database software → ses.load(-,-) → domain class object → java.sql.Blob object → create Blob file using streams.

**To retrieve CLOB value:**

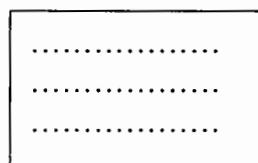
Database software → ses.load(-,-) → domain class object → java.sql.Clob object → create CLOB file using streams.

→ Buffer is a temporary memory that holds data for temporary period .While transferring data from one file to another file if buffer support is taken we can reduce no of read operations on source file and no of write operations on destination file.

Source file (10kb)

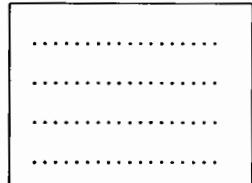


Destination file



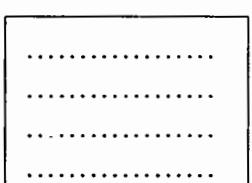
- Here no buffer support, So it performs  $10 \times 1024$  times of read operations on source file and  $10 \times 1024$  times of write operations on destination file.

Source File



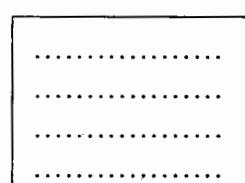
(10kb)

buffer



(byte[]) (4kb)

Destination file



(10kb)

- Here buffer support is taken(4kb size) ,due to this application just performs 3 read operations on source file and 3 write operations on destination file, so we can complete 10 kb data transfer from source file to destination file in less no.of round trips.

### HBProj40(LOBs)

- ◀ src
  - ◀ com.nt.cfgs
    - ☒ Employee.hbm.xml
    - ☒ hibernate.cfg.xml
  - ◀ com.nt.model
    - ▷ EmpBean.java
  - ◀ com.nt.test
    - ▷ InsertClient.java
    - ▷ RetrieveClient.java
  - ◀ com.nt.utility
    - ▷ FileUtils.java
    - ▷ HibernateUtil.java

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<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="hibernate.connection.username">scott</property>
<property name="hibernate.connection.password">tiger</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
 "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
<class name="com.nt.model.EmpBean" table="EmployeeProfile">
<id name="eid" column="emp_id"/>
<property name="ename" column="emp_name"/>
<property name="eresume" column="emp_resume" type="clob"/>
<property name="ephoto" column="emp_photo" type="blob"/>
</class>
</hibernate-mapping>

<!--
create table EmployeeProfile
(
emp_id number(5) primary key,
emp_name varchar2(20),
emp_resume clob,
emp_photo blob
);
-->
```

**EmpBean.java**

```
package com.nt.model;

import java.sql.Blob;
import java.sql.Clob;
```

```
public class EmpBean
{
private int eid;
private String ename;
private Clob eresume;
private Blob ephoto;

public int getEid() {
    return eid;
}

public void setEid(int eid) {
    this.eid = eid;
}

public String getEname() {
    return ename;
}

public void setEname(String ename) {
    this.ename = ename;
}

public Clob getEresume() {
    return eresume;
}

public void setEresume(Clob eresume) {
    this.eresume = eresume;
}

public Blob getEphoto() {
    return ephoto;
}

public void setEphoto(Blob ephoto) {
    this.ephoto = ephoto;
}

}
```

### FileUtility.java

```
package com.nt.utility;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.Reader;
import java.sql.Blob;
import java.sql.Clob;
```

```
public class FileUtility
{
    public static String readFileAsString(String filePath) throws Exception
    {
        String fileData="";
        BufferedReader reader = new BufferedReader(new FileReader(filePath));
        char[] buf = new char[1024];
        int numRead=0;
        while((numRead=reader.read(buf)) != -1){
            String readData = String.valueOf(buf, 0, numRead);
            fileData+=readData;

        }
        reader.close();
        return fileData.toString();
    }//readFileAsString

    //Returns the contents of the file in a byte array.
    public static byte[] getBytesArrayFromFile(String fpath)
    {
        try{
            File file=new File(fpath);
            InputStream is = new FileInputStream(file);
            long length = file.length();

            // Create the byte array to hold the data
            byte[] bytes = new byte[(int)length];

            // Read in the bytes
            int offset = 0;
            int numRead = 0;
            while (offset < bytes.length && (numRead=is.read(bytes, offset, bytes.length-offset)) >= 0) {
                offset += numRead;
            }

            // Ensure all the bytes have been read in
            if (offset < bytes.length) {
                throw new IOException("Could not completely read file "+file.getName());
            }

            // Close the input stream and return bytes
            is.close();
            return bytes;
        }catch(Exception e){return null;}
    }
}
```

```
}
```

```
public static String createFileFromClob(Clob clb, String fpath)
{
try
{
Reader reader=clb.getCharacterStream();
File data = new File(fpath);
FileWriter writer = new FileWriter(data);
char[] buffer = new char[1024];
while (reader.read(buffer) > 0) {
writer.write(buffer);
}
writer.close();
}catch(Exception e) {}
return "Data Stored Successfully in "+fpath;
}
```

```
public static String createFileFromBlob(Blob blb, String fpath)
{
try
{
InputStream in=blb.getBinaryStream();
FileOutputStream fos=new FileOutputStream(fpath);
int bytesRead=0;
byte buffer[]=new byte[4096];

while((bytesRead=in.read(buffer))!=-1)
{
fos.write(buffer,0,bytesRead);
}

fos.close();
}catch(Exception e) {}
return "photo stored Successfully in "+fpath;
}
}
```

### HibernateUtil.java

Same as previous

### InsertClient.java

```
package com.nt.test;

import java.sql.Blob;
import java.sql.Clob;

import org.hibernate.Hibernate;
```

```
import org.hibernate.LobHelper;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.model.EmpBean;
import com.nt.utility.FileUtility;
import com.nt.utility.HibernateUtil;

public class InsertClient
{
    public static void main(String s[])throws Exception
    {
        // Get Session obj
        Session ses=HibernateUtil.getSession();

        // convert to Text file in to a Stirng
        String fileContent=FileUtility.readFileAsString("c:\\store\\resume.txt");
        //Covert String data in to java.sql.Clob obj
        LobHelper helper=ses.getLobHelper();
        Clob resume=helper.createClob(fileContent);

        //convert image file data inot byte[]
        byte fileByte[]={};
        fileByte=FileUtility.getBytesArrayFromFile("c:/store/gavin.png");
        //convert byte[] into java.sql.Blob obj
        Blob photo=helper.createBlob(fileByte);

        EmpBean eb=new EmpBean();
        eb.setEid(1001);
        eb.setEname("raja");
        eb.setEresume(resume);
        eb.setEphoto(photo);

        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(eb);
            tx.commit();
            System.out.println("Successfully Inserted");
        } //try
        catch(Exception e){
            tx.rollback();
        }
    }
    HibernateUtil.closeSession();
    HibernateUtil.closeSessionFactory();
}
```

**RetrieveClient.java**

```

package com.nt.test;
import java.sql.Blob;
import java.sql.Clob;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import com.nt.model.EmpBean;
import com.nt.utility.FileUtility;
import com.nt.utility.HibernateUtil;
public class RetrieveClient
{
    public static void main(String s[])throws Exception
    {
        //Get session
        Session ses=HibernateUtil.getSession();
        //Load obj
        EmpBean eb=(EmpBean)ses.load(EmpBean.class,1001);
        //get Blob,Clob objects
        Clob resume=eb.getEresume();
        Blob photo=eb.getEphoto();
        //retrieve files
        String res1=FileUtility.createFileFromBlob(resume, "c:\\store2\\newResume.txt");

        String res2=FileUtility.createFileFromBlob(photo, "c:/store2/newpic.jpg");

        System.out.println(res1);
        System.out.println(res2);

        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}

```

**In hibernate 3.x we use:**

**Blob blob =Hibernate.createBlob(byte); or byte[] ----->deprecated.**

**Clob clob= Hibernate.createClob(file) ----->deprecated.**

**In hibernate 4.x (the alternates)**

**LobHelper helper = new LobHelper();**

**Blob blob = helper. createBlob(bytes); or(byte[])**

**Clob clob = helper. createClob(file)**

->Before executing this application make sure that c:\\..... foldername is available.

**Note:**

Saving large object values directly in database columns is good practices in standalone application but bad practices in web application but bad practices in web application, in webapplication it is recommended to use file uploading concepts to save the LOB files in server machine file system and write there address bars to database table columns as string values.

## Association mapping

### (Relationships in hibernate)

→ Storing multiple Entities (items) data in a single table gives the following two problems. They are

- Data redundancy:** For the sake of one entity the other entity value will be repeated/duplicated.
- Maintenance problem:** Managing huge amounts of data always complex process and we can not use one entity data without accessing another entity data.

eg:

- Storing students, faculties info in a single DATABASE table.
- Storing users, phonenumbers info in a single DATABASE table.
- Storing patients, doctors information in a single DATABASE table.

### Data redundancy problem:

#### Users phonenumbers info(Database table)

userid	firstname	address	phone	numbertype	provider
1	raja	hyd	999999	office	airtel
1	raja	hyd	888888	residence	vodafone
1	raja	hyd	777777	personal	uninor
2	ravi	delhi	666666	office	airtel
2	ravi	delhi	666333	personal	uninor
2	ravi	delhi	660077	residence	reliance

→ Here user details are duplicated, so it raises **data redundancy problem**.

#### Solution1:

Split data into multiple tables (place user details in separate table and place phonenumbers details in separate table)

#### (User table)

userid	firstname	address
1	raja	hyd
2	ravi	hyd

#### (phone number)

phone	numbertype	provider
9999	office	airtel
8888	res	airtel
-----	-----	-----
7777	personal	-----

Solves data redundancy, maintenance problem because of data division ,but raises navigation problem i.e. we can't navigate to phone\_number details from user details and vice-versa.

#### Solution2:

Keep the Database tables in relationship/Association. Here use one-to-many association.

#### user table(parent table)

Same as solution1

phone number(child table)

<u>phone</u>	<u>numbertype</u>	<u>provider</u>	<u>unid(fk with userid)</u>
9999	-----	-----	1
8888	-----	-----	1
7777	-----	-----	1
6600	-----	-----	2
66333	-----	-----	2

The above db tables designing solves the basic problem and also solves solution1 limitations.

**Associations in Hibernate:**

When DATABASE tables are taken in relationship/association, taking single domain class having huge number of properties representing multiple entities/db tables data gives object level **data redundancy, maintenance problem**. In order to overcome this problem we need to design domain classes having relationship/association that are required to map with DATABASE tables. For this we need to add special properties in domain classes of type **collection**. To make each object of domain class to hold one or more objects of **other associated domain class** and configuring these domain classes in mapping file by specifying their relationship/association is called **association mapping**. We highlight the properties that are required build the relationship, We also specify type and mode of relationship in mapping file.

**Hibernate supports 4 types relationship:**

1. **one-to-one**( eg: person<-----> license)
2. **one-to-many**( eg: user<-----> phone numbers)
3. **many-to-many**(eg:patients<-----> doctors)
4. **many-to-one**(eg:Employees<-----> department)

**Hibernate supports relationships/association in two modes:**

1. **Uni-directional:** we can access child objects from parent objects or parent objects from child objects.
2. **Bi-directional:** Here **parent to child and child to parent** navigation possible

**For association mapping we can use the following tags/annotations**

- <one-to-one> / @OneToOne
- <one-to-many> / @OneToMany
- <many-to-many> / @ManyToMany
- <many-to-one> / @ManyToOne

**one -to-many uni-directional association(parent to child)**

- Here each parent object holds one or more child objects(association class objects)
- Here parent class contains collection type property to hold one or more child object. We need to configure these collection properties by using **<set>**, **<list>**, **<map>** tags(**collection mapping**).
- The object that holds other object as associated object is called **parent object** and the object that is acting as associated object to other object is called **child object**.
- In **one to many, many to many** association we take collection type property to **build the association**.

**Example:**

one user holds multiple phone numbers (one-to-many)

**User table(parent)**

**phone number(child table)**

<u>user_id(pk)</u>	<u>first_name</u>	<u>phone(pk)</u>	<u>number_type</u>	<u>unid(fk)</u>
101	raja	9999	office	101
102	ravi	8888	residence	101
		7777	office	102

**Domain classes****User.java(parent class)**

```
public class User{
private int userid;
private String firstname;
private Set<PhoneNumber> phones;
//setter and getter
.....
.....
}
```

**PhoneNumber.java(child class)**

```
public class PhoneNumber{
private long phone;
private String numberType;
// setter and getter
.....
.....
}
```

**One -to-many Uni-Directional Association ( Parent to child)**

- => Here every obj of class parent class must be ready to hold 1 or more child class objs.
- => The relationship User and PhoneNumber is one to many ,Every user can have 1 or more PhoneNumbers.
- => To build this relationship/Association
  - Parent class should collection type property to hold 1 or more child objs
  - The collection property should be cfg in mapping file using collection tags like <set>/<list>/<map> and etc... tags and also specify relationship through <one-to-many> tag and FK col.

Tables design:

**User.java (parent class)**

```
public class User{
private int userId;
private String first Name;
private Set<PhoneNumber> phones;
// setters and getters
.....
.....
}
```

**PhoneNumber.java (Child class)**

```
public class PhoneNumber{
private long phone;
private String numberType;
// setters & getters
.....
.....
}
```

**user.hbm.xml**

```
<hibernate-mapping>
<class name="pkg.User" table="user_table">
<id name="userId" column="user_id">
<generator class="increment">
</id>
<property name="firstName" column="first_name"/>
<set name="phones" cascade="all">
<key column="unid"/>
<one-to-many class="pkg.PhoneNumber"/>
</set>
</class>
```

**PhoneNumber.hbm.xml**

```
<hibernate-mapping>
<class name="pkg.PhoneNumber" table="Phone_Numbers">
<id name="phone"/>
<property name="numberType" column="number_type"/>
</class>
</hibernate-mapping>
```

 HBProj41(One-to-Many Uni-set Collection)

```

    ▲ src
      ▲ com.nt.cfgs
        X hibernate.cfg.xml
        X phoneNumber.hbm.xml
        X user.hbm.xml
      ▲ com.nt.dao
        ▶ OToMDAO.java
        ▶ OToMDAOFactory.java
        ▶ OToMDAOImpl.java
      ▲ com.nt.domain
        ▶ PhoneNumber.java
        ▶ User.java
      ▲ com.nt.test
        ▶ ClientApp.java
      ▲ com.nt.utility
        ▶ HibernateUtil.java

```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/user.hbm.xml"/>
    <mapping resource="com/nt/cfgs/phoneNumber.hbm.xml"/>
  </session-factory>
</hibernate-configuration>

```

[user.hbm.xml](#)

```

<!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.domain.User" table="user_table" >
    <id name="userId" column="user_id"/>
    <property name="firstName" column="first_name"/>
    <set name="phones" cascade="all" lazy="true">
      <key column="unid"/>
      <one-to-many class="com.nt.domain.PhoneNumber"/>
    </set>
  </class>
</hibernate-mapping>

```

phoneNumber.hbm.xml

```
<!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.domain.PhoneNumber" table="Phone_numbers">
<id name="phone" />
<property name="numberType" column="number_type"/>
</class>
</hibernate-mapping>
```

PhoneNumber.java

```
package com.nt.domain;

public class PhoneNumber{
    private long phone;
    private String numberType;
    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }
    //toString
    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
    }
}
```

User.java

```
package com.nt.domain;
import java.util.Set;
public class User{
    private int userId;
    private String firstName;
```

```

private Set<PhoneNumber>phones;
public User() {
System.out.println("User:0-param constrctor");
}
public int getUserId() {
return userId;
}
public void setUserId(int userId) {
this.userId = userId;
}
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}
public Set<PhoneNumber> getPhones() {
return phones;
}
public void setPhones(Set<PhoneNumber> phones) {
this.phones = phones;
}
//toString
@Override
public String toString() {
return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}
/*
=====parent table=====
create table user_table
(user_id number(5) primary key,
first_name varchar2(20)
);
=====child table=====
create table Phone_numbers
( phone number(10) primary key,
number_type varchar2(20),
uid number(5) references user_table(user_id)
);
*/

```

OToMDAO.java

```

package com.nt.dao;

public interface OToMDAO {
public void saveData();

```

```
public void loadData();
public void addNewPhoneNumberToExistingUser();
public void deleteUserWithPhoneNumbers();
public void deleteOnePhoneNumberOfAUser();
public void deleteAllPhoneNumbersOfAUser();

}
```

**OToMDAOImpl.java**

```
package com.nt.dao;

import java.util.HashSet;
import java.util.Set;

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

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {
    private static final String HQL_GET_ALL_USERS_QUERY="from User";

    @Override
    public void saveData() {
        Session ses=HibernateUtil.getSession();
        //parent obj
        User user=new User();
        user.setUserId(1001);
        user.setFirstName("raja2");
        //child obj1
        PhoneNumber ph1=new PhoneNumber();
        ph1.setPhone(999999);
        ph1.setNumberType("office1");

        //child obj1
        PhoneNumber ph2=new PhoneNumber();
        ph2.setPhone(888888);
        ph2.setNumberType("residence");
        //set multiple child objs to parent obj (1 to many)
        Set <PhoneNumber>childs=new HashSet<PhoneNumber>();
        childs.add(ph1); childs.add(ph2);
        user.setPhones(childs);

        //parent obj
        User user1=new User();
        user1.setUserId(1002);
```

```
user1.setFirstName("ravi1");
//child obj1
PhoneNumber ph3=new PhoneNumber();
ph3.setPhone(6666666);
ph3.setNumberType("office");

//child obj1
PhoneNumber ph4=new PhoneNumber();
ph4.setPhone(77777777);
ph4.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
Set <PhoneNumber>childs1=new HashSet<PhoneNumber>();
childs1.add(ph3); childs1.add(ph4);
user1.setPhones(childs1);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(user);
    ses.save(user1);
    tx.commit();
    System.out.println("Objs are saved");
} //try
catch(Exception e){
    tx.rollback();
}
//close Session
HibernateUtil.closeSession();
}//save Data

@Override
public void loadData() {
Session ses=null;
Query query=null;
List<User>list=null;
Set<PhoneNumber> childs=null;
//Get Session
ses=HibernateUtil.getSession();
//prepare HQL Query
query=ses.createQuery(HQL_GET_ALL_USERS_QUERY);
list=query.list();
for(User user:list){
    System.out.println("parent---->" + user);
    //get all child objs of a parent
    childs=user.getPhones();
    for(PhoneNumber ph:childs){
        System.out.println("child---->" + ph);
    } //for
} //for
}
```

```
/*Session ses=HibernateUtil.getSession();
User user=(User)ses.get(User.class,1001);
Set<PhoneNumber> childs=user.getPhones();
childs.size();
for(PhoneNumber ph:childs){
    System.out.println(ph);
}
*/
}//loadData()

//Adding new Child to existing childs of a parent
@Override
public void addNewPhoneNumberToExistingUser() {
//get Session
Session ses=HibernateUtil.getSession();
//Load User obj/parent obj (1001)
User user=(User)ses.get(User.class,1001);
// load existing childs of a User/parent
Set<PhoneNumber> childs=user.getPhones();
//create new Child (new PhoneNumber)
PhoneNumber ph=new PhoneNumber();
ph.setPhone(9998888); ph.setNumberType("personal");
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.add(ph);
    tx.commit();
    System.out.println("New child is added");
} //try
catch(Exception e){
    tx.rollback();
}
}

@Override
public void deleteUserWithPhoneNumbers() {
//get Session
Session ses=HibernateUtil.getSession();
//load User obj (parent obj)
User user=(User)ses.load(User.class,1002);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.delete(user);
    tx.commit();
} //try
}
```

```
catch(Exception e){  
    tx.rollback();  
}  
}//method  
  
//Deleting one child from collection of childs beloging to a parent  
@Override  
public void deleteOnePhoneNumberOfAUser() {  
    // Get Session  
    Session ses=HibernateUtil.getSession();  
    //Load 1 parent obj (User obj)  
    User user=(User)ses.get(User.class,1002);  
    // get all childs of a parent obj (PhoneNumber objs)  
    Set <PhoneNumber> childs=user.getPhones();  
    //Load PhoneNumber obj  
    PhoneNumber ph=(PhoneNumber)ses.get(PhoneNumber.class,(long)66666666);  
    Transaction tx=null;  
    try{  
        tx=ses.beginTransaction();  
        childs.remove(ph);  
        tx.commit();  
    } //try  
    catch(Exception e){  
        tx.rollback();  
    }  
} //method  
  
/* deleting all child objs of parent */  
@Override  
public void deleteAllPhoneNumbersOfAUser() {  
    // get Session  
    Session ses=HibernateUtil.getSession();  
    //Load parent obj  
    User user=(User)ses.get(User.class,1001);  
    //get all childs of a parent  
    Set <PhoneNumber> childs=user.getPhones();  
    Transaction tx=null;  
    try{  
        tx=ses.beginTransaction();  
        childs.clear();  
        tx.commit();  
    } //try  
    catch(Exception e){  
        tx.rollback();  
    }  
}  
}//class
```

**OToMDAOFactory.java**

```
package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}
```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```
package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();
        //call DAO methods
        //dao.saveData();
        //dao.loadData();
        //dao.addNewPhoneNumberToExistingUser();
        //dao.deleteUserWithPhoneNumbers();
        //dao.deleteOnePhoneNumberOfAUser();
        dao.deleteAllPhoneNumbersOfAUser();
        //close Session
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

- Cascading means propagating/reflecting non-select persistence operations done on the main objects to the associated objects. For this we need to use “**cascade**” attribute
- **cascade="all"** means any non-select persistence operations performed on the parent /main objects will be cascaded/reflected to the associated child objects.  
`<set name="phones" cascade="all">`  
.....  
`</set>`

**Select operations in one-to-many association:**

- In association mapping parent objects will be loaded normally and the associated child objects will be loaded only on demand basis, this is called **lazy loading in association**.
- We can enable or disable this lazy loading in one-to-many association by using **lazy=".....">** attribute of <list>, <map>, <set> and etc.. tags.
- This attribute allows three possible values and default value is **true**. They are:
  1. **lazy=true**
  2. **lazy = extra**
  3. **lazy= false**

**lazy = false:**

→ Performs **eager loading**, loads child objects/associated objects along with main/parent objects irrespective of whether child objects are used or not.

**lazy = true:**

→ Performs **lazy loading**, loads parent objects normally but loads the associated child lazily on demand basis. Here child objects for every small activity performed on the collection like finding size, checking object empty or not.

**lazy = extra:**

→ Performs **extra lazy loading** i.e. same as lazy=true but the collection property of each parent object will be initialized with child object only for the appropriate operation that is using child object data i.e. collection will not be initialized with child objects for small operations on collection like getting size etc... but will be initialized when child object's data is used.

**Q. How to enable lazy loading in hibernate?**

Ans:

- In one-to-many,many-to -many association we use **lazy=true or lazy = extra attribute in <set> , <map>, <list> tag and etc...**
- While working with many-to-one, one-to-one association we use **lazy="proxy" or lazy="no-proxy"** attribute.
- Use **Query.iterate()** method while working with HQL queries.
- Use **FetchType.LAZY** for "fetch" param association mapping with annotations.

**Note:** While working with collection properties, if you modify collection elements data in a transaction then modification will be **reflected to DATABASE table records**.

**Delete operations in one- to-many association**

→ Here if we take cascade ="all" or cascade="delete" then the delete persistence operation performed on the parent objects will be cascaded to associated child objects automatically. If we place other values in cascade attribute like **none, persist** and etc then only parent objects/main objects will be deleted.

**Deleting one child from collection of childs belonging to a parent**

```

public void deleteOnePhoneNumberOfAUser() {
    // Get Session
    Session ses=HibernateUtil.getSession();
    //Load 1 parent obj (User obj)
    User user=(User)ses.get(User.class,1002);

    // get all childs of a parent obj (PhoneNumber objs)
    Set <PhoneNumber> childs=user.getPhones();

    //Load PhoneNumber obj
    PhoneNumber ph=(PhoneNumber)ses.get(PhoneNumber.class,(long)6666666);
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        childs.remove(ph);
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }
}

```

→ When the above code is executed it does not delete child table records but it deletes links between child table records with associated parent table records **by emptying fk column in child table** and such records are called **orphan records**.

**User table:**

<u>user id</u>	<u>first name</u>
<u>1001</u>	raja

**Phone number**

<u>phone(pk)</u>	<u>number type</u>	<u>unid(fk)</u>
<u>66666</u>	office	----- ( <i>orphan record</i> )
8888	residence	1001

→ In hibernate framework for deleting even orphan records that are created while deleting Child record or records of parent records, we need to use **cascade="delete-orphan"** or **cascade="all-delete-orphan"** in **<set>**,**<list>**,**<map>** and etc tags.

eg:

```

<set name="phones" cascade="all-delete-orphan" lazy="extra">
.....
.....
</set>

```

**Q. Can you explain cascading strategy in hibernate with respect to one-to-many association?**

- The non-select persistence operations performed on parent object will be reflected/ propagated to associated child objects based on the cascading that is enabled. cascading is no way related to select persistence operations, for this use **lazy** attribute.
- Default value of cascade attribute is **none** i.e **no cascading takes place**.
  - **cascade = "persist"** cascades persist/save operations to the associated objects.
  - **cascade = "save-update"** cascades save,update operations to the associated objects.
  - **cascade = "merge"** cascades merge operations to the associated objects
  - **cascade = "delete"** cascades delete operations to the associated objects.
  - **cascade = "all"** cascades all non-select persistence operations to the associated objects.
  - **cascade = "delete-orphan"** same as cascade = "delete" but also deletes orphan records.
  - **cascade = "all-delete-orphan"** same as cascade="all" but also deletes orphan records. and etc...

**Delete all phone numbers of a user (all childs of a parent)**

```
public void deleteAllPhoneNumbersOfAUser() {
    // get Session
    Session ses=HibernateUtil.getSession();
    //Load parent obj
    User user=(User)ses.get(User.class,1001);
    //get all childs of a parent
    Set <PhoneNumber> childs=user.getPhones();
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        childs.clear();
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }
}
```

**Taking java.util.List as the Collection type**

- If you use **Set collection** in association mapping we can not identify which child belongs to which parent and we can't even preserve the **insertion order** because **java.util.Set** collection does not maintain the elements having indexes and does not preserve insertion order.
- To overcome this problem and to maintain numeric indexes for child objects of a parent object we need to go for **List collection** as the special property of parent class.

- We can configure this **java.util.List collection** using **<list>** tag and we can add an additional column to the table. To maintain list indexes we need to configure these list indexes column by using **<list-index>** tag.

(User table)

<b>user_id</b>	<b>firstname</b>
1001	raja

(Phone numbers)

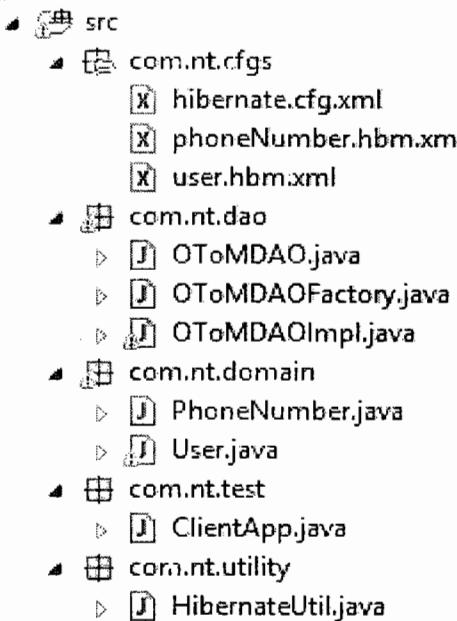
<b>Phone</b>	<b>numbertype</b>	<b>unid(fk)</b>	<b>lst_idx</b>
9999	office	1001	0
8888	personal	1001	1

In user.hbm.xml

```
<list name="phones" cascade="all-delete-orphan" lazy="extra">
  <key column="unid"/>
  <list-index column="lst_idx"/>
  <one-to-many class="com.nt.domain.PhoneNumber"/>
</list>
```

**Note:**If the collection type **java.util.List** then index column configuration is mandatory.For this we use **<list>** tag

## HBProj42(One-to-Many Uni-List Collection)

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
  <session-factory>
    <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">scott</property>
    <property name="connection.password">tiger</property>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/user.hbm.xml"/>
    <mapping resource="com/nt/cfgs/phoneNumber.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**phoneNumber.hbm.xml**

```
<!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.domain.PhoneNumber" table="Phone_numbers1">
    <id name="phone" />
    <property name="numberType" column="number_type"/>
  </class>
</hibernate-mapping>
```

**user.hbm.xml**

```
<!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.domain.User" table="user_table1" >
    <id name="userId" column="user_id"/>
    <property name="firstName" column="first_name"/>
    <list name="phones" cascade="all-delete-orphan" lazy="extra">
      <key column="unid"/>
      <list-index column="lst_idx"/>
      <one-to-many class="com.nt.domain.PhoneNumber"/>
    </list>
  </class>
</hibernate-mapping>
```

**PhoneNumber.java**

```
package com.nt.domain;

public class PhoneNumber{
  private long phone;
```

```
private String numberType;
public PhoneNumber() {
System.out.println("PhoneNumber:0-param constructor");
}
//setters and getters
public long getPhone() {
return phone;
}
public void setPhone(long phone) {
this.phone = phone;
}
public String getNumberType() {
return numberType;
}
public void setNumberType(String numberType) {
this.numberType = numberType;
}
//toString
@Override
public String toString() {
return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
}
}
```

### User.java

```
package com.nt.domain;

import java.util.List;
import java.util.Set;

public class User{
private int userId;
private String firstName;
private List<PhoneNumber>phones;
public User() {
System.out.println("User:0-param constrctor");
}
public int getUserId() {
return userId;
}
public void setUserId(int userId) {
this.userId = userId;
}
public String getFirstName() {
return firstName;
}
}
```

```
public void setFirstName(String firstName) {  
    this.firstName = firstName;  
}  
public List<PhoneNumber> getPhones() {  
    return phones;  
}  
public void setPhones(List<PhoneNumber> phones) {  
    this.phones = phones;  
}  
//toString  
@Override  
public String toString() {  
    return "User [userId=" + userId + ", firstName=" + firstName + "]";  
}  
}  
  
/*====parent table=====  
create table user_table1  
(user_id number(5) primary key,  
first_name varchar2(20)  
);  
=====child table=====  
create table Phone_numbers1  
( phone number(10) primary key,  
number_type varchar2(20),  
unid number(5) references user_table1(user_id),  
lst_indx number(5));  
  
);  
*/
```

**OToMDAO.java**

```
package com.nt.dao;  
  
public interface OToMDAO {  
    public void saveData();  
    public void deleteOnePhoneNumberOfAUser();  
}
```

**OToMDAOImpl.java**

```
package com.nt.dao;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import org.hibernate.Session;  
import org.hibernate.Transaction;
```

```
import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {
private static final String HQL_GET_ALL_USERS_QUERY="from User";

@Override
public void saveData() {
Session ses=HibernateUtil.getSession();
//parent obj
User user=new User();
user.setUserId(1001);
user.setFirstName("raja2");
//child obj1
PhoneNumber ph1=new PhoneNumber();
ph1.setPhone(999999);
ph1.setNumberType("office1");

//child obj2
PhoneNumber ph2=new PhoneNumber();
ph2.setPhone(888888);
ph2.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
List <PhoneNumber>childs=new ArrayList<PhoneNumber>();
childs.add(ph1); childs.add(ph2);
user.setPhones(childs);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(user);
tx.commit();
} //try
catch(Exception e){
tx.rollback();
}
}//saveData()

@Override
public void deleteOnePhoneNumberOfAUser() {
// Get Session
Session ses=HibernateUtil.getSession();
//Load Parent obj/User obj
User user=(User)ses.get(User.class,1001);
//get All childs of a User
List<PhoneNumber>childs=user.getPhones();
// Get specific child that u want to delete
PhoneNumber ph=(PhoneNumber)childs.get(0);
```

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.remove(ph);
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}

}
}//class
```

**OToMDAOFactory.java**

```
package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}
```

**HibernateUtil.java**

Same as above

**ClientApp.java**

```
package com.nt.test;
import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();

        //call DAO methods
        dao.saveData();
        dao.deleteOnePhoneNumberofAUser();

        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    } //main
}
```

**Important points:**

- While working with java.util.Set collection if you are deleting one child from collection of childs belonging to a parent, we must load that child object from DATABASE because we can't access elements from Set collection through indexes.
- While working with List collection in the above said situation ,there is no need of loading child object from database because we can get it directly from List collection using index.

**<bag> tag:**

When we configure java.util.List collection using <list> tag then configuration of index column is mandatory but we want to use List collection without **having restriction of configuring list index column** configuration because we don't want to store index of child in child DATABASE table but we want to take other benefit of list collection like accessing element through indexes for this we need to configure the java.util.List collection by using <bag> tag.

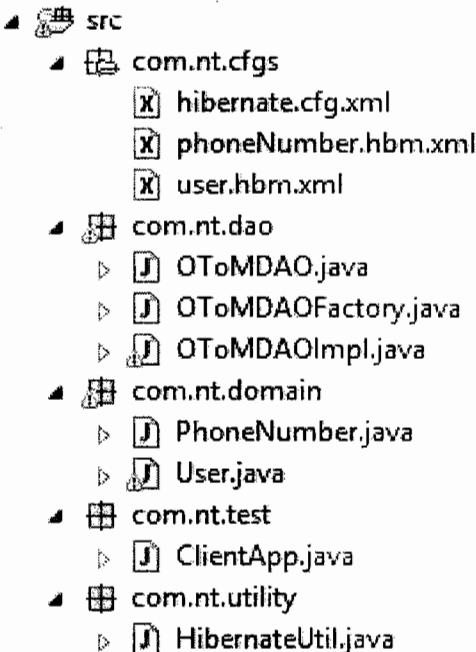
**Example:**

- a) Create tables without index column:

<i>User table2(parent table)</i>		<i>phone number2(child table)</i>		
<i>user id(pk)</i>	<i>firstname</i>	<i>phone(pk)</i>	<i>numbertype</i>	<i>unid(fk)</i>
101	raja	9999999	office	101
		8888888	residence	101

- b) Make sure that List collection property is placed in User.java  
 c) Configure java.util.List collection property in mapping file using<bag> tag in user.hbm.xml

```
<bag name="phones" cascade="all-delete-orphan" lazy="extra">
  <key column="unid"/>
  <one-to-many class="com.nt.domain.PhoneNumber"/>
</bag>
```

 HBProj43(One-to-Many Uni-List Collection with Bag)


**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/user.hbm.xml"/>
        <mapping resource="com/nt/cfgs/phoneNumber.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**phoneNumber.hbm.xml**

```
<!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.domain.PhoneNumber" table="Phone_numbers2">
        <id name="phone" />
        <property name="numberType" column="number_type"/>
    </class>
</hibernate-mapping>
```

**user.hbm.xml**

```
<!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.domain.User" table="user_table2" >
        <id name="userId" column="user_id"/>
        <property name="firstName" column="first_name"/>
        <bag name="phones" cascade="all-delete-orphan" lazy="extra">
            <key column="unid"/>
            <one-to-many class="com.nt.domain.PhoneNumber"/>
        </bag>
    </class>
</hibernate-mapping>
```

**PhoneNumber.java**

```
package com.nt.domain;

public class PhoneNumber{
    private long phone;
    private String numberType;
    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }
    //toString
    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
    }
}
```

**User.java**

```
package com.nt.domain;

import java.util.List;
import java.util.Set;

public class User{
    private int userId;
    private String firstName;
    private List<PhoneNumber> phones;
    public User() {
        System.out.println("User:0-param constructor");
    }
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
```

```

this.userId = userId;
}
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}
public List<PhoneNumber> getPhones() {
return phones;
}
public void setPhones(List<PhoneNumber> phones) {
this.phones = phones;
}
//toString
@Override
public String toString() {
return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}

/*====parent table=====
create table user_table2
(user_id number(5) primary key,
first_name varchar2(20)
);
=====child table=====
create table Phone_numbers2
( phone number(10) primary key,
number_type varchar2(20),
unid number(5) references user_table2(user_id)
);
*/

```

**OToMDAO.java**

```

package com.nt.dao;

public interface OToMDAO {
public void saveData();
public void deleteOnePhoneNumberOfAUser();

}

```

**OToMDAOImpl.java**

```

package com.nt.dao;

import java.util.ArrayList;
import java.util.List;

```

```
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {
private static final String HQL_GET_ALL_USERS_QUERY="from User";

@Override
public void saveData() {
Session ses=HibernateUtil.getSession();
//parent obj
User user=new User();
user.setUserId(1001);
user.setFirstName("raja2");
//child obj1
PhoneNumber ph1=new PhoneNumber();
ph1.setPhone(9999999);
ph1.setNumberType("office1");

//child obj2
PhoneNumber ph2=new PhoneNumber();
ph2.setPhone(8888888);
ph2.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
List <PhoneNumber>childs=new ArrayList<PhoneNumber>();
childs.add(ph1); childs.add(ph2);
user.setPhones(childs);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(user);
tx.commit();
} //try
catch(Exception e){
tx.rollback();
}
//close HB session
HibernateUtil.closeSession();
}//saveData()

@Override
public void deleteOnePhoneNumberOfAUser() {
// Get Session
Session ses=HibernateUtil.getSession();
```

```

//Load Parent obj/User obj
User user=(User)ses.get(User.class,1001);
//get All childs of a User
List<PhoneNumber>childs=user.getPhones();
// Get specific child that u want to delete
PhoneNumber ph=(PhoneNumber)childs.get(0);
Transaction tx=null;
try{
tx=ses.beginTransaction();
childs.remove(ph);
tx.commit();
}//try
catch(Exception e){
tx.rollback();
}
//close Session
HibernateUtil.closeSession();

}
}//class

```

**OToMDAOFactory.java**

```

package com.nt.dao;

public class OToMDAOFactory {
public static OToMDAO getInstance(){
return new OToMDAOImpl();

}

}

```

**HibernateUtil.java**

```

package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

public static void main(String[] args) {
// get DAO
OToMDAO dao=OToMDAOFactory.getInstance();

```

```
//call DAO methods
//dao.saveData();
dao.deleteOnePhoneNumberOfAUser();
//close Session
HibernateUtil.closeSessionFactory();
}//main
}//class
```

### Taking java.util.Map collection for building association:

- If we take java.util.List type collection using <list> tag the child objects of a parent object maintain indexes that are '0' basis index. To have control on indexes(**String or other type indexes**) we need to go for java.util.Map type collection. This Map collection element keys act as indexes and values acts as child objects.

#### Example:

<u>User table</u>		<u>phone numbers</u>			
<u>user id</u>	<u>firstname</u>	<u>phone(pk)</u>	<u>number type</u>	<u>unid(fk)</u>	<u>map indx</u>
101	raja	9999	office	101	phone1
		8888	residence	101	phone2

- In List collection indexes are numeric values so child table must have numeric index column. The keys of Map elements can hold any type of objects, So index column of child table can be of **any type**.
- Make sure that User class having java.util.Map type collection property

```
private <String,PhoneNumber>phones;
```

- Configure the property in mapping file using <map> tag.

#### In user.hbm.xml

```
<map name="phones" cascade="all-delete-orphan" lazy="extra">
  <key column="unid"/>
  <map-key column="map_indx" type="java.lang.String"/>
  <one-to-many class="com.nt.domain.PhoneNumber"/>
</map>
```

#### HBProj44(One-to-Many Uni-Map Collection)

```

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
    - phoneNumber.hbm.xml
    - user.hbm.xml
  - com.nt.dao
    - OToMDAO.java
    - OToMDAOFactory.java
    - OToMDAOImpl.java
  - com.nt.domain
    - PhoneNumber.java
    - User.java
  - com.nt.test
    - ClientApp.java
  - com.nt.utility
    - HibernateUtil.java

```

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/user.hbm.xml"/>
        <mapping resource="com/nt/cfgs/phoneNumber.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**phoneNumber.hbm.xml**

```
<!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.domain.PhoneNumber" table="Phone_numbers3">
        <id name="phone" />
        <property name="numberType" column="number_type"/>
    </class>
</hibernate-mapping>
```

**user.hbm.xml**

```
<!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.domain.User" table="user_table3" >
        <id name="userId" column="user_id"/>
        <property name="firstName" column="first_name"/>
        <map name="phones" cascade="all-delete-orphan" lazy="extra">
            <key column="unid"/>
            <map-key column="map_idx" type="java.lang.String"/>
            <one-to-many class="com.nt.domain.PhoneNumber"/>
        </map>
    </class>
</hibernate-mapping>
```

PhoneNumber.java

```
package com.nt.domain;

public class PhoneNumber{
    private long phone;
    private String numberType;
    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }
    //toString
    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
    }
}
```

User.java

```
package com.nt.domain;

import java.util.Map;

public class User{
    private int userId;
    private String firstName;
    private Map<String,PhoneNumber> phones;
    public User() {
        System.out.println("User:0-param constrctor");
    }
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
}
```

```

}

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public Map<String,PhoneNumber> getPhones() {
    return phones;
}

public void setPhones(Map<String,PhoneNumber> phones) {
    this.phones = phones;
}

//toString
@Override
public String toString() {
    return "User [userId=" + userId + ", firstName=" + firstName + "]";
}

}

/*====parent table=====
create table user_table3
(user_id number(5) primary key,
first_name varchar2(20)
);

=====child table=====
create table Phone_numbers3
( phone number(10) primary key,
number_type varchar2(20),
unid number(5) references user_table3(user_id),
map_indx varchar2(10));

);
*/

```

**OToMDAO.java**

```

package com.nt.dao;

public interface OToMDAO {
    public void saveData();
    public void deleteOnePhoneNumberOfAUser();

}

```

**OToMDAOImpl.java**

```

package com.nt.dao;

import java.util.LinkedHashMap;
import java.util.Map;

```

```
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {
private static final String HQL_GET_ALL_USERS_QUERY="from User";

@Override
public void saveData() {
Session ses=HibernateUtil.getSession();
//parent obj
User user=new User();
user.setUserId(1001);
user.setFirstName("raja2");
//child obj1
PhoneNumber ph1=new PhoneNumber();
ph1.setPhone(999999);
ph1.setNumberType("office1");

//child obj2
PhoneNumber ph2=new PhoneNumber();
ph2.setPhone(888888);
ph2.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
Map <String,PhoneNumber>childs=new LinkedHashMap<String,PhoneNumber>();
childs.put("phone1",ph1); childs.put("phone2",ph2);
user.setPhones(childs);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(user);
tx.commit();
} //try
catch(Exception e){
tx.rollback();
}
//close HB session
HibernateUtil.closeSession();
}//saveData()

@Override
public void deleteOnePhoneNumberOfAUser() {
// Get Session
Session ses=HibernateUtil.getSession();
```

```
//Load Parent obj/User obj
User user=(User)ses.get(User.class,1001);
//get All childs of a User
Map<String,PhoneNumber>childs=user.getPhones();

Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.remove("phone1");
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}

}
}//class
```

#### OToMDAOFactory.java

```
package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}
```

#### HibernateUtil.java

Same as previous

#### ClientApp.java

```
package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();

        //call DAO methods
        //dao.saveData();
        dao.deleteOnePhoneNumberOfAUser();

        //close Session,SessionFactory
    }
}
```

```

HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

**Note:**

The key of map collection become indexes column value of child table.

**Q. can you explain different type of collection property that are taken in hibernate?**

Java.util.List	Java.util.Set	Java.util.Map
<ol style="list-style-type: none"> <li>1. Allows duplicates</li> <li>2. Preserves insertion order.</li> <li>3. In hibernate we use &lt;list&gt; /&lt;bag&gt; tags to configure collection type property</li> <li>4. Index column configuration is optional with &lt;bag&gt; tag but mandatory with &lt;list&gt; tag</li> <li>5. To configure index column use &lt;list-index&gt; tag</li> <li>6. While deleting one-child from collection of childs, loading child from DATABASE is not required</li> </ol>	<ol style="list-style-type: none"> <li>1. Does not allow</li> <li>2. Does preserve insertion order.</li> <li>3. We use &lt;set&gt; tag</li> <li>4. Not applicable</li> <li>5. Not applicable</li> <li>6. Required</li> </ol>	<ol style="list-style-type: none"> <li>1. Allows duplicate values but does not allow duplicate keys.</li> <li>2. Does not</li> <li>3. &lt;map&gt; tag</li> <li>4. Mandatory</li> <li>5. &lt;map-key&gt; tag</li> <li>6. Not Required</li> </ol>

HBProj55-OneToMany-Uni-FKColWithPropertyinDomainClass

- src/main/java
  - com.nt.cfgs
    - hibernate.cfg.xml
    - phonenumbers.hbm.xml
    - user.hbm.xml
  - com.nt.dao
    - OneToManyDAO.java
    - OneToManyDAOFactory.java
    - OneToManyDAOImpl.java
  - com.nt.domain
    - PhoneNumber.java
    - User.java
  - com.nt.test
    - OneToManyTest.java
  - com.nt.utility
    - HibernateUtil.java
- src/test/java
- JRE System Library [JavaSE-1.8]
- Maven Dependencies
- src
- target
- DBTables.txt
- pom.xml

**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/user.hbm.xml"/>
        <mapping resource="com/nt/cfgs/phonenumbers.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**phonenumbers.hbm.xml:-**

```
<!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.domain.PhoneNumber" table="phone_numbers">
        <id name="phone"/>
        <property name="numberType" column="number_type"/>
        <property name="uid" column="unid" insert="false" update="false" />
    </class>
</hibernate-mapping>
```

**user.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!--Inheritance mapping cfgs -->
    <class name="com.nt.domain.User" table="user_table">
        <id name="userId" column="user_id"/>
        <property name="firstName" column="first_name"/>
        <set name="phones" lazy="true" cascade="all-delete-orphan">
            <key column="unid"/>
            <one-to-many class="com.nt.domain.PhoneNumber"/>
        </set>
    </class>
</hibernate-mapping>
```

**OneToManyDAO.java:-**

```
package com.nt.dao;

public interface OneToManyDAO {
    public void saveData();
    public void loadData();
    public void deleteOnePhoneNumberOfAUser();
    public void deleteAllPhoneNumbersOfAUser();
    public void addNewPhoneNumberToExistingUser();
    public void movePhoneNumberFromOneUserToAnotherUser();
}
```

**OneToManyDAOFactory.java:-**

```
package com.nt.dao;

public class OneToManyDAOFactory {

    public static OneToManyDAO getInstance(){
        return new OneToManyDAOImpl();
    }
}
```

**OneToManyDAOImpl.java:-**

```
package com.nt.dao;

import java.util.HashSet;
import java.util.List;
import java.util.Set;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OneToManyDAOImpl implements OneToManyDAO {

    @Override
    public void saveData() {
        Session ses=null;
        User user=null;
        PhoneNumber ph1=null,ph2=null;
        Set<PhoneNumber> set=null;
        Transaction tx=null;
        //get Session
        ses=HibernateUtil.getSession();
        //prepare parent object
        user=new User();
        user.setUserId(1006);
        user.setFirstName("chari");
        //prepare child objects
        ph1=new PhoneNumber();
        ph1.setPhone(129292291);
        ph1.setNumberType("office");

        ph2=new PhoneNumber();
        ph2.setPhone(1288228221);
        ph2.setNumberType("residence");
        //add child objs to Set collection
        set=new HashSet();
        set.add(ph1);set.add(ph2);
        //add childs to parent
        user.setPhones(set);

        //save objects
        try{
            tx=ses.beginTransaction();
            ses.save(user);
            tx.commit();
            System.out.println("Objects are saved");
        } //try
        catch(Exception e){
```

```
        tx.rollback();
    }
}//method

@Override
public void loadData() {
    Session ses=null;
    List<User> list=null;
    Set<PhoneNumber> set=null;
    Query query=null;
    //get Session
    ses=HibernateUtil.getSession();
    //load Users
    query=ses.createQuery("from User");
    list=query.list();
    //process the results
    for(User user:list){
        System.out.println(user);
        set=user.getPhones();
        set.size();
        for(PhoneNumber ph:set){
            System.out.println(ph);
        }//for
    }//for
}
}//method

@Override
public void deleteOnePhoneNumberOfAUser() {
    Session ses=null;
    Query query=null;
    User user=null;
    Set<PhoneNumber> childs=null;
    PhoneNumber ph=null;
    Transaction tx=null;
    //get Session
    ses=HibernateUtil.getSession();
    //Load parent object (User)
    user=ses.get(User.class,1001);
    //get childs of a parent
    childs=user.getPhones();
    //load PhoneNumber object
    ph=ses.get(PhoneNumber.class,888888888L);
    //delete child from collection
    try{
        tx=ses.beginTransaction();
        childs.remove(ph);
        tx.commit();
        System.out.println("Removing 1 child from collection of
```

```
childss");
    }//try
    catch(Exception e){
        tx.rollback();
    }

}//method

@Override
public void deleteAllPhoneNumbersOfAUser() {
    Session ses=null;
    Query query=null;
    User user=null;
    Set<PhoneNumber> childss=null;
    PhoneNumber ph=null;
    Transaction tx=null;
    //get Session
    ses=HibernateUtil.getSession();
    // load parent object
    user=ses.get(User.class,1001);
    //load all child objs of a parent
    childss=user.getPhones();
    //delete all childss of a parent
    try{
        tx=ses.beginTransaction();
        childss.removeAll(childss);
        tx.commit();
        System.out.println("all childss of parent are deleted");
    }//try
    catch(Exception e){
        tx.rollback();
    }
}//method

@Override
public void addNewPhoneNumberToExistingUser() {
    Session ses=null;
    Query query=null;
    User user=null;
    Set<PhoneNumber> childss=null;
    PhoneNumber ph=null;
    Transaction tx=null;
    //get Session
    ses=HibernateUtil.getSession();
    // load parent object
    user=ses.get(User.class,1001);
    //get all chids of a parent
    childss=user.getPhones();
    //create new Child
    ph=new PhoneNumber();
```

```
ph.setPhone(7777777777);
ph.setNumberType("night calls");
//add new child to existing childs
try{
    tx=ses.beginTransaction();
    childs.add(ph);
    tx.commit();
    System.out.println("new JIO number is added ");
} //try
catch(Exception e){
    tx.rollback();
}
}//method

@Override
public void movePhoneNumberFromOneUserToAnotherUser() {
    Session ses=null;
    User srcUser=null,destUser;
    Set<PhoneNumber> srcUserChilds=null;
    Set<PhoneNumber> destUserChilds=null;
    PhoneNumber ph1=null;
    Transaction tx=null;
    // get Session
    ses=HibernateUtil.getSession();
    //Load Source User (1001)
    srcUser=ses.get(User.class,1001);
    //load all childs of 1001 user
    srcUserChilds=srcUser.getPhones();
    //Load 1001 User PhoneNumber that u want to delete
    ph1=ses.get(PhoneNumber.class,777777777L);

    //Load Dest user 1002
    destUser=ses.get(User.class,1002);
    //get All PhoneNumber of dest user 1002
    destUserChilds=destUser.getPhones();
    try{
        tx=ses.beginTransaction();
        //delete phoneNuber src user
        srcUserChilds.remove(ph1);
        tx.commit();
        System.out.println("PhoneNumber is deleted from 1001
user");
    } //try
    catch(Exception e){
        tx.rollback();
    } //catch

    try{
        tx=ses.beginTransaction();
        //add phoneNuber to dest user
    }
```

```
        destUserChilds.add(ph1);
        tx.commit();
        System.out.println("PhoneNumber is added to 1002 user");
    }//try
    catch(Exception e){
        tx.rollback();
    }//catch

}//method
}//class
```

**PhoneNumber.java:-**

```
package com.nt.domain;

public class PhoneNumber {
    private long phone;
    private String numberType;
    private int uid;

    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }

    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }
    public int getUid() {
        return uid;
    }

    public void setUid(int uid) {
        this.uid = uid;
    }

    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" +
numberType + ", uid=" + uid + "]";
    }
}//class
```

User.java:-

```
package com.nt.domain;

import java.util.Set;

public class User {
    private int userId;
    private String firstName;
    private Set<PhoneNumber> phones;

    public User() {
        System.out.println("User:0-param constructor");
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public Set<PhoneNumber> getPhones() {
        return phones;
    }

    public void setPhones(Set<PhoneNumber> phones) {
        this.phones = phones;
    }

    @Override
    public String toString() {
        return "User [userId=" + userId + ", firstName=" + firstName;
    }
}
```

OneToManyTest.java:-

```
package com.nt.test;

import com.nt.dao.OneToManyDAO;
import com.nt.dao.OneToManyDAOFactory;
```

```

import com.nt.utility.HibernateUtil;

public class OneToManyTest {

    public static void main(String[] args) {
        OneToManyDAO dao=null;
        //get DAO
        dao=OneToManyDAOFactory.getInstance();
        //use DAO
        //dao.saveData();
        dao.loadData();
        //dao.deleteOnePhoneNumberOfAUser();
        //dao.deleteAllPhoneNumbersOfAUser();
        //dao.addNewPhoneNumberToExistingUser();
        //dao.movePhoneNumberFromOneUserToAnotherUser();

        //close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class

```

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=null;
    private static ThreadLocal<Session>threadLocal=
        new ThreadLocal<Session>();

    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
    }

    public static Session getSession(){
        Session session=null;
        if(threadLocal.get()==null){
            session=factory.openSession();
            threadLocal.set(session);
        }
        else{
            session=threadLocal.get();
        }
    }
}

```

```

    }
    return session;

}//method

public static void closeSession(){
    if(threadLocal.get()!=null){
        threadLocal.get().close();
        threadLocal.remove();
    }
}//method

public static void closeSessionFactory(){
    factory.close();
}

}

```

### many-to-one association (uni directional) child to parent:

- one-to-many uni-directional association is generally **parent to child association** and many-to-one association is **child to parent association**.
- many-to-one association means the multiple objects of child domain class holds single parent class object.
- To build many-to-one association ,child domain class should have parent class reference type property having capability to make multiple objects of child class pointing to single parent class object.Configure the above special property using **<many-to-one>** tag having **cascade ,lazy** attributes.
- While working with one-to-one, many-to-one associations ,we don't need collection property to build the association.
- While working with one-to-many , many-to-one associations, we need to take collection property to build the association.

*eg: Employee-department (many-to-one)  
Department-employee (one-to-many)*

Tables:

#### Department.java(parent class)

```

public class Department{
private int deptno;
private String deptHead;
private String deptName;
//setters and getters
.....
}

```

**EmpDetails.java(child class)**

```
public class EmpDetails{
private int eno;
private String ename;
private int salary;
private Department dept;
//setters and getters
```

.....

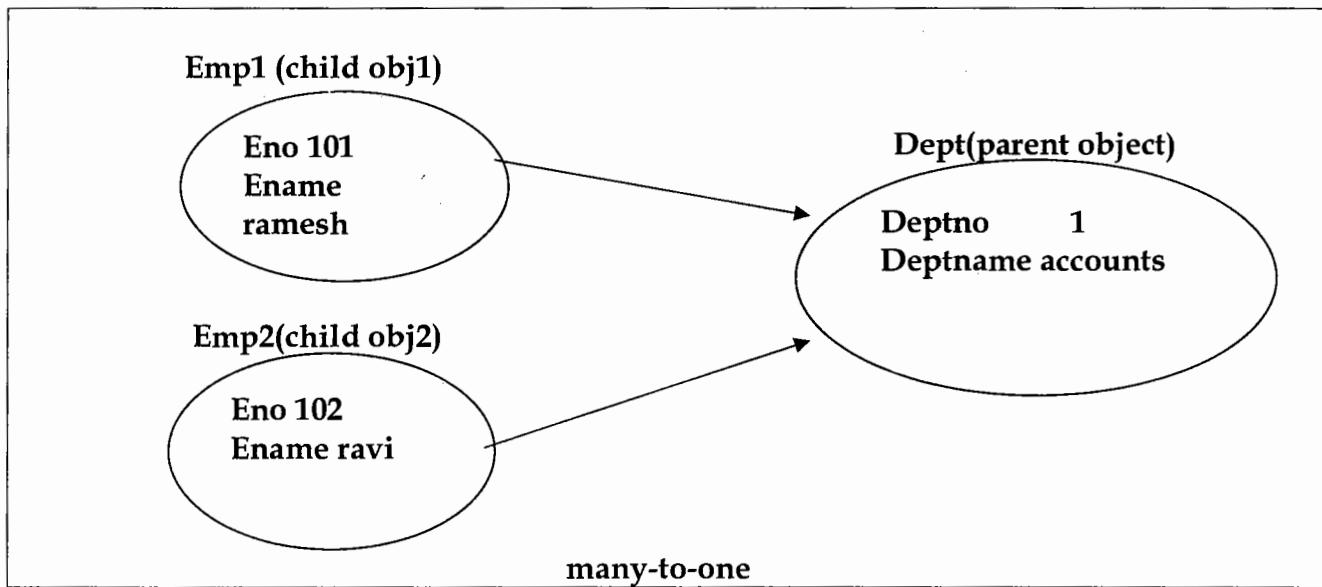
}

**Department.hbm.xml**

```
<class name="com.nt.domain.Department">
  <id name="deptno"/>
  <property name="deptname"/>
  <property name="deptHead"/>
</class>
```

**EmpDetails.hbm.xml**

```
<class name="com.nt.domain.EmpDetails">
  <id name="eno" />
  <property name="ename" />
  <property name="salary" />
  <many-to-one name="dept" class="com.nt.domain.Department"
    cascade="all" column="deptno" lazy="proxy"/>
</class>
```



**hibernate.cfg.xml**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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">scott</property>
        <property name="connection.password">tiger</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/department.hbm.xml"/>
        <mapping resource="com/nt/cfgs/empdetails.hbm.xml"/>
    </session-factory>
</hibernate-configuration>

```

**department.hbm.xml**

```

<!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.domain.Department">
        <id name="deptno"/>
        <property name="deptname"/>
        <property name="deptHead"/>
    </class>
</hibernate-mapping>

```

**empdetails.hbm.xml**

```
<!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.domain.EmpDetails">
    <id name="eno" />
    <property name="ename" />
    <property name="salary" />
    <many-to-one name="dept"
      class="com.nt.domain.Department"
      cascade="all"
      column="deptno"
      lazy="proxy"/>
  </class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  private int eno;
  private String ename;
  private double salary;
  private Department dept;
  public EmpDetails() {
    System.out.println("EmpDetails: 0-param constructor");
  }
  public int getEno() {
    return eno;
  }
  public void setEno(int eno) {
    this.eno = eno;
  }
  public String getEname() {
    return ename;
  }
  public void setEname(String ename) {
    this.ename = ename;
  }
  public double getSalary() {
    return salary;
  }
  public void setSalary(double salary) {
    this.salary = salary;
  }
  public Department getDept() {
```

```
return dept;
}
public void setDept(Department dept) {
this.dept = dept;
}
@Override
public String toString() {
return "EmpDetails [eno=" + eno + ", ename=" + ename + ", salary=" + salary + "]";
}

}
```

**Department.java**

```
package com.nt.domain;

public class Department {
private int deptno;
private String deptname;
private String deptHead;

public Department() {
System.out.println("Department:0-param constructor");
}
public int getDeptno() {
return deptno;
}
public void setDeptno(int deptno) {
this.deptno = deptno;
}
public String getDeptname() {
return deptname;
}
public void setDeptname(String deptname) {
this.deptname = deptname;
}
public String getDeptHead() {
return deptHead;
}
public void setDeptHead(String deptHead) {
this.deptHead = deptHead;
}
@Override
public String toString() {
return "Department [deptno=" + deptno + ", deptname=" + deptname + ", deptHead=" +
deptHead + "]";
}
}
```

**M2ODAO.java**

```
package com.nt.dao;

public interface M2ODAO {
    public void saveEmpsWithDept();
    public void listEmpsWithDept();
    public void addNewEmpToExistingDept();
    public void addExistingEmpToNewDept();
    public void changeExistingEmpToExsitingDept();
    public void deleteAllEmpsWithDept();
    public void deleteEmpFromDept();

}
```

**M2ODAOImpl.java**

```
package com.nt.dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Department;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class M2ODAOImpl implements M2ODAO {

    @Override
    public void saveEmpsWithDept() {
        //get Session
        Session ses=HibernateUtil.getSession();
        //create parent obj
        Department dept=new Department();
        dept.setDeptno(1001); dept.setDeptname("Accounts");
        dept.setDeptHead("SMITH");
        //create child objs
        EmpDetails ed=new EmpDetails();
        ed.setEno(101); ed.setEname("rajesh"); ed.setSalary(9000);
        EmpDetails ed1=new EmpDetails();
        ed1.setEno(102); ed1.setEname("ramesh"); ed1.setSalary(8000);
        //set parent obj to child objs
        ed.setDept(dept); ed1.setDept(dept);
        //save objs (parent to child)
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
```

```
ses.save(ed); ses.save(ed1);
tx.commit();
}
catch(Exception e){
tx.rollback();
}

}//method

@Override
public void listEmpsWithDept() {
// Get Session
Session ses=HibernateUtil.getSession();
// prepare HQL Query
Query query=ses.createQuery("from EmpDetails");
List<EmpDetails>list=query.list();
/* for(EmpDetails ed:list{
System.out.println("child --->" +ed);
//get Associated parent
Department dept=ed.getDept();
System.out.println("parent--->" +dept);
} */


}

@Override
public void addNewEmpToExistingDept() {
// get Session
Session ses=HibernateUtil.getSession();
//Load existing Dept (Parent obj)
Department dept=(Department)ses.get(Department.class,1001);
//create new Employee (child obj)
EmpDetails emp=new EmpDetails();
emp.setEno(654); emp.setEname("karan"); emp.setSalary(9000);
// set Existing Dept to new Employee
emp.setDept(dept);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(emp);
tx.commit();
System.out.println("New Emp is added to existing Dept");
}//try
catch(Exception e){
tx.rollback();
}
}

}//method
```

```
@Override
public void addExistingEmpToNewDept() {
// get Session
Session ses=HibernateUtil.getSession();
// Load existing Employee
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,101);
// create new Department
Department dept=new Department();
dept.setDeptno(1111); dept.setDeptname("HR"); dept.setDeptHead("SCOTT");
// move existing employee new Department
ed.setDept(dept);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.update(ed);
tx.commit();
} //try
catch(Exception e){
tx.rollback();
}
}

@Override
public void changeExistingEmpToExsitingDept() {
// get Session
Session ses=HibernateUtil.getSession();
// Load existing Employee
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,101);
//Load existing Dept (Parent obj)
Department dept=(Department)ses.get(Department.class,1001);
//set Loaded Dept(parent) to Load Emp (child)
ed.setDept(dept);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.update(ed);
tx.commit();
System.out.println("Employee Details are updated");
} //try
catch(Exception e){
tx.rollback();
}
}

@Override
public void deleteAllEmpsWithDept() {
```

```
// get Session
Session ses=HibernateUtil.getSession();
// Load all childs
Query query=ses.createQuery("from EmpDetails");
List<EmpDetails>list=query.list();
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    for(EmpDetails ed:list){
        ses.delete(ed);
    }
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}
}

@Override
public void deleteEmpFromDept() {
// get Session
Session ses=HibernateUtil.getSession();
/* // Load Emp (child)
EmpDetails emp=(EmpDetails)ses.get(EmpDetails.class,101);
//Delete child
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.delete(emp);
    tx.commit();
}
catch(Exception e){
    tx.rollback();
} */
Query query=ses.createQuery("delete from EmpDetails where eno=:no");
query.setInteger("no",102);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    int result=query.executeUpdate();
    System.out.println(result+" no.of records are deleted");
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}
}
```

```
HibernateUtil.closeSession();
}//method
}//class
```

**M2ODAOFactory.java**

```
package com.nt.dao;

public class M2ODAOFactory {

    public static M2ODAO getInstance(){
        return new M2ODAOImpl();
    }

}
```

**HibernateUtil.java**

```
Same as previous
```

**ClientApp.java**

```
package com.nt.test;

import com.nt.dao.M2ODAO;
import com.nt.dao.M2ODAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        M2ODAO dao=M2ODAOFactory.getInstance();
        //call methods
        //dao.saveEmpsWithDept();
        //dao.listEmpsWithDept();
        //dao.addNewEmpToExistingDept();
        //dao.addExistingEmpToNewDept();
        //dao.changeExistingEmpToExsitingDept();
        //dao.deleteAllEmpsWithDept();
        //dao.deleteEmpFromDept();
        dao.deleteEmpFromDept();

        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```

**Note:**

- While performing association navigation from parent to child and child to parent we just need to place one **fk column** in child table, but to perform navigation between parent domain class objects to child domain class objects we need to place **special property** in parent class, similarly for child objects to parent objects navigation we need to place **special property** in child domain class.
- While performing save operation/ insert operation in many-to-one association using child objects, the hibernate first checks parent object is inserted or not, If not inserted, first inserts parent record then inserts the associated child records, so that multiple child records can be linked with one parent record to build **many-to-one** association.

**Select operation in many-to-one association:**

- In this association, the child object will be loaded normally, but we can demand hibernate to load associated parent objects lazily or eagerly by the taking the support of **lazy attribute** placed in `<many-to-one>` tag. The possible values of this attribute are **false, proxy(default), no-proxy**.

**lazy="false":**

→ Loads the associated parent object along with child objects from DATABASE i.e it performs **eager loading**.

**lazy="proxy":**

Loads the child objects normally but loads the associated parent object lazily on demand basis. In this process first the proxy object will be created for parent and real object will be created for parent on demand basis.

**lazy="no-proxy":**

Loads the child objects normally but loads the associated parent object lazily on demand basis. But does not create proxy objects, For this we need **build time byte code instrumentation tool**. If that is not supplied it is same as **lazy="proxy"**

**Note:**

- In one-to-many and many-to-many association the possible values for **lazy** attribute are **extra(best), true(default), false** because they support **collection type** property.
- In one-to-one, many-to-one association the possible values for **lazy** attribute are **proxy, no-proxy(default), false**.
- In one-to-many, many-to-many the best value for cascade attribute is "**all-delete-orphan**"
- In one-to-one and many-to-one association the best value for cascade value is "**all**"

**Delete operation in many-to-one association:**

If you delete one child object the associated parent and its other child object will also be deleted. Here there is no possibility of getting orphan record, so we can't place **cascade="delete-orphan** or **cascade="all-delete-orphan**

**Note:**

→ If you delete child object by loading child object directly using `ses.get(-,-)` (or) `ses.load(-,-)`, it not only deletes child objects and also deletes associated parent objects. To solve this problem change

`cascade="none"` from `cascade="all"`, but this is not practically possible to change `cascade` attribute value in the middle of application execution, so it is recommended to use HQL delete query to delete only child objects with out deleting the associated parent object.

```
Query query=ses.createQuery("delete from EmpDetails where eno=:no");
query.setInteger("no",102);
Transaction tx=null;
try{
tx=ses.beginTransaction();
int result=query.executeUpdate();
System.out.println(result+" no.of records are deleted");
tx.commit();
} //try
catch(Exception e){
tx.rollback();
}
```

- The cascade attribute of many-to-one supports every possible value except **delete-orphan**, **all-delete-orphan**.
- We can take cascade attribute with multiple cascading activities like  
`cascade="delete,save-update" cascade="persist ,delete"`

### **one-to-many/many-to-one bi-directional association:**

- It is combination of **one to many** and **many to one** uni-directional associations
- In unidirectional association only parent to child or child to parent navigation is possible where as in bi-directional association both parent to child and child to parent navigations are possible.
- To build the above navigation the parent class should have **collection type** of special property to hold child's object and child class should have **parent class reference variable** as the special property make multiple child objects pointing to parent class.
- Configure parent class collection property using `<set>/<list>/<map>` tags with the support of `<one-to-many>` tag. Configure child class special property using `<many-to-one>` tag.
- Using single **FK column** we can design tables required for unidirectional association and for bi-directional i.e. no need to take two FK columns.

#### **User.java**

```
public class User{
private int userId
private String firstName
private set<phoneNumber>phones;
//setters and getters
```

```
}
```

#### **phoneNumber.java**

```
public class phoneNumber{
private long phoneNumner;
private String numberType;
private User parent;
//setters and getters
```

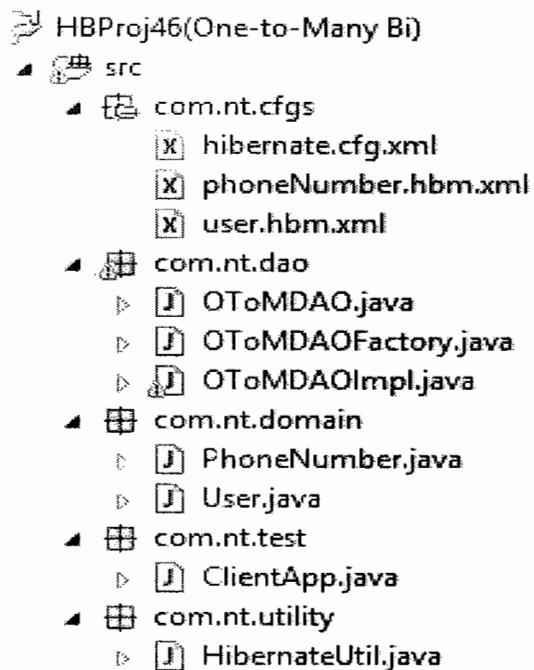
```
}
```

User.hbm.xml:

```
<class name="com.nt.domain.User" table="user_table">
    <id name="userId" column="user_id"/>
    <property name="firstName" column="first_name"/>
    <set name="phones" cascade="all-delete-orphan" lazy="extra" inverse="true" >
        <key column="unid"/>
        <one-to-many class="com.nt.domain.PhoneNumber"/>
    </set>
</class>
```

PhoneNumber.hbm.xml:

```
<class name="com.nt.domain.PhoneNumber" table="Phone_numbers">
    <id name="phone" />
    <property name="numberType" column="number_type"/>
    <many-to-one name="parent"
        class="com.nt.domain.User"
        column="unid"
        cascade="all"
        lazy="proxy"
        />
</class>
```

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
```

```

<session-factory>
  <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>
  <property name="show_sql">true</property>
  <property name="hbm2ddl.auto">update</property>
  <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
  <mapping resource="com/nt/cfgs/user.hbm.xml"/>
  <mapping resource="com/nt/cfgs/phoneNumber.hbm.xml"/>
</session-factory>
</hibernate-configuration>

```

**user.hbm.xml**

```

<!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.domain.User" table="user_table" >
    <id name="userId" column="user_id"/>
    <property name="firstName" column="first_name"/>
    <set name="phones" cascade="all-delete-orphan" lazy="extra" inverse="true" >
      <key column="unid"/>
      <one-to-many class="com.nt.domain.PhoneNumber"/>
    </set>
  </class>
</hibernate-mapping>

```

**phoneNumber.hbm.xml**

```

<!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.domain.PhoneNumber" table="Phone_numbers">
    <id name="phone" />
    <property name="numberType" column="number_type"/>
    <many-to-one name="parent"
      class="com.nt.domain.User"
      column="unia"
      cascade="all"
      lazy="proxy"
      />
  </class>
</hibernate-mapping>

```

PhoneNumber.java

```
package com.nt.domain;

public class PhoneNumber{
    private long phone;
    private String numberType;
    private User parent;
    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }
    public User getParent() {
        return parent;
    }
    public void setParent(User parent) {
        this.parent = parent;
    }
    //toString
    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
    }
}
```

User.java

```
package com.nt.domain;

import java.util.Set;

public class User{
    private int userId;
    private String firstName;
    private Set<PhoneNumber> phones;
    public User() {
```

```
System.out.println("User:0-param constructor");
}
public int getUserId() {
return userId;
}
public void setUserId(int userId) {
this.userId = userId;
}
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}
public Set<PhoneNumber> getPhones() {
return phones;
}
public void setPhones(Set<PhoneNumber> phones) {
this.phones = phones;
}
//toString
@Override
public String toString() {
return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}
/*
<!--
=====parent table=====
create table user_table
(user_id number(5) primary key,
first_name varchar2(20)
);
=====child table=====
create table Phone_numbers
(phone number(10) primary key,
number_type varchar2(20),
unid number(5) references user_table(user_id)
);
-->
*/

```

**OToMDAO.java**

```
package com.nt.dao;
```

```
public interface OToMDAO {  
    public void saveDataUsingUser();  
    public void saveDataUsingPhoneNumber();  
    public void listDataUsingUser();  
    public void listDateUsingPhoneNumber();  
    public void listDataUsingUserWithJoins();  
    public void listDataUsingPhoneNumberWithJoins();  
    public void listDataUsingUserAndFetchJoins();  
    public void listDataUsingUserAndQBCWithFetchMode();  
}
```

**OToDAOImpl.java**

```
package com.nt.dao;  
  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
import org.hibernate.Criteria;  
import org.hibernate.FetchMode;  
import org.hibernate.Query;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import com.nt.domain.PhoneNumber;  
import com.nt.domain.User;  
import com.nt.utility.HibernateUtil;  
  
public class OToDAOImpl implements OToMDAO {  
  
    @Override  
    public void saveDataUsingUser() {  
        // get Session  
        Session ses=HibernateUtil.getSession();  
        // create parent and child objs  
        User user=new User();  
        user.setUserId(1001); user.setFirstName("raja");  
  
        PhoneNumber ph1=new PhoneNumber();  
        ph1.setNumberType("office");  
        ph1.setPhone(99999999);  
  
        PhoneNumber ph2=new PhoneNumber();  
        ph2.setNumberType("residence");  
        ph2.setPhone(88888888);  
        //add parent to child objs
```

```
ph1.setParent(user); ph2.setParent(user);
//add child objs to parent obj (One to many)
Set <PhoneNumber> childs=new HashSet<PhoneNumber>();
childs.add(ph1); childs.add(ph2);
user.setPhones(childs);
//save objs(parent to child)
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(user);
    tx.commit();
    System.out.println("Objs are saved");
} //try
catch(Exception e){
    tx.rollback();
}
}//method
```

```
@Override
public void saveDataUsingPhoneNumber() {
// get Session
Session ses=HibernateUtil.getSession();
//create parent and child objs
User user=new User();
user.setUserId(1002); user.setFirstName("ravi");

PhoneNumber ph1=new PhoneNumber();
ph1.setNumberType("office");
ph1.setPhone(43434344);
```

```
PhoneNumber ph2=new PhoneNumber();
ph2.setNumberType("residence");
ph2.setPhone(56567667);
//add parent to child objs
ph1.setParent(user); ph2.setParent(user);
//set child objs parent obj
Set <PhoneNumber> childs=new HashSet<PhoneNumber>();
childs.add(ph1); childs.add(ph2);
user.setPhones(childs);
//save objs (child to parent)
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(ph1); ses.save(ph2);
    tx.commit();
} //try
catch(Exception e){}
```

```
tx.rollback();
}

}

@Override
public void listDataUsingUser() {
//get Session
Session ses=HibernateUtil.getSession();
// using parent to child objs
Query query=ses.createQuery("from User");
List <User> list=query.list();
for(User user:list){
System.out.println("parent ---->" + user);
//get Associated childs
Set <PhoneNumber> childs=user.getPhones();
for(PhoneNumber ph:childs){
System.out.println("child----->" + ph);
} //for
} //for
} //method

@Override
public void listDateUsingPhoneNumber() {
// get Session
Session ses=HibernateUtil.getSession();
// List Data (child to parent)
Query query=ses.createQuery("from PhoneNumber");
List<PhoneNumber> list=query.list();
for(PhoneNumber phone:list){
System.out.println("Child----->" + phone);
User user=phone.getParent();
System.out.println("parent---->" + user);
} //for
} //method

@Override
public void listDataUsingUserWithJoins() {
// get SEssion
Session ses=HibernateUtil.getSession();
String qry="select u.userId,p.phone from User u full join u.phones p";
Query query=ses.createQuery(qry);
List<Object[]> list=query.list();
for(Object row[]:list ){
for(Object val:row){
System.out.print(val+ " ");
}
System.out.println();
}
```

```
//method

@Override
public void listDataUsingPhoneNumberWithJoins() {
// get Session
Session ses=HibernateUtil.getSession();
//prepare HQL Query
String qry="select p.phone,u.userId from PhoneNumber p right join p.parent u ";
Query query=ses.createQuery(qry);
List<Object[]>list=query.list();
for(Object row[]:list ){
for(Object val:row){
System.out.print(val+" ");
}
System.out.println();
}

}

@Override
public void listDataUsingUserAndFetchJoins() {
//get Session
Session ses=HibernateUtil.getSession();
//prepare Query
String qry="select u from User u inner join fetch u.phones";
Query query=ses.createQuery(qry);
List<User> list=query.list();
for(User user:list){
System.out.println("Parent----> "+user);
Set<PhoneNumber> childs=user.getPhones();
for(PhoneNumber ph:childs){
System.out.println("child----> "+ph);
}
}
}
//methiod

@Override
public void listDataUsingUserAndQBCWithFetchMode() {
//get Session
Session ses=HibernateUtil.getSession();
//prepare QBC obj
Criteria criteria=ses.createCriteria(User.class);
criteria.setFetchMode("phones",FetchMode.DEFAULT);
List<User>list=criteria.list();
for(User user:list){
System.out.println("Parent----> "+user);
Set<PhoneNumber> childs=user.getPhones();
for(PhoneNumber ph:childs){
```

```
System.out.println("child---->" + ph);
} //for
} //for

}//method

}//class
```

**OToMDAOFactory.java**

```
package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}
```

**HibernateUtil.java****ClientApp.java**

```
package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao = OToMDAOFactory.getInstance();
        // call DAO methods
        //dao.saveDataUsingUser();
        //dao.saveDataUsingPhoneNumber();
        //dao.listDataUsingUser();
        //dao.listDateUsingPhoneNumber();
        //dao.listDataUsingUserWithJoins();
        //dao.listDataUsingPhoneNumberWithJoins();
        //dao.listDataUsingUserAndFetchJoins();
        dao.listDataUsingUserAndQBCWithFetchMode();

        // close Session,SessionFactory objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();

    } //main
} //class
```

**Q. What is inverse association or what is inverse="true"?**

Ans: As a programmer of hibernate we build bi-directional association by setting child objects to parent object and parent objects to child objects appropriately, but hibernate has to generate lots of SQL queries and should check for lot of constraints internally to build that association i.e. it generates **SQL queries in all angles to build parent to child and also to build child to parent association.**

→ If we place inverse="true" at one side association that informs the hibernate other side of this association is mirror image of current side, due to this hibernate reduce SQL queries generation little bit to build other side of association. So it is always recommended to enable **inverse association** while dealing with bi-directional association. The rules are → enable inverse="true" at many side in one-to-many bi-directional association by using <set> or <bag> tag. Enable inverse="true" at either side in many-to-many association.

**Example:**

```
<set name="phones" cascade="all-delete-orphan" lazy="extra" inverse="true">
    <key column="unid"/>
    <one-to-many class="com.nt.domain.PhoneNumber"/>
</set>
```

**HQL JOINS:**

- SQL joins are useful to get data from multiple tables by using some built in conditions and to use the SQL joins the DATABASE tables need not be placed in relation.
- HQL joins are given to get data from multiple objects of domain class using single HQL query. For this multiple domain classes, nothing but multiple tables must be there in relation, that means domain classes must be designed supporting **one-to-one, one-to-many, many-to-one, many-to-many relations.**

**HQL supports 4 types of joins****1. Inner join:**

Given common data from right side and left side table.

**2. Left join/Left outer join:**

Gives common data from both right side and left side tables and also gives uncommon data of left side table.

**3. Right join/Right outer join:**

Gives common data and also gives uncommon data of right side table.

**4. Full Join:**

Gives both common and uncommon data of both Database tables.

**Example HQL Join queries:**

(parent to child)

Select u.userid, p.phone from user u inner join u.phones

(child to parent)

Select p.phone, u.userId from phoneNumber p inner join u.parent

→ For example application on Joins refer above application

**Q. What is 1+n select problem?**

Ans: In one-to-many or many-to-one association the hibernate uses **one select SQL query** to get all

parent records/objects and generates “n” number of select queries to get Childs of “n” number of parents .For example if parent table is having 10 records then hibernate generates **one select query** to get all 10 parent records and generates 10 select queries to **get all child records of 10 parent records** i.e. it generates **1+n select queries** to get parent and child records of “n” number parent records. **More SQL queries means more hits to DATABASE software which degrades the performance of the Application.**

### **Q. How can we solve 1+n select problem?**

Ans: We can solve this problem using multiple ways/solutions.

#### **Solution1:Use fetch join as shown below**

```
String qry: "select u from user u inner join fetch u.phones";
Query query = ses.createQuery(qry);
```

→Fetch join disables lazy loading and enables eager loading to generate single select query to get all parent records and the associated child table records.

#### **Solution 2: Using QBC+FetchMode:**

```
Criteria criteria = ses.createCriteria(User.class)
criteria.setFetchMode("phones",FetchMode.JOIN);
List<user>user = criteria.list();
```

#### **Note:**

We can't enable this **fetch** while working with HQL, Native SQL.

#### **Solution3: Using normal joins by specifying propertynames:**

```
Select u.userId, u.firstName, ph.phones, ph.number type from user u innerjoin u.phone ph;
```

### **Q. Can u explain different fetching strategies?**

Ans: Hibernate support 3 types of fetching strategy:

#### **1. select:**

Here hibernate uses one select query to get all parent records and n select queries to get child records of “n” number parent record.

#### **2. subselect:**

Here hibernate uses one select SQL query to get all parent records and one select SQL queries to get all child records of parent record.

#### **3. join:**

Here hibernate uses one select SQL query to get all parent records and associated child records.

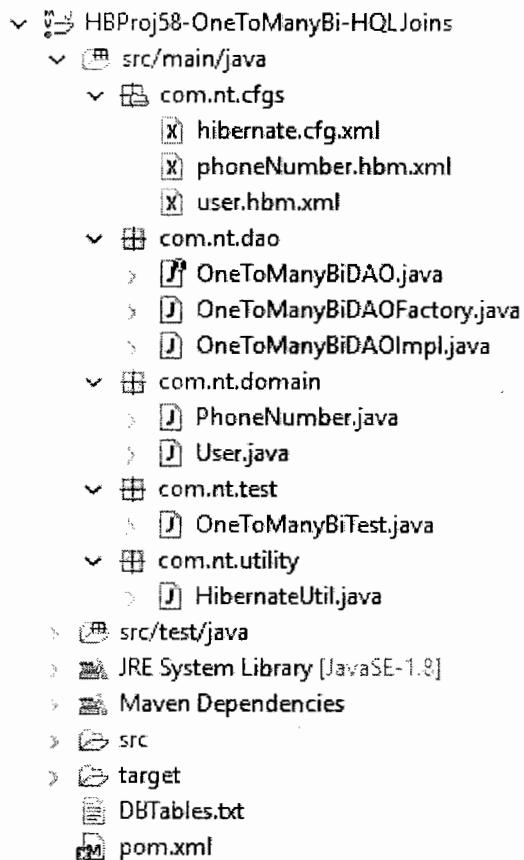
- We can specify fetching strategies by using **fetch** attribute of **<set>**,**<list>**,**<map>** **<bag>** tag  
**<set name="phones" cascade="all-delete-orphan" lazy="extra" inverse="true" fetch="subselect">**  
**<key column="unid"/>**  
**<one-to-many class="com.nt.domain.PhoneNumber"/>**  
**</set>**
- We can also specify fetching strategies by using **criteria.setFetchMode(-,-)** method.

```
criteria.setFetchMode("phones",FetchMode.JOIN);
```

**FetchMode.DEFAULT**

**FetchMode.SELECT**

- Here there is no provision to work with subselect fetch mode. **FetchMode.DEFAULT** means the fetching strategy specified in the mapping file (xml) will be taken as **fetch mode**.



---

### hibernate.cfg.xml:-

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql ">true</property>
        <property name="format_sql ">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/user.hbm.xml"/>
        <mapping resource="com/nt/cfgs/phoneNumber.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

**phoneNumber.hbm.xml:-**

```
<!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.domain.PhoneNumber" table="phone_numbers">
        <id name="phone"/>
        <property name="numberType" column="number_type"/>
        <many-to-one name="parent"
                    class="com.nt.domain.User"
                    column="unid"
                    cascade="all"
                    lazy="proxy"/>
    </class>
</hibernate-mapping>
```

**user.hbm.xml:-**

```
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <!--Inheritance mapping cfgs -->
    <class name="com.nt.domain.User" table="user_table">
        <id name="userId" column="user_id"/>
        <property name="first_name" column="first_name"/>
        <set name="phones" lazy="false" cascade="all-delete-orphan">
```

```
inverse="true" fetch="subselect">
    <key column="unid"/>
    <one-to-many class="com.nt.domain.PhoneNumber"/>
</set>
</class>
</hibernate-mapping>
```

**OneToManyBiDAO.java:-**

```
package com.nt.dao;

public interface OneToManyBiDAO {

    public void loadDataUsingUserWithTheSupportOfHQLJoins();
    public void loadDataUsingPhoneNumberWithTheSupportOfHQLJoins();
    public void loadDataUsingUser();
    public void loadDataUsingUserThroughQBC();

}
```

**OneToManyBiDAOFactory.java:-**

```
package com.nt.dao;

public class OneToManyBiDAOFactory {

    public static OneToManyBiDAO getInstance(){
        return new OneToManyBiDAOImpl();
    }

}
```

**OneToManyBiDAOImpl.java:-**

```
package com.nt.dao;

import java.util.List;
import java.util.Set;

import org.hibernate.Criteria;
import org.hibernate.FetchMode;
import org.hibernate.Query;
import org.hibernate.Session;

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OneToManyBiDAOImpl implements OneToManyBiDAO {

    @Override
    public void loadDataUsingUserWithTheSupportOfHQLJoins() {
        Session ses=null;
```

```
Query query=null;
List<Object[]> list=null;
// get Session
ses=HibernateUtil.getSession();
//prepare HQL Join Query (parent to child)
//query=ses.createQuery("select
u.userId,u.first_name,ph.phone,ph.numberType from User u inner join
u.phones ph");
//query=ses.createQuery("select
u.userId,u.first_name,ph.phone,ph.numberType from User u left join
u.phones ph");
//query=ses.createQuery("select
u.userId,u.first_name,ph.phone,ph.numberType from User u right join
u.phones ph");
query=ses.createQuery("select
u.userId,u.first_name,ph.phone,ph.numberType from User u full join
u.phones ph");
//execute HQL
list=query.list();
//process the result
for(Object row[]:list){
    for(Object val:row){
        System.out.print(val+" ");
    }
    System.out.println();
}
}

}//method

@Override
public void loadDataUsingPhoneNumberWithTheSupportOfHQLJoins() {
    Session ses=null;
    Query query=null;
    List<Object[]> list=null;
    // get Session
    ses=HibernateUtil.getSession();
    //prepare Query
    query=ses.createQuery("select
u.userId,u.first_name,ph.phone,ph.numberType from PhoneNumber ph inner
join ph.parent u ");
    //execute HQL
    list=query.list();
    list.forEach(row->{
        for(Object val:row){
            System.out.print(val+" ");
        }
        System.out.println();
    });
    /*//process the result
    for(Object row[]:list){
```

```
for(Object val:row){
    System.out.print(val+" ");
}
//for
System.out.println();
}//for
*/ } //method

@Override
public void loadDataUsingUser() {
    Session ses=null;

    List<User> list=null;
    Query query=null;
    //get Session
    ses=HibernateUtil.getSession();
    //query=ses.createQuery("select u from User u inner join fetch
u.phones");
    query=ses.createQuery("from User ");
    list=query.list();
    //use Java 8 forEach method
    list.forEach(user->{
        System.out.println(user);
        Set<PhoneNumber> childs=user.getPhones();
        //childs.forEach(ph->System.out.println(ph));
        //childs.forEach(System.out::println);
        childs.stream().forEach(System.out::println);
    });
    /*for(User user:list){
        System.out.println(user);
        childs=user.getPhones();
        for(PhoneNumber ph:childs){
            System.out.println(ph);
        }
    }
 */
} //method

@Override
public void loadDataUsingUserThroughQBC() {
    Session ses=null;
    List<User> list=null;
    Set<PhoneNumber> childs=null;
    Query query=null;
    Criteria criteria=null;
    //get Session
    ses=HibernateUtil.getSession();
    //prepare Criteria Object
    criteria=ses.createCriteria(User.class);
    criteria.setFetchMode("phones",FetchMode.JOIN);
    list=criteria.list();
    for(User user:list){
```

```
System.out.println(user);
childs=user.getPhones();
for(PhoneNumber ph:childs){
    System.out.println(ph);
}//for
}//for
}//method

}//class
```

**PhoneNumber.java:-**

```
package com.nt.domain;

public class PhoneNumber {
    private long phone;
    private String numberType;
    private User parent;

    public long getPhone() {
        return phone;
    }

    public void setPhone(long phone) {
        this.phone = phone;
    }

    public String getNumberType() {
        return numberType;
    }

    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }

    public User getParent() {
        return parent;
    }

    public void setParent(User parent) {
        this.parent = parent;
    }

    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" +
numberType + "]";
    }
}
```

User.java:-

```
package com.nt.domain;

import java.util.Set;

public class User {
    private int userId;
    private String first_name;
    private Set<PhoneNumber> phones;
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public String getFirst_name() {
        return first_name;
    }
    public void setFirst_name(String first_name) {
        this.first_name = first_name;
    }
    public Set<PhoneNumber> getPhones() {
        return phones;
    }
    public void setPhones(Set<PhoneNumber> phones) {
        this.phones = phones;
    }
    @Override
    public String toString() {
        return "User [userId=" + userId + ", first_name=" + first_name
+ "]";
    }
}
```

OneToManyBiTest.java:-

```
package com.nt.test;

import com.nt.dao.OneToOneDAO;
import com.nt.dao.OneToOneDAOFacory;
import com.nt.utility.HibernateUtil;

public class OneToManyBiTest {

    public static void main(String[] args) {
        OneToOneDAO dao=null;
        //get DAO
        dao=OneToOneDAOFacory.getInstance();
        //use DAO
        dao.loadDataUsingUserWithTheSupportOfHQLJoins();
```

```
//dao.loadDataUsingPhoneNumberWithTheSupportOfHQLJoins();
//dao.loadDataUsingUser();
//dao.loadDataUsingUserThroughQBC();
//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
} //main
}//class
```

**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=null;
    private static ThreadLocal<Session>threadLocal=
        new ThreadLocal<Session>();
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
    }
    public static Session getSession(){
        Session session=null;
        if(threadLocal.get()==null){
            session=factory.openSession();
            threadLocal.set(session);
        }
        else{
            session=threadLocal.get();
        }
        return session;
    }
    //method
    public static void closeSession(){
        if(threadLocal.get()!=null){
            threadLocal.get().close();
            threadLocal.remove();
        }
    }
    //method
    public static void closeSessionFactory(){
        factory.close();
    }
}
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>nit</groupId>
  <artifactId>HBProj20LayeredWebApp</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>HBProj20LayeredWebApp Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.1.0.Final</version>
    </dependency>
    <dependency>
      <groupId>com.oracle</groupId>
      <artifactId>ojdbc6</artifactId>
      <version>11.2.0</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <groupId>taglibs</groupId>
      <artifactId>standard</artifactId>
      <version>1.1.2</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>HRProj20LayeredWebApp</finalName>
  </build>
</project>
```

## Many-to-Many association:

- It is combination of one-to-many association from parent and one-to-many association from child.
- In this association one or more parent records will have one or more child records/objects and vice-versa.
- The association between doctors and patients, programmers and projects is many-to-many association. One or more programmer can work for one or more projects and vice-versa.
- To build this association parent class should have collection property to hold one or more child objects and child class should have collection property to hold one or more parent objects.
- To build association we need 3 tables, table1 to hold parent objects/records table2 to hold child records and table3 as join table to build association.

### Example:

<u>programmers(table1)</u>		
<u>pid(pk)</u>	<u>pname</u>	<u>salary</u>
1	raja	9000
2	ravi	8000
3	jani	5000

<u>project(table2)</u>	
<u>projid(pk)</u>	<u>projname</u>
1001	proj1
1002	proj2

<u>Programmers_projects(table3)</u>	
<u>programmers id(pk)</u>	<u>proj_id(fk)(composite pk on both columns)</u>
1	1001
1	1002
2	1001
3	1001
3	1003

### Programmers.java(parent class)

```
public class Programmer{
private int id;
private String phone;
private double salary;
private Set<Project> project = new HashSet<Project>();
//setters and getters
}
```

### Project.java

```
public class Project{
private int projid;
private Set<Programmer> programmer = new HashSet<Programmer>();
//setters and getters
}
```

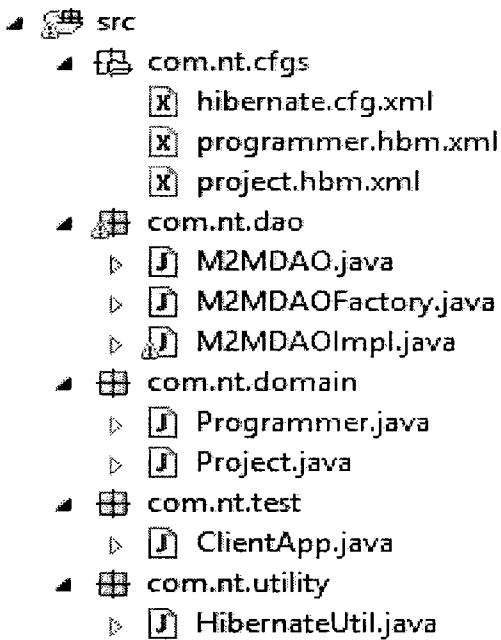
**Programmer.hbm.xml:**

```
<class name="com.nt.domain.Programmer" table="programmers">
  <id name="pid"/>
  <property name="pname"/>
  <property name="salary"/>
  <set name="projects" table="programmers_projects" cascade="all"
        lazy="true" inverse="true">
    <key column="programmer_id"/>
    <many-to-many class="com.nt.domain.Project" column="project_id"/>
  </set>
</class>
```

**project.hbm.xml:**

```
<class name="com.nt.domain.Project" table="projects">
  <id name="proid"/>
  <property name="proname"/>
  <set name="programmers" table="programmers_projects" cascade="all"
        lazy="true">
    <key column="project_id"/>
    <many-to-many class="com.nt.domain.Programmer" column="programmer_id"/>
  </set>
</class>
```

## HBProj47(ManyToManyBi)



**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/programmer.hbm.xml"/>
        <mapping resource="com/nt/cfgs/project.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**programmers.hbm.xml**

```
<!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.domain.Programmer" table="programmers">
        <id name="pid"/>
        <property name="pname"/>
        <property name="salary"/>
        <set name="projects" table="programmers_projects" cascade="all"
                lazy="true" inverse="true">
            <key column="programmer_id"/>
            <many-to-many class="com.nt.domain.Project" column="project_id"/>
        </set>
    </class>
</hibernate-mapping>
```

**project.hbm.xml**

```
<!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.domain.Project" table="projects">
        <id name="proid"/>
        <property name="proname"/>
        <set name="programmers" table="programmers_projects" cascade="all"
                lazy="true" >
```

```
<key column="project_id"/>
<many-to-many class="com.nt.domain.Programmer" column="programmer_id"/>
</set>
</class>
</hibernate-mapping>
```

**Programmer.java**

```
package com.nt.domain;

import java.util.HashSet;
import java.util.Set;

public class Programmer {
    private int pid;
    private String pname;
    private double salary;
    private Set<Project> projects=new HashSet<Project>();
    public Programmer() {
        System.out.println("Programmer:0-param consturctor");
    }
    public int getPid() {
        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    public String getPname() {
        return pname;
    }
    public void setPname(String pname) {
        this.pname = pname;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public Set<Project> getProjects() {
        return projects;
    }
    public void setProjects(Set<Project> projects) {
        this.projects = projects;
    }
    @Override
    public String toString() {
        return "Programmer [pid=" + pid + ", pname=" + pname + ", salary=" + salary + "]";
    }
}
```

```
}

/*
===== table1=====
create table programmers
( pid number(5) primary key,
pname varchar2(20),
salary number(10)
);
=====table2=====
create table projects
( proid number(5) primary key,
proname varchar2(10)
);
=====table3=====
create table programmers_projects(
programmer_id number(5) references Programmers(pid),
project_id number(5) references Projects(proid),
primary key(programmer_id,project_id)
);

*/
```

### Project.java

```
package com.nt.domain;

import java.util.HashSet;
import java.util.Set;

public class Project {
private int proid;
private String proname;
private Set<Programmer> programmers=new HashSet<Programmer>();
public Project() {
System.out.println("Project: 0-param constructor");
}
public int getProid() {
return proid;
}
public void setProid(int proid) {
this.proid = proid;
}
public String getProname() {
return proname;
}
public void setProname(String proname) {
this.proname = proname;
}
```

```
public Set<Programmer> getProgrammers() {  
    return programmers;  
}  
public void setProgrammers(Set<Programmer> programmers) {  
    this.programmers = programmers;  
}  
@Override  
public String toString() {  
    return "Project [proid=" + proid + ", proname=" + proname + ", programmers=" + programmers +  
    "]";  
}  
}
```

**M2MDAO.java**

```
package com.nt.dao;  
  
public interface M2MDAO {  
  
    public void saveDataUsingProgrammer();  
    public void saveDataUsingProjects();  
    public void listDataUsingProgrammer();  
    public void listDataUsingProject();  
    public void addNewProgrammerToExistingProject();  
    public void addExistingProjectToExistingProgrammer();  
    public void deleteProgrammerFromExistingProject();  
    public void deleteAllProgrammersOfExistingProject();  
    public void resignProgrammerToCompany();  
  
}
```

**M2MDAOImpl.java**

```
package com.nt.dao;  
  
import java.util.List;  
import java.util.Set;  
  
import org.hibernate.Query;  
import org.hibernate.Session;  
import org.hibernate.Transaction;  
  
import com.nt.domain.Programmer;  
import com.nt.domain.Project;  
import com.nt.utility.HibernateUtil;  
  
public class M2MDAOImpl implements M2MDAO{  
  
    @Override
```

```
public void saveDataUsingProgrammer() {  
    // get Session  
    Session ses=HibernateUtil.getSession();  
    //create parent and child objs  
    Programmer prgmr1=new Programmer();  
    prgmr1.setPid(1); prgmr1.setPname("raja"); prgmr1.setSalary(9000);  
  
    Programmer prgmr2=new Programmer();  
    prgmr2.setPid(2); prgmr2.setPname("ravi"); prgmr2.setSalary(8000);  
  
    Programmer prgmr3=new Programmer();  
    prgmr3.setPid(3); prgmr3.setPname("jani"); prgmr3.setSalary(8000);  
  
    Project proj1=new Project();  
    proj1.setProid(1001); proj1.setProname("proj1");  
  
    Project proj2=new Project();  
    proj2.setProid(1002); proj2.setProname("proj2");  
  
    //set Projects to Programmer (childs to Parent)  
    prgmr1.getProjects().add(proj1);  
    prgmr1.getProjects().add(proj2);  
  
    prgmr2.getProjects().add(proj2);  
  
    prgmr3.getProjects().add(proj1);  
    prgmr3.getProjects().add(proj2);  
    //set Programmers to Projects (parents to childs)  
    proj1.getProgrammers().add(prgmr1);  
    proj1.getProgrammers().add(prgmr3);  
  
    proj2.getProgrammers().add(prgmr1);  
    proj2.getProgrammers().add(prgmr2);  
    proj2.getProgrammers().add(prgmr3);  
  
    //save objs (parent to child)  
    Transaction tx=null;  
    try{  
        tx=ses.beginTransaction();  
        ses.save(prgmr1);  
        ses.save(prgmr2);  
        ses.save(prgmr3);  
        tx.commit();  
        System.out.println("Objects are saved");  
    } //try  
    catch(Exception e){  
        tx.rollback();  
    }  
}
```

```
//saveDataUsingProgrammer

@Override
public void saveDataUsingProjects() {
    // get Session
    Session ses=HibernateUtil.getSession();
    //create parent and child objs
    Programmer prgmr4=new Programmer();
    prgmr4.setPid(4); prgmr4.setPname("rakesh"); prgmr4.setSalary(9000);

    Project proj3=new Project();
    proj3.setProid(1003); proj3.setProname("proj3");

    Project proj4=new Project();
    proj4.setProid(1004); proj4.setProname("proj4");

    //set Projects to Programmer (childs to Parent)
    prgmr4.getProjects().add(proj3);
    prgmr4.getProjects().add(proj4);

    //set Programmers to Projects (parents to childs)
    proj3.getProgrammers().add(prgmr4);

    proj4.getProgrammers().add(prgmr4);

    //save objs (child to parent)
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.save(proj3);
        ses.save(proj4);
        tx.commit();
        System.out.println("Objects are saved");
    }

    //try
    catch(Exception e){
        tx.rollback();
    }
} //method

@Override
public void listDataUsingProgrammer() {
    // GEt Session
    Session ses=HibernateUtil.getSession();
    Query query=ses.createQuery("from Programmer");
    List<Programmer>list=query.list();
    for(Programmer prgmr:list){
```

```
System.out.println("parent---->"+prgmr);
Set<Project>childs=prgmr.getProjects();
for(Project proj:childs){
System.out.println("Child ----->"+proj);
}//for
}//for
}//method

@Override
public void listDataUsingProject() {
// get Session
Session ses=HibernateUtil.getSession();
Query query=ses.createQuery("from Project");
List<Project>list=query.list();
for(Project proj:list){
System.out.println("child ----->"+proj);
Set<Programmer>childs=proj.getProgrammers();
for(Programmer prgmr:childs){
System.out.println("parent---->"+prgmr);
}//for
}//for
}

@Override
public void addNewProgrammerToExistingProject() {
// Get session
Session ses=HibernateUtil.getSession();
// get Existing Project obj
Project proj=(Project)ses.get(Project.class,1004);
//create new Programmer
Programmer prgmr=new Programmer();
prgmr.setPid(123); prgmr.setPname("karan"); prgmr.setSalary(9000);
// Add Programmer to project.
Transaction tx=null;
try{
tx=ses.beginTransaction();
proj.getProgrammers().add(prgmr);
tx.commit();
System.out.println(" New Programmer added");
} //try
catch(Exception e){
tx.rollback();
}
}//method

@Override
public void addExistingProjectToExistingProgrammer() {
```

```
// get Session
Session ses=HibernateUtil.getSession();
// Load existing Programmer
Programmer prgmr=(Programmer)ses.get(Programmer.class,123);
//Load Existing Project
Project proj=(Project)ses.get(Project.class,1001);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    prgmr.getProjects().add(proj);
    proj.getProgrammers().add(prgmr);
    tx.commit();
    System.out.println(" Programmer is added to project");
} //try
catch(Exception e){
    tx.rollback();
}
HibernateUtil.closeSession();
}//method

@Override
public void deleteProgrammerFromExistingProject() {
    // get Session
    Session ses=HibernateUtil.getSession();
    // Load Existing Project
    Project proj=(Project)ses.get(Project.class,1001);
    //get all the programmers of the project
    Set<Programmer> childs=proj.getProgrammers();
    // Load the Programmer obj that u want to delete
    Programmer prgmr=(Programmer)ses.get(Programmer.class,123);
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        childs.remove(prgmr);
        tx.commit();
        System.out.println("Programmer is removed");
    }
    catch(Exception e){
        tx.rollback();
    }
} //method

@Override
public void deleteAllProgrammersOfExistingProject() {
    // get Session
    Session ses=HibernateUtil.getSession();
    //Load Exsting Project
    Project proj=(Project)ses.get(Project.class,1001);
```

```
// get All programmers of Existing Project
Set<Programmer> childs=proj.getProgrammers();
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.clear();
    tx.commit();
    System.out.println("All programmers of a project deleted");
} //try
catch(Exception e){
    tx.rollback();
}

}//method
@Override
public void resignProgrammerToCompany() {
// get Session
Session ses=HibernateUtil.getSession();
Query query=ses.createQuery("delete from Programmer where pid=:no");
query.setInteger("no",1);
//delete programmer
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    int result=query.executeUpdate();
    tx.commit();
    if(result==0)
        System.out.println("Programmer not deleted");
    else
        System.out.println("Programmer deleted");
} //try
catch(Exception e){
    tx.rollback();
}

}//method
}//class
```

**M2MDAOFactory.java**

```
package com.nt.dao;

public class M2MDAOFactory {

    public static M2MDAO getInstance(){
        return new M2MDAOImpl();
    }
}
```

HibernateUtil.java

Same as previous

ClientApp.java

```
package com.nt.test;

import com.nt.dao.M2MDAO;
import com.nt.dao.M2MDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        M2MDAO dao=M2MDAOFactory.getInstance();
        //dao.saveDataUsingProgrammer();
        //dao.saveDataUsingProjects();
        //dao.listDataUsingProgrammer();
        //dao.listDataUsingProject();
        //dao.addNewProgrammerToExistingProject();
        //dao.addExistingProjectToExistingProgrammer();
        //dao.deleteProgrammerFromExistingProject();
        //dao.deleteAllProgrammersOfExistingProject();
        dao.resignProgrammerToCompany();

        //close Session, SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

<idbag> tag in <many-to-many> association

- In third table of many-to-many association there is no uniqueness because both FK columns are holding duplicate values. To operate third table independently and directly, we need to work with **<idbag>** tag, that is add one additional column in 3 table to have unique values as index values. use **<idbag>** tag only many-to-many association.

- 

Example:

- Keep many to many association example is ready.
- Add one additional column in 3<sup>rd</sup> table to maintain uniqueness.
- Change collection type to **java.util.List** in **Project.java**

```
private List<Programmer> programmer = new ArrayList<Programmer>();
```
- Configure the collection property of Project class in project.hbm.xml file using **<idbag>** tag and also configure the above added index column by using **<idbag>** tag.

```

<class name="com.nt.domain.Project" table="projects">
  <id name="proid"/>
  <property name="proname"/>
  <idbag name="programmers" table="programmers_projects" cascade="all"
         lazy="true" >
    <collection-id column="prgmr_proj_idx" type="int">
      <generator class="increment"/>
    </collection-id>
    <key column="project_id"/>
    <many-to-many class="com.nt.domain.Programmer" column="programmer_id"/>
  </idbag>
</class>

```

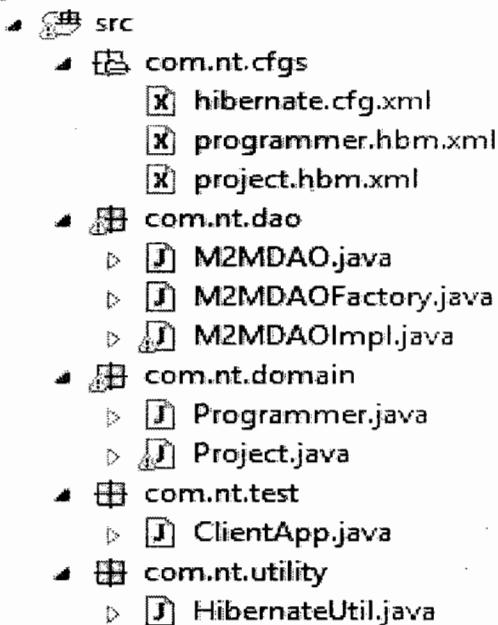
*As of now cannot use assigned, native generator here.*

- e) Save object using child domain class through DAO implementation class.

Join table

<b>programmerid</b>	<b>projected</b>	<b>prgmr proj indx</b>
1	1001	<b>1</b>
2	1001	<b>2</b>
1	1002	<b>3</b>
2	1002	<b>4</b>
1	1003	<b>5</b>

#### HBProj48(ManyToManyBi-idbag)



#### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

```

```
<hibernate-configuration>
<session-factory>
    <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>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping resource="com/nt/cfgs/programmer.hbm.xml"/>
    <mapping resource="com/nt/cfgs/project.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

**programmer.hbm.xml**

```
<!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.domain.Programmer" table="programmers">
    <id name="pid"/>
    <property name="pname"/>
    <property name="salary"/>
    <set name="projects" table="programmers_projects" cascade="all"
          lazy="true" inverse="true">
        <key column="programmer_id"/>
        <many-to-many class="com.nt.domain.Project" column="project_id"/>
    </set>
</class>
</hibernate-mapping>
```

**project.hbm.xml**

```
<!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.domain.Project" table="projects">
    <id name="proid"/>
    <property name="proname"/>
    <idbag name="programmers" table="programmers_projects" cascade="all"
          lazy="true">
        <collection-id column="prgmr_proj_idx" type="int">
            <generator class="increment"/>
        </collection-id>
        <key column="project_id"/>
    </idbag>
</class>
</hibernate-mapping>
```

```
<many-to-many class="com.nt.domain.Programmer" column="programmer_id"/>
</idbag>
</class>
</hibernate-mapping>
```

**Project.java**

```
package com.nt.domain;

import java.util.ArrayList;
import java.util.List;
import java.util.Set;

public class Project {
    private int proid;
    private String proname;
    private List<Programmer> programmers=new ArrayList<Programmer>();
    public Project() {
        System.out.println("Project: 0-param constructor");
    }
    public int getProid() {
        return proid;
    }
    public void setProid(int proid) {
        this.proid = proid;
    }
    public String getProname() {
        return proname;
    }
    public void setProname(String proname) {
        this.proname = proname;
    }
    public List<Programmer> getProgrammers() {
        return programmers;
    }
    public void setProgrammers(List<Programmer> programmers) {
        this.programmers = programmers;
    }
    @Override
    public String toString() {
        return "Project [proid=" + proid + ", proname=" + proname + ", programmers=" + programmers +
    "]";
    }
}
```

**Programmer.java**

```
package com.nt.domain;

import java.util.HashSet;
```

```
import java.util.Set;

public class Programmer {
    private int pid;
    private String pname;
    private double salary;
    private Set<Project> projects=new HashSet<Project>();
    public Programmer() {
        System.out.println("Programmer:0-param consturctor");
    }
    public int getPid() {
        return pid;
    }
    public void setPid(int pid) {
        this.pid = pid;
    }
    public String getPname() {
        return pname;
    }
    public void setPname(String pname) {
        this.pname = pname;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public Set<Project> getProjects() {
        return projects;
    }
    public void setProjects(Set<Project> projects) {
        this.projects = projects;
    }
    @Override
    public String toString() {
        return "Programmer [pid=" + pid + ", pname=" + pname + ", salary=" + salary + "]";
    }
}
```

**M2MDAO.java**

```
package com.nt.dao;

public interface M2MDAO {
    public void saveDataUsingProjects();
}
```

**M2MDAOImpl.java**

```
package com.nt.dao;

import java.util.List;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Programmer;
import com.nt.domain.Project;
import com.nt.utility.HibernateUtil;

public class M2MDAOImpl implements M2DAO{

    @Override
    public void saveDataUsingProjects() {
        // get Session
        Session ses=HibernateUtil.getSession();
        //create parent and child objs
        Programmer prgmr1=new Programmer();
        prgmr1.setPid(1); prgmr1.setPname("rakesh"); prgmr1.setSalary(9000);

        Programmer prgmr2=new Programmer();
        prgmr2.setPid(2); prgmr2.setPname("ravi"); prgmr2.setSalary(9000);

        Project proj1=new Project();
        proj1.setProid(1001); proj1.setProname("proj1");

        Project proj2=new Project();
        proj2.setProid(1002); proj2.setProname("proj2");

        Project proj3=new Project();
        proj3.setProid(1003); proj3.setProname("proj3");

        //set Projects to Programmer (childs to Parent)
        prgmr1.getProjects().add(proj1);
        prgmr1.getProjects().add(proj2);
        prgmr1.getProjects().add(proj3);

        prgmr2.getProjects().add(proj1);
        prgmr2.getProjects().add(proj2);

        //set Programmers to Projects (parents to childs)
        proj1.getProgrammers().add(prgmr1);
        proj1.getProgrammers().add(prgmr2);
```

```
proj2.getProgrammers().add(prgmr1);
proj2.getProgrammers().add(prgmr2);

proj3.getProgrammers().add(prgmr1);

//save objs (child to parent)
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(proj1);
    ses.save(proj2);
    ses.save(proj3);
    tx.commit();
    System.out.println("Objects are saved");
} //try
catch(Exception e){
    tx.rollback();
}
}//method

}//class
```

#### M2MDAOFactory.java

```
package com.nt.dao;

public class M2MDAOFactory {

    public static M2MDAO getInstance(){
        return new M2MDAOImpl();
    }

}
```

#### HibernateUtil.java

Same as above...

#### ClientApp.java

```
package com.nt.test;

import com.nt.dao.M2MDAO;
import com.nt.dao.M2MDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        M2MDAO dao=M2MDAOFactory.getInstance();
```

```

dao.saveDataUsingProjects();

//close Session,SessionFactory
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

## one-to-one association

- Here every object holds one associated object to build the relationship.
- Here parent object holds one associated child object and vice-versa.
- It can be implemented in two ways:
  - a) As **one-to-one PK** (no FK column is required ,we can build association using PK values)
  - b) As **one -to-one FK**(FK Colum support will be taken to build the association)

Q. When do you to use one-to-one PK association and when do you use FK association?

Ans:

### one-to-one PK

- If association is bi-directional and every main object ready to use its associated object's identity value as its identity values then go for **one-to-one PK association**.
- The association between **Student** and **LibraryMembership** is one to one ,more ever we can use the identity value of student as the identity value of LibraryMembership. we can build the association without FK column i.e. we can build this association as one-to-one PK

### one-to-one FK:

- If the association is unidirectional and parent, child objects want to have two different identity values, then we associate those objects through FK column, For this we need to go for **one-to-one FK association**.
- The association between **Person** and **DrivingLicense** is uni directional child to parent association because every Person need not to have License but every license belongs to a Person, more ever we can't take person id as driving license id. For this we need to use **one to one FK association** .

### one-to-one PK implementation

- Here no FK column is required to build association ,we just use **pk values/identity values** to build association.
- Here child object gets identity value from the associated object and uses it as its identity value by using **Foreign generator**.
- We build this relation by suing <one-to-one> tag.

Example : ( one-to-one bi-directional association) pk

<u>student(parent)</u>			<u>lib_membership(child)</u>	
<u>sid(pk)</u>	<u>name</u>	<u>address</u>	<u>lid(pk)</u>	<u>doj(date)</u>
1	raja	hyd	1	20-oct-90
2	ramesh	hyd	2	10-nov-10

#### Student.java(parent class)

```
public class Student {
    private int id;
    private String name;
    private String address;
    private LibraryMembership libraryDetails;
    //setter and getter
}
```

#### LibraryMembership.java

```
public class LibraryMembership {
    private int id;
    private Date joiningDate;
    private Student studentDetails;
    //setter and getter
}
```

#### Student.hbm.xml

```
<class name="com.nt.domain.Student" table="student">
    <id name="id" column="SID">
        <generator class="increment"/>
    </id>
    <property name="name"/>
    <property name="address"/>
    <one-to-one name="libraryDetails"
        class="com.nt.domain.LibraryMembership"
        cascade="all"
        lazy="proxy" />
</class>
```

#### Librabry.hbm.xml

```
<class name="com.nt.domain.LibraryMembership" table="lib_membership">
    <id name="id" column="LID">
        <generator class="foreign">
            <param name="property">studentDetails</param>
        </generator>
    </id>
    <property name="joiningDate" column="doj"/>
    <one-to-one name="studentDetails"
        class="com.nt.domain.Student"
```

```
    cascade="all"
    lazy="proxy"/>
</class>
```

### In one-to-one PK bi-directional association:

- We take Child class reference variable in parent class and parent class reference variable in child class to build the association.
- We don't need FK column here ,but we need **foreign identity generator** to build association.
- Since there is no FK column, so there is no possibility of getting orphan records, due to this **cascade="all"** best value.
- We can not take **lazy="true"** but we can take **lazy="proxy"** or **lazy="no-proxy"**.
- Since there is no FK to build association and association will be build based on PK column values /identity values, so we do not take **inverse="true"** even though it is **bi-directional association**.
- There are no columns, inverse attributes in **one-to-one pk association** based <one-to-one> tag.

Student(parent )			lib_membership (child)		
sid(pk)	name	address	lid (pk)	doj	(date)
101	raja	hyd	101	31-OCT-15	
102	ravi	hyd	102	30-OCT-15	

**Student.java (parent )**

```
public class Student{
    private int id;
    private String name;
    private String address;
    private LibraryMembership libraryDetails;
    //setters & getters
    ...
}
```

**student.hbm.xml**

```
<class name="pkg.Student" table="student">
    <id name="id" column="sid">
        <generator class="increment"/>
    </id>
    <property name="name"/>
    <property name="address"/>
    <one-to-one name="libraryDetails" class="pkg.LibraryMembership" lazy="proxy"/>
</class>
```

**library.hbm.xml**

```
<class name="pkg.LibraryMembership" table="lib_membership">
    <id name="id" column="lid">
        <generator class="foreign">
            <param name="property">studentDetails</param>
        </generator>
    </id>
    <property name="joiningDate" column="doj"/>
    <one-to-one name="studentDetails" class="pkg.Student" lazy="proxy"/>
</class>
```

**LibraryMembership.java(child)**

```
public class LibraryMembership{
    private int id;
    private Date joiningDate;
    private Student studentDetails;
    //setters and getters
    ...
}
```

HBProj49(One -One PK -Bi)

- ▶ src
  - ▶ com.nt.cfgs
    - [X] hibernate.cfg.xml
    - [X] library.hbm.xml
    - [X] student.hbm.xml
  - ▶ com.nt.dao
    - ▷ [J] O2ODAO.java
    - ▷ [J] O2ODAOFactory.java
    - ▷ [J] O2ODAOImpl.java
  - ▶ com.nt.domain
    - ▷ [J] LibraryMembership.java
    - ▷ [J] Student.java
  - ▶ com.nt.test
    - ▷ [J] ClientApp.java
  - ▶ com.nt.utility
    - ▷ [J] HibernateUtil.java

### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<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>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/nt/cfgs/student.hbm.xml"/>
<mapping resource="com/nt/cfgs/library.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

### library.hbm.xml

```
<!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.domain.LibraryMembership" table="lib_membership">
<id name="ia" column="LID">
<generator class="foreign">
<param name="property">studentDetails</param>
</generator>
```

```

</id>
<property name="joiningDate" column="doj"/>
<one-to-one name="studentDetails"
            class="com.nt.domain.Student"
            cascade="all"
            lazy="proxy"/>
</class>
</hibernate-mapping>

```

**student.hbm.xml**

```

<!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.domain.Student" table="student">
    <id name="id" column="SID">
        <generator class="increment"/>
    </id>
    <property name="name"/>
    <property name="address"/>
    <one-to-one name="libraryDetails"
                class="com.nt.domain.LibraryMembership"
                cascade="all"
                lazy="proxy" />
</class>
</hibernate-mapping>

```

**LibraryMembership.java**

```

package com.nt.domain;

import java.util.Date;

public class LibraryMembership {
    private int id;
    private Date joiningDate;
    private Student studentDetails;
    public LibraryMembership() {
        System.out.println("LibraryMembership:0-param constructor");
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public Date getJoiningDate() {
        return joiningDate;
    }
}

```

```
}

public void setJoiningDate(Date joiningDate) {
this.joiningDate = joiningDate;
}
public Student getStudentDetails() {
return studentDetails;
}
public void setStudentDetails(Student studentDetails) {
this.studentDetails = studentDetails;
}
@Override
public String toString() {
return "LibraryMembership [id=" + id + ", joiningDate=" + joiningDate + "]";
}

}
```

### Student.java

```
package com.nt.domain;

public class Student {
private int id;
private String name;
private String address;
private LibraryMembership libraryDetails;
public Student() {
System.out.println("Student:0-param constructor");
}
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public String getAddress() {
return address;
}
public void setAddress(String address) {
this.address = address;
}
public LibraryMembership getLibraryDetails() {
```

```

return libraryDetails;
}
public void setLibraryDetails(LibraryMembership libraryDetails) {
this.libraryDetails = libraryDetails;
}
@Override
public String toString() {
return "Student [id=" + id + ", name=" + name + ", address=" + address + "]";
}

/*SQL> create table student(sid number(5) primary key,name varchar2(20),
address varchar2(20));

SQL> create table lib_membership(lid number(5) primary key,doj date);
*/
}

```

**O2ODAO.java**

```

package com.nt.dao;

public interface O2ODAO {

public void saveDataUsingStudent();
public void saveDataUsingLibraryMembeship();
public void listDataUsingStudent();
public void listDataUsingLibraryMembership();
public void deleteUsingStudent();
public void deleteUsingLibraryMembership();

}

```

**O2ODAOImpl.java**

```

package com.nt.dao;

import java.util.Date;
import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.LibraryMembership;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class O2ODAOImpl implements O2ODAO{

@Override

```

```
public void saveDataUsingStudent() {
    // get Session
    Session ses=HibernateUtil.getSession();
    //prepare parent,child objs
    Student raja=new Student();
    raja.setName("raja"); raja.setAddress("hyd");

    LibraryMembership rajaLib=new LibraryMembership();
    rajaLib.setJoiningDate(new Date());
    //set parent to child and child to parent
    raja.setLibraryDetails(rajaLib);
    rajaLib.setStudentDetails(raja);
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.save(raja);
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }
} //method
```

**@Override**

```
public void saveDataUsingLibraryMembeship() {
    // Get Session
    Session ses=HibernateUtil.getSession();
    //prepare parent,child objs
    Student ravi=new Student();
    ravi.setName("ravi"); ravi.setAddress("hyd");

    LibraryMembership raviLib=new LibraryMembership();
    raviLib.setJoiningDate(new Date());
    //set parent to child and child to parent
    ravi.setLibraryDetails(raviLib);
    raviLib.setStudentDetails(ravi);
    //save objs( child to parent)
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.save(raviLib);
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }
} //method
```

```
@Override  
public void listDataUsingStudent() {  
    //get Session  
    Session ses=HibernateUtil.getSession();  
    //prepare Query  
    Query query=ses.createQuery("from Student");  
    List<Student> list=query.list();  
    for(Student st:list){  
        System.out.println("parent---->" +st);  
        //get associated child  
        LibraryMembership lib=st.getLibraryDetails();  
        System.out.println("child---->" +lib);  
    } //for  
} //method
```

```
@Override  
public void listDataUsingLibraryMembership() {  
    //get Session  
    Session ses=HibernateUtil.getSession();  
    //prepare Query  
    Query query=ses.createQuery("from LibraryMembership");  
    List<LibraryMembership> list=query.list();  
    for(LibraryMembership lib:list){  
        System.out.println("child---->" +lib);  
        //get associated child  
        Student st=lib.getStudentDetails();  
        System.out.println("parent---->" +st);  
    } //for  
} //method
```

```
@Override  
public void deleteUsingStudent() {  
    //get Session  
    Session ses=HibernateUtil.getSession();  
    //Load Student obj  
    Student st=(Student)ses.load(Student.class,1);  
    //delete (parent to child)  
    Transaction tx=null;  
    try{  
        tx=ses.beginTransaction();  
        ses.delete(st);  
        tx.commit();  
    } //try  
    catch(Exception e){  
        tx.rollback();  
    }  
}
```

```

@Override
public void deleteUsingLibraryMembership() {
    //get Session
    Session ses=HibernateUtil.getSession();
    //Load LibraryMemberShip obj
    LibraryMembership lib=(LibraryMembership)ses.load(LibraryMembership.class,2);
    //delete (child to parent)
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.delete(lib);
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }

} //method
}//class

```

**O2ODAOFactory.java**

```

package com.nt.dao;

public class O2ODAOFactory {

    public static O2ODAO getInstance(){
        return new O2ODAOImpl();
    }

}

```

**HibernateUtil.java**

```

package com.nt.test;

import com.nt.dao.O2ODAO;
import com.nt.dao.O2ODAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        O2ODAO dao=O2ODAOFactory.getInstance();
        //dao.saveDataUsingStudent();
        //dao.saveDataUsingLibraryMembeship();
    }
}

```

```
//dao.listDataUsingStudent();
//dao.listDataUsingLibraryMembership();
//dao.deleteUsingStudent();
//dao.deleteUsingLibraryMembership();

//close Session,SessionFactory
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class
```

### one-to-one FK implementation:

- Use this association when parent and child objects want to have two different id values and wants to build relation using FK column. This association can be bi-directional or uni-directional association, but parent and child objects can not to use **same identity value**.
- To build this relation we can not use <one-to-one> tag ,there is no provision to specify **FK column in that tag** .
- This tag is designed by keeping one-to-one PK association in mind, so use <many-to-one> tag with **unique="true"** or **not-null="true"** to bring the **effect one to one association** (**unique="true"** or **not-null = "true"** applies unique key constraint , not-null constraint on FK column to bring the effect of one to one FK association in <many-to-one>).

#### Use case:

Every driving License belongs to a person(one to one ) but every person need not to have driving Llicense, so it is **one-to-one uni directional child to parent association**, More ever we want to separate person id and separate license id, so build this relation as **one -to-one Fk relation**.

**Note:** SQL> create sequence HC\_SEQ start with 1000 increment by 1

**Example One to One FK (Child to Parent)****Person (table) (parent table)**

person_id(pk)	firstName	lastName	age
1	raja	rao	24
2	jani	bhasa	45

```
public class Person{
    private int id;
    private String firstName;
    private String lastName;
    private byte age;
    //setters and getters
    ....
    ....
}
```

**person.hbm.xml**

```
<class name="pkg.Person" table="person">
    <id name="id" column="person_id">
        <generator class="increment"/>
    </id>
    <property name="firstName"/>
    <property name="lastName"/>
    <property name="age"/>
</class>
```

**License (table) (child table)**

license_id(pk)	type	valid_from	valid_to	license_holder (fk)
1001	2-wheeler	1-nov-15	31-oct-45	1

```
public class License{
    private int id;
    private String type;
    private Date dateFrom,dateTo;
    private Person licenseHolder;
    //setters and getters
    ....
    ....
}
```

**license.hbm.xml**

```
<class name="pkg.License" table="lincense">
    <id name="id" column="license_id">
        <generator class="sequence">
            <param name="sequence"/>license_seq</param>
        </generator>
    </id>
    <property name="type"/>
    <property name="dateFrom" column="valid_From"/>
    <property name="dateTo" column="valid_To"/>
    <many-to-one name="licenseHolder"
        class="pkg.Person"
        column="license_holder"
        cascade="all"
        unique="true"
        not-null="true"
        lazy="proxy"/>
</class>
```

## HBProj50(One -One FK -Uni)



**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping resource="com/nt/cfgs/person.hbm.xml"/>
        <mapping resource="com/nt/cfgs/license.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

**person.hbm.xml**

```
<!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.domain.Person" table="Person">
        <id name="id" column="person_id">
            <generator class="increment"/>
        </id>
        <property name="firstName"/>
        <property name="lastName"/>
        <property name="age"/>
    </class>
</hibernate-mapping>
```

**license.hbm.xml**

```
<!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.domain.License" table="License">
        <id name="id" column="license_id">
            <generator class="sequence">
                <param name="sequence">lic_seq</param>
            </generator>
        </id>
        <property name="type"/>
    </class>
</hibernate-mapping>
```

```
<property name="dateFrom" column="valid_from"/>
<property name="dateTo" column="valid_to"/>
<many-to-one name="licenseHolder"
    class="com.nt.domain.Person"
    column="license_holder"
    unique="true"
    not-null="true"
    cascade="all"
    lazy="proxy"/>
</class>
</hibernate-mapping>
```

**Person.java**

```
package com.nt.domain;

public class Person {
    private int id;
    private String firstName,lastName;
    private byte age;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastname() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public byte getAge() {
        return age;
    }
    public void setAge(byte age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "Person [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", age=" +
        age + "]";
    }
}
```

```
}

/*
create table Person(person_id number(5) primary key,
                   firstName varchar2(20),
                   lastName varchar2(20),
                   age number(5));
create table license(license_id number(5) primary key,
                     type varchar2(20),
                     valid_from date,
                     valid_to date,
                     license_holder number(5) references Person(person_id));
create sequence lic_seq start with 1000 increment by 1;
*/
```

### License.java

```
package com.nt.domain;

import java.util.Date;

public class License {
    private int id;
    private String type;
    private Date dateFrom;
    private Date dateTo;
    private Person licenseHolder;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public Date getDateFrom() {
        return dateFrom;
    }
    public void setDateFrom(Date dateFrom) {
        this.dateFrom = dateFrom;
    }
    public Date getDateTo() {
        return dateTo;
    }
}
```

```
}

public void setDateTo(Date dateTo) {
    this.dateTo = dateTo;
}

public Person getLicenseHolder() {
    return licenseHolder;
}

public void setLicenseHolder(Person licenseHolder) {
    this.licenseHolder = licenseHolder;
}

@Override
public String toString() {
    return "License [id=" + id + ", type=" + type + ", dateFrom=" + dateFrom + ", dateTo=" + dateTo +
    "]";
}

}
```

#### O2ODAO.java

```
package com.nt.dao;

public interface O2ODAO {

    public void saveDataUsingLicense();
    public void listDataUsingLicense();
    public void deleteUsingLicense();

}
```

#### O2ODAOImpl.java

```
package com.nt.dao;

import java.util.Date;
import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.License;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class O2ODAOImpl implements O2ODAO{

    @Override
```

```
public void saveDataUsingLicense() {
    // get Session
    Session ses=HibernateUtil.getSession();
    //prepare parent,child objs
    Person person1=new Person();
    person1.setFirstName("raja");
    person1.setLastName("Rao");
    person1.setAge((byte)30);

    License license=new License();
    license.setType("2-wheeler");
    license.setDateFrom(new Date());
    license.setDateTo(new Date(146,3,28));
    license.setLicenseHolder(person1);

    Person person2=new Person();
    person2.setFirstName("ravi");
    person2.setLastName("rathod");
    person2.setAge((byte)35);
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.save(person2);
        ses.save(license);
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }
    System.out.println("objs are saved");
} //method

@Override
public void listDataUsingLicense() {
    // Get Session
    Session ses=HibernateUtil.getSession();
    //prepare Query
    Query query=ses.createQuery("from License");
    List<License> list=query.list();
    for(License license:list){
        System.out.println("Child ---->" +license);
        Person person=license.getLicenseHolder();
        System.out.println("Parent---->" +person);
    } //for
} //method
```

```
@Override
public void deleteUsingLicense() {
    // Get Session
    Session ses=HibernateUtil.getSession();
    //Load License
    License license=(License)ses.get(License.class,1001);
    //delete obj (child tp parent)
    Transaction tx=null;
    try{
        tx=ses.beginTransaction();
        ses.delete(license);
        tx.commit();
    } //try
    catch(Exception e){
        tx.rollback();
    }
}

}//class
```

#### O2ODAOFactory.java

```
package com.nt.dao;

public class O2ODAOFactory {

    public static O2ODAO getInstance(){
        return new O2ODAOImpl();
    }

}
```

#### HibernateUtil.java

```
package com.nt.test;

import com.nt.dao.O2ODAO;
import com.nt.dao.O2ODAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        O2ODAO dao=O2ODAOFactory.getInstance();
        //dao.saveDataUsingLicense();
        //dao.listDataUsingLicense();
    }
}
```

```

dao.deleteUsingLicense();
//close SessionFactory,Session
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

### **Annotation based association mapping:**

Always prefer JPA annotations, Don't use Hibernate annotations. JPA has supplied multiple annotations to build all kinds associations..

#### **one -to-many (uni-directional)**

- @OneToMany → To build association
- @JoinColumn → To specify FK column
- → CascadeType.ALL/DETACH/REFRESH/PERSIST/MERGE/REMOVE (Enum)
- → FetchType.LAZY/EAGER (Enum)

#### **DETACH:**

If main object becomes Detached object,then the associated object becomes **Detached object**.

#### **REFRESH:**

If main object is reloaded/refreshed from database, then the associated object will also be reloaded/refreshed from DATABASE.

#### **MERGE:**

if main object is merged with session then the associated object will also be merged with the session.

- ➔ In annotation based one-to-many association the default fetch type is "lazy" where as in annotation based many-to-one association the default fetch type is "Eager".

#### **Note:**

In JPA annotations there is no cascade type to remove orphan records so use `orphanRemoval = "true"` of `@one-to-many` and `@one-to-one` annotation.

To enable orphan record removal place `orphanRemoval = "true"` in appropriate annotation like

- `@OneToMany`
- `@OneToOne`

`user_table(parent)`

```

|----->User_id(pk)
|----->First_name

```

`phone_number(child)`

```

|-----> phone(pk)
|-----> number_type
|-----> unid(fk)

```

User.java(parent)

```

@Entity
@Table(name="user_table")
public class User{
    @Id
    @Column(name="user_id")
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int userId;

    @Column(name="first_name")
    private String firstName;

    @OneToMany(targetEntity=PhoneNumber.class,
               cascade=CascadeType.ALL,
               fetch=FetchType.LAZY,
               orphanRemoval=true)
    @JoinColumn(name="unid", referencedColumnName="user_id")
    // @Fetch(value=FetchMode.JOIN)
    private Set<PhoneNumber> phones;
    //setter and getter
}

```

PhoneNumber.java(child)

```

@Entity
@Table(name="phone_numbers")
public class PhoneNumber{
    @Id
    private long phone;
    @Column(name="number_type")
    private String numberType;
    //setter and getter
}

```

## HBProj51(Anno-One-to-Many Uni-set Collection)

- src
  - com.nt.cfgs
    - [X] hibernate.cfg.xml
  - com.nt.dao
    - > [J] OToMDAO.java
    - > [J] OToMDAOFactory.java
    - > [J] OToMDAOImpl.java
  - com.nt.domain
    - > [J] PhoneNumber.java
    - > [J] User.java
  - com.nt.test
    - > [J] ClientApp.java
  - com.nt.utility
    - > [J] HibernateUtil.java

hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.User"/>
        <mapping class="com.nt.domain.PhoneNumber"/>
    </session-factory>
</hibernate-configuration>
```

PhoneNumber.java

```
package com.nt.domain;

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

@Entity
@Table(name="phone_numbers")
public class PhoneNumber{
    @Id
    private long phone;
    @Column(name="number_type")
    private String numberType;

    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
}
```

```
}

public void setNumberType(String numberType) {
this.numberType = numberType;
}

//toString
@Override
public String toString() {
return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
}

}
```

**User.java**

```
package com.nt.domain;

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.Fetch;
import org.hibernate.annotations.FetchMode;
import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="user_table")
public class User{

@Id
@Column(name="user_id")
@GenericGenerator(name="gen1",strategy="increment")
@GeneratedValue(generator="gen1")
private int userId;

@Column(name="first_name")
private String firstName;

@OneToMany(targetEntity=PhoneNumber.class,
cascade=CascadeType.ALL,
fetch=FetchType.LAZY,
```

```
        orphanRemoval=true)
@JoinColumn(name="unid", referencedColumnName="user_id")
//@Fetch(value=FetchMode.JOIN)
private Set<PhoneNumber> phones;
public User() {
System.out.println("User:0-param constrctor");
}
public int getUserId() {
return userId;
}
public void setUserId(int userId) {
this.userId = userId;
}
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}
public Set<PhoneNumber> getPhones() {
return phones;
}
public void setPhones(Set<PhoneNumber> phones) {
this.phones = phones;
}
//toString
@Override
public String toString() {
return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}
/*
<!--
=====parent table=====
create table user_table
(user_id number(5) primary key,
first_name varchar2(20)
);
=====child table=====
create table Phone_numbers
( phone number(10) primary key,
number_type varchar2(20),
unid number(5) references user_table(user_id)
);
-->
*/

```

**OToMDAO.java**

```
package com.nt.dao;

public interface OToMDAO {
    public void saveDataUsingUser();
    public void loadDataUsingUser();
    public void addNewPhoneNumberToExistingUser();
    public void deleteUserWithPhoneNumbers();
    public void deleteOnePhoneNumberOfAUser();
    public void deleteAllPhoneNumbersOfAUser();
    public void loadDataUsingUserAndQBCAndFetchModes();

}
```

**OToMDAOImpl.java**

```
package com.nt.dao;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {
    private static final String HQL_GET_ALL_USERS_QUERY="from User";

    @Override
    public void saveDataUsingUser() {
        Session ses=HibernateUtil.getSession();
        //parent obj
        User user=new User();
        user.setFirstName("raja2");
        //child obj1
        PhoneNumber ph1=new PhoneNumber();
        ph1.setPhone(999999);
        ph1.setNumberType("office1");

        //child obj1
        PhoneNumber ph2=new PhoneNumber();
        ph2.setPhone(888888);
    }
}
```

```
ph2.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
Set <PhoneNumber>childs=new HashSet<PhoneNumber>();
childs.add(ph1); childs.add(ph2);
user.setPhones(childs);

//parent obj
User user1=new User();
user1.setFirstName("ravi1");
//child obj1
PhoneNumber ph3=new PhoneNumber();
ph3.setPhone(6666666);
ph3.setNumberType("office");

//child obj1
PhoneNumber ph4=new PhoneNumber();
ph4.setPhone(77777777);
ph4.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
Set <PhoneNumber>childs1=new HashSet<PhoneNumber>();
childs1.add(ph3); childs1.add(ph4);
user1.setPhones(childs1);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(user);
    ses.save(user1);
    tx.commit();
    System.out.println("Objs are saved");
} //try
catch(Exception e){
    tx.rollback();
}

}//save Data

@Override
public void loadDataUsingUser() {
Session ses=null;
Query query=null;
List<User>list=null;
Set<PhoneNumber> childs=null;
//Get Session
ses=HibernateUtil.getSession();
//prepare HQL Query
query=ses.createQuery(HQL_GET_ALL_USERS_QUERY);
list=query.list();
for(User user:list){
```

```
System.out.println("parent---->"+user);
//get all child objs of a parent
childs=user.getPhones();
for(PhoneNumber ph:childs){
System.out.println("child---->"+ph);
}//for
}//for

}// loadData()

//Adding new Child to existing childs of a parent
@Override
public void addNewPhoneNumberToExistingUser() {
//get Session
Session ses=HibernateUtil.getSession();
//Load User obj/parent obj (1001)
User user=(User)ses.get(User.class,1001);

// load existing childs of a User/parent
Set<PhoneNumber> childs=user.getPhones();

//create new Child (new PhoneNumber)
PhoneNumber ph=new PhoneNumber();
ph.setPhone(9998888); ph.setNumberType("personal");
Transaction tx=null;
try{
tx=ses.beginTransaction();
childs.add(ph);
tx.commit();
System.out.println("New child is added");
}//try
catch(Exception e){
tx.rollback();
}
}

@Override
public void deleteUserWithPhoneNumbers() {
//get Session
Session ses=HibernateUtil.getSession();
//load User obj (parent obj)
User user=(User)ses.load(User.class,1002);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.delete(user);
tx.commit();
}
```

```
//try
catch(Exception e){
    tx.rollback();
}
}//method

//Deleting one child from collection of childs beloging to a parent
@Override
public void deleteOnePhoneNumberOfAUser() {
    // Get Session
Session ses=HibernateUtil.getSession();
//Load 1 parent obj (User obj)
User user=(User)ses.get(User.class,1);
// get all childs of a parent obj (PhoneNumber objs)
Set <PhoneNumber> childs=user.getPhones();
//Load PhoneNumber obj
PhoneNumber ph=(PhoneNumber)ses.get(PhoneNumber.class,(long)9999999);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.remove(ph);
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}
}//method

/* deleting all child objs of parent */
@Override
public void deleteAllPhoneNumbersOfAUser() {
// get Session
Session ses=HibernateUtil.getSession();
//Load parent obj
User user=(User)ses.get(User.class,1001);
//get all childs of a parent
Set <PhoneNumber> childs=user.getPhones();
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    childs.clear();
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}
}
```

```

@Override
public void loadDataUsingUserAndQBCAndFetchModes() {
    // get Session
    Session ses=HibernateUtil.getSession();
    // preare Criteria obj
    Criteria criteria=ses.createCriteria(User.class);
    List<User> list=criteria.list();
    for(User user:list){
        System.out.println("parent---->" + user);
        //get all child objs of a parent
        Set<PhoneNumber> child= user.getPhones();
        for(PhoneNumber ph:child){
            System.out.println("child---->" + ph);
        }
    }
}
//class

```

**OToMDAOFactory.java**

```

package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}

```

**HibernateUtil.java**

Same as above

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();
        //call DAO methods
        //dao.saveDataUsingUser();
        //dao.loadDataUsingUser();
    }
}

```

```

//dao.addNewPhoneNumberToExistingUser();
//dao.deleteUserWithPhoneNumbers();
//dao.deleteOnePhoneNumberOfAUser();
// dao.deleteAllPhoneNumbersOfAUser();
dao.loadDataUsingUserAndQBCAndFetchModes();

//close Session
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

### Working with java.util.List type collection

We need to configure index column of child table either with **@IndexColumn** (deprecated) given by hibernate or using **@OrderColumn** annotation (JPA annotation).

#### user table (parent)

user id	first name
1	raja

#### phone number(child)

phone(pk)	number type	unid(fk)	lst_idx
8888	office	1	0
9999	residence	1	1

#### In User.java

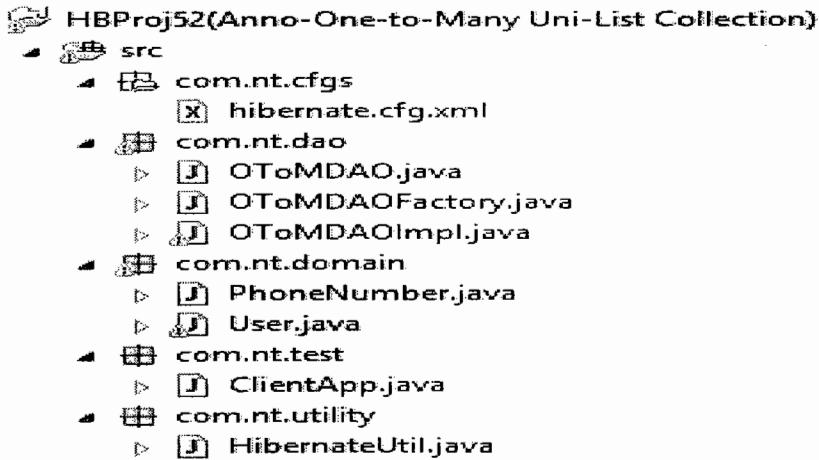
```

//@IndexColumn(name="lst_idx") //deprecated Hibernate annotation
@OrderColumn(name="lst_idx") // JPA annotation
    private List<PhoneNumber> phones;

```

#### Note:

No annotations are given to alternate of <bag> tag as of now.



**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.User"/>
        <mapping class="com.nt.domain.PhoneNumber"/>
    </session-factory>
</hibernate-configuration>
```

**User.java**

```
package com.nt.domain;

import java.util.List;

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.OrderColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.ListIndexBase;

@Entity
@Table(name="user_table1")
public class User{
    @Id
    @Column(name="user_id")
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int userId;

    @Column(name="first_name")
```

```
private String firstName;

@OneToMany(targetEntity=PhoneNumber.class,
           cascade=CascadeType.ALL,
           fetch=FetchType.LAZY,
           orphanRemoval=true)
@JoinColumn(name="unid", referencedColumnName="user_id")
//@IndexColumn(name="lst_idx")
@OrderColumn(name="lst_idx")
private List<PhoneNumber> phones;
public User() {
System.out.println("User:0-param constrctor");
}
public int getUserId() {
return userId;
}
public void setUserId(int userId) {
this.userId = userId;
}
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}
public List<PhoneNumber> getPhones() {
return phones;
}
public void setPhones(List<PhoneNumber> phones) {
this.phones = phones;
}
//toString
@Override
public String toString() {
return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}
/*
<!--
=====parent table=====
create table user_table1
(user_id number(5) primary key,
first_name varchar2(20)
);
=====child table=====
create table Phone_numbers1
( phone number(10) primary key,
number_type varchar2(20),
```

```
  unid number(5) references user_table1(user_id),
  lst_idx number(5)
);
-->
*/
```

**PhoneNumber.java**

```
package com.nt.domain;

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

@Entity
@Table(name="phone_numbers1")
public class PhoneNumber{
  @Id
  private long phone;
  @Column(name="number_type")
  private String numberType;

  public PhoneNumber() {
System.out.println("PhoneNumber:0-param constructor");
}
  //setters and getters
  public long getPhone() {
return phone;
}
  public void setPhone(long phone) {
this.phone = phone;
}
  public String getNumberType() {
return numberType;
}
  public void setNumberType(String numberType) {
this.numberType = numberType;
}
  //toString
  @Override
  public String toString() {
return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
}
}
```

**OToMDAO.java**

```
package com.nt.dao;

public interface OToMDAO {
    public void saveDataUsingUser();

}
```

**OToMDAOImpl.java**

```
package com.nt.dao;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

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

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {

    @Override
    public void saveDataUsingUser() {
        Session ses=HibernateUtil.getSession();
        //parent obj
        User user=new User();
        user.setFirstName("raja2");
        //child obj1
        PhoneNumber ph1=new PhoneNumber();
        ph1.setPhone(999999);
        ph1.setNumberType("office1");

        //child obj1
        PhoneNumber ph2=new PhoneNumber();
        ph2.setPhone(888888);
        ph2.setNumberType("residence");
        //set multiple child objs to parent obj (1 to many)
        List <PhoneNumber>childs=new ArrayList<PhoneNumber>();
        childs.add(ph1); childs.add(ph2);
        user.setPhones(childs);

        //parent obj
        User user1=new User();
        user1.setFirstName("ravi1");
```

```

//child obj1
PhoneNumber ph3=new PhoneNumber();
ph3.setPhone(6666666);
ph3.setNumberType("office");

//child obj1
PhoneNumber ph4=new PhoneNumber();
ph4.setPhone(77777777);
ph4.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
List <PhoneNumber>childs1=new ArrayList<PhoneNumber>();
childs1.add(ph3); childs1.add(ph4);
user1.setPhones(childs1);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(user);
    ses.save(user1);
    tx.commit();
    System.out.println("Objs are saved");
} //try
catch(Exception e){
    tx.rollback();
}
//close Session
HibernateUtil.closeSession();
}//save Data

}//class

```

**OToMDAOFactory.java**

```

package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}

```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.OToMDAO;

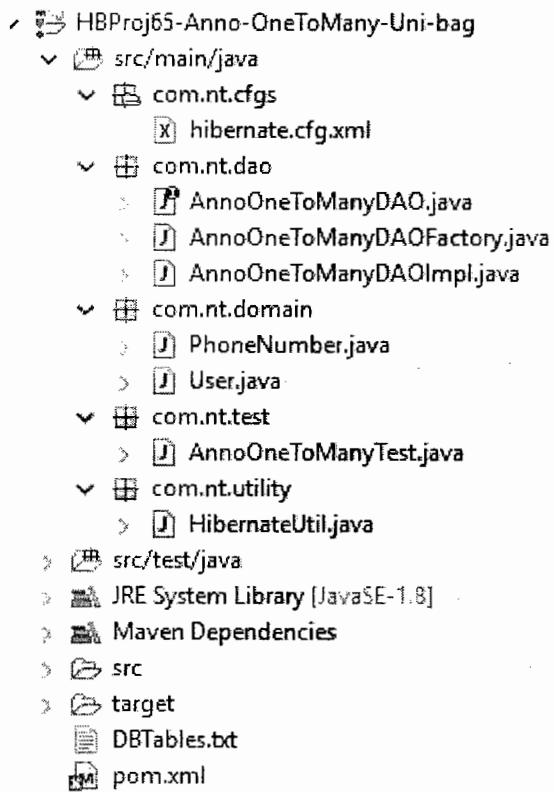
```

```
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();
        //call DAO methods
        dao.saveDataUsingUser();

        //close SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```



hibernate.cfg.xml:-

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql ">true</property>
        <property name="format_sql ">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.User"/>
        <mapping class="com.nt.domain.PhoneNumber"/>
    </session-factory>

</hibernate-configuration>

```

AnnoOneToManyDAO.java:-

```

package com.nt.dao;

public interface AnnoOneToManyDAO {
    public void saveData();
}

```

AnnoOneToManyDAOFactory.java:-

```

package com.nt.dao;

public class AnnoOneToManyDAOFactory {

    public static AnnoOneToManyDAO getInstance(){
        return new AnnoOneToManyDAOImpl();
    }
}

```

AnnoOneToManyDAOImpl.java:-

```

package com.nt.dao;

import java.util.ArrayList;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

```

```

public class AnnoOneToManyDAOImpl implements AnnoOneToManyDAO {

    @Override
    public void saveData() {
        Session ses=null;
        User user=null;
        PhoneNumber ph1=null,ph2=null;
        List<PhoneNumber> childs=null;
        Transaction tx=null;
        //get Session
        ses=HibernateUtil.getSession();
        //prepare objects
        user=new User();
        user.setFirstName("raja");

        ph1=new PhoneNumber();
        ph1.setNumberType("office"); ph1.setPhone(9999999991);
        ph2=new PhoneNumber();
        ph2.setNumberType("residence"); ph2.setPhone(88888881);
        //add phoneNumber objects Set collection
        childs=new ArrayList();
        childs.add(ph1); childs.add(ph2);
        //set chils to parent
        user.setPhones(childs);
        //save objs (parent to childs)
        try{
            tx=ses.beginTransaction();
            ses.save(user);

            tx.commit();
            System.out.println("objects are saved");
        } //try
        catch(Exception e){
            tx.rollback();
        }
    } //method

} //class

```

**PhoneNumber.java:-**

```

package com.nt.domain;

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

@Entity

```

```
@Table(name="phone_numbers2")
public class PhoneNumber {
    @Id
    private long phone;
    @Column(name = "number_type")
    private String numberType;

    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }

    // setters & getters
    public long getPhone() {
        return phone;
    }

    public void setPhone(long phone) {
        this.phone = phone;
    }

    public String getNumberType() {
        return numberType;
    }

    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }

    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" +
numberType + "]";
    }
}
```

User.java:-

```
package com.nt.domain;

import java.util.List;
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.OrderColumn;
import javax.persistence.Table;
```

```
import org.hibernate.annotations.Fetch;
import org.hibernate.annotations.FetchMode;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.IndexColumn;

@Entity
@Table(name = "user_table2")
public class User {
    @Id
    @Column(name = "user_id")
    @GenericGenerator(strategy="increment",name="gen1")
    @GeneratedValue(generator="gen1")
    private int userId;
    @Column(name = "first_name")
    private String firstName;

    @OneToMany(targetEntity = PhoneNumber.class,
               cascade = CascadeType.ALL,
               fetch = FetchType.EAGER,
               orphanRemoval=true)
    @Fetch(value=FetchMode.JOIN)
    @JoinColumn(name = "unid", referencedColumnName = "user_id")
    private List<PhoneNumber> phones;

    public User() {
        System.out.println("User:0-param constructor");
    }

    public int getUserId() {
        return userId;
    }

    public void setUserId(int userId) {
        this.userId = userId;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public List<PhoneNumber> getPhones() {
        return phones;
    }

    public void setPhones(List<PhoneNumber> phones) {
        this.phones = phones;
    }
}
```

```
    }

    @Override
    public String toString() {
        return "User [userId=" + userId + ", firstName=" + firstName +
        ", phones=" + phones + "]";
    }
}
```

**AnnoOneToManyTest.java:-**

```
package com.nt.test;

import com.nt.dao.AnnoOneToManyDAO;
import com.nt.dao.AnnoOneToManyDAOFacory;
import com.nt.utility.HibernateUtil;

public class AnnoOneToManyTest {
    public static void main(String[] args) {
        AnnoOneToManyDAO dao=null;
        //get DAO
        dao=AnnoOneToManyDAOFacory.getInstance();
        //use dAO
        dao.saveData();
        //close objects
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```

**HibernateUtil.java:-**

```
package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        System.out.println("SessionFactory
hashCode:::"+factory.hashCode());
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
```

```
        ses=factory.openSession();
        threadLocal.set(ses);
    }
    System.out.println("Session obj hashCode::"+ses.hashCode());
    return ses;
}//getSession()

public static void closeSession(){
    Session ses=null;

    ses=threadLocal.get();
    if(ses!=null){
        ses.close();
        threadLocal.remove();
    }
}

public static void closeSessionFactory(){
    factory.close();
}

}//class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>nit</groupId>
    <artifactId>HBProj20LayeredWebApp</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>HBProj20LayeredWebApp Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>5.1.0.Final</version>
        </dependency>
        <dependency>
            <groupId>com.oracle</groupId>
            <artifactId>ojdbc6</artifactId>
```

```

        <version>11.2.0</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>javax.servlet-api</artifactId>
        <version>3.1.0</version>
    </dependency>
    <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
    </dependency>

</dependencies>
<build>
    <finalName>HBProj20LayeredWebApp</finalName>
</build>
</project>

```

### Working with java.util.Map type collection:

We need to configure the special index column by using @MapKeyColumn annotation (JPA annotation).

->While working with list collection we can take numeric values as the indexes in order to take our choice values as index we can take map collection, here map collection holds index values as keys and child objects as the values in the elements.

#### (user table)

<u>user_id</u>	<u>first name</u>
1	raja

#### (phone number)

<u>phone</u>	<u>number type</u>	<u>unid(pk)</u>	<u>map indx</u>
8888	office	1	phone1
9999	residence	1	phone2

#### In user.java

```

@MapKeyColumn(name="map_indx")
private Map<String,PhoneNumber> phones;

```

#### ExampleApplication:

 HBProj53(Anno-One-to-Many Uni-Map Collection)

- ▲  src
  -  com.nt.cfgs
    - ☒ hibernate.cfg.xml
  -  com.nt.dao
    - ▷ J OToMDAO.java
    - ▷ J OToMDAOFactory.java
    - ▷ J OToMDAOImpl.java
  -  com.nt.domain
    - ▷ J PhoneNumber.java
    - ▷ J User.java
  -  com.nt.test
    - ▷ J ClientApp.java
  -  com.nt.utility
    - ▷ J HibernateUtil.java

**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.User"/>
        <mapping class="com.nt.domain.PhoneNumber"/>
    </session-factory>
</hibernate-configuration>
```

**User.java**

```
package com.nt.domain;

import java.util.Map;

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.MapKeyColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
@Table(name="user_table3")
public class User{
    @Id
    @Column(name="user_id")
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int userId;

    @Column(name="first_name")
    private String firstName;

    @OneToMany(targetEntity=PhoneNumber.class,
               cascade=CascadeType.ALL,
               fetch=FetchType.LAZY,
               orphanRemoval=true)
    @JoinColumn(name="unid", referencedColumnName="user_id")
    @MapKeyColumn(name="map_idx")
    private Map<String,PhoneNumber> phones;
    public User() {
        System.out.println("User:0-param constructor");
    }
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public Map<String,PhoneNumber> getPhones() {
        return phones;
    }
    public void setPhones(Map<String,PhoneNumber> phones) {
        this.phones = phones;
    }
    //toString
    @Override
```

```

public String toString() {
    return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}
/*
<!--
=====parent table=====
create table user_table3
(user_id number(5) primary key,
firstName varchar2(20)
);
=====child table=====
create table Phone_numbers3
( phone number(10) primary key,
number_type varchar2(20),
unid number(5) references user_table3(userId),
map_indx varchar2(20)
);
-->
*/

```

**PhoneNumber.java**

```

package com.nt.domain;

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

@Entity
@Table(name="phone_numbers3")
public class PhoneNumber{
    @Id
    private long phone;
    @Column(name="number_type")
    private String numberType;

    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
}
```

```
}

public String getNumberType() {
    return numberType;
}

public void setNumberType(String numberType) {
    this.numberType = numberType;
}

//toString
@Override
public String toString() {
    return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
}

}
```

#### OToMDAO.java

```
package com.nt.dao;

public interface OToMDAO {
    public void saveDataUsingUser();
}
```

#### OToMDAOImpl.java

```
package com.nt.dao;

import java.util.HashMap;
import java.util.Map;

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

import com.nt.domain.PhoneNumber;
import com.nt.domain.User;
import com.nt.utility.HibernateUtil;

public class OToMDAOImpl implements OToMDAO {

    @Override
    public void saveDataUsingUser() {
        Session ses=HibernateUtil.getSession();
        //parent obj
        User user=new User();
        user.setFirstName("raja2");
        //child obj1
        PhoneNumber ph1=new PhoneNumber();
}
```

```

ph1.setPhone(999999);
ph1.setNumberType("office1");

//child obj1
PhoneNumber ph2=new PhoneNumber();
ph2.setPhone(888888);
ph2.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
Map <String,PhoneNumber>childs=new HashMap<String,PhoneNumber>();
childs.put("phone1",ph1); childs.put("phone2",ph2);
user.setPhones(childs);

//parent obj
User user1=new User();
user1.setFirstName("ravi1");
//child obj1
PhoneNumber ph3=new PhoneNumber();
ph3.setPhone(6666666);
ph3.setNumberType("office");

//child obj1
PhoneNumber ph4=new PhoneNumber();
ph4.setPhone(77777777);
ph4.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
Map <String,PhoneNumber>childs1=new HashMap<String,PhoneNumber>();
childs1.put("phone1",ph3); childs1.put("phone2",ph4);
user1.setPhones(childs1);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(user);
    ses.save(user1);
    tx.commit();
    System.out.println("Objs are saved");
} //try
catch(Exception e){
    tx.rollback();
}
}//save Data

}//class

```

**OToMDAOFactory.java**

```
package com.nt.dao;
```

```
public class OToMDAOFactory {
```

```
public static OToMDAO getInstance(){
    return new OToMDAOImpl();
}

}
```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```
package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();
        //call DAO methods
        dao.saveDataUsingUser();

        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

**Annotation driven many-to-one association**

- @ManyToOne : To build association
- @JoinColumn: To specify Fk column

->There is no possibility of getting orphan records in many-to-one, many-to-many association, so we don't see orphanRemoval parameters in @ManyToOne, @ManyToMany annotation.

**Note:**

In one to many association defaults fetch type is **LAZY** and in many to one association default fetch type is **EAGER**

 HBProj54(Anno-ManyToOne Uni)

```

    ▲ # src
      ▲ com.nt.cfgs
        X hibernate.cfg.xml
      ▲ com.nt.dao
        ▷ M2ODAO.java
        ▷ M2ODAOFactory.java
        ▷ M2ODAOImpl.java
      ▲ com.nt.domain
        ▷ Department.java
        ▷ EmpDetails.java
      ▲ com.nt.test
        ▷ ClientApp.java
      ▲ com.nt.utility
        ▷ HibernateUtil.java

```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Department"/>
    <mapping class="com.nt.domain.EmpDetails"/>
  </session-factory>
</hibernate-configuration>

```

[EmpDetails.java](#)

```

package com.nt.domain;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

```

```
@Entity
@Table
public class EmpDetails{
    @Id
    private int eno;
    private String ename;
    private double salary;
    @ManyToOne(targetEntity=Department.class,
               cascade=CascadeType.ALL,
               fetch=FetchType.LAZY)
    @JoinColumn(name="deptno",referencedColumnName="deptno")
    private Department dept;
    public EmpDetails() {
        System.out.println("EmpDetails: 0-param constructor");
    }
    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public Department getDept() {
        return dept;
    }
    public void setDept(Department dept) {
        this.dept = dept;
    }
    @Override
    public String toString() {
        return "EmpDetails [eno=" + eno + ", ename=" + ename + ", salary=" + salary + "]";
    }
}
```

### Department.java

```
package com.nt.domain;
```

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

@Entity
@Table
public class Department{
    @Id
    private int deptno;
    private String deptname;
    private String deptHead;
    public Department() {
        System.out.println("Department:0-param constructor");
    }

    public int getDeptno() {
        return deptno;
    }
    public void setDeptno(int deptno) {
        this.deptno = deptno;
    }
    public String getDeptname() {
        return deptname;
    }
    public void setDeptname(String deptname) {
        this.deptname = deptname;
    }
    public String getDeptHead() {
        return deptHead;
    }
    public void setDeptHead(String deptHead) {
        this.deptHead = deptHead;
    }
    @Override
    public String toString() {
        return "Department [deptno=" + deptno + ", deptname=" + deptname + ", deptHead=" + deptHead + "]";
    }
}
```

#### M2ODAO.java

```
package com.nt.dao;

public interface M2ODAO {
    public void saveEmpsWithDept();
    public void listEmpsWithDept();
    public void addNewEmpToExistingDept();
    public void addExistingEmpToNewDept();
```

```
public void changeExistingEmpToExsitingDept();
public void deleteAllEmpsWithDept();
public void deleteEmpFromDept();

}
```

**M2ODAOImpl.java**

```
package com.nt.dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Department;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class M2ODAOImpl implements M2ODAO {

    @Override
    public void saveEmpsWithDept() {
        //get Session
        Session ses=HibernateUtil.getSession();
        //create parent obj
        Department dept=new Department();
        dept.setDeptno(1001); dept.setDeptname("Accounts");
        dept.setDeptHead("SMITH");
        //create child objs
        EmpDetails ed=new EmpDetails();
        ed.setEno(101); ed.setEname("rajesh"); ed.setSalary(9000);
        EmpDetails ed1=new EmpDetails();
        ed1.setEno(102); ed1.setEname("ramesh"); ed1.setSalary(8000);
        //set parent obj to child objs
        ed.setDept(dept); ed1.setDept(dept);
        //save objs (parent to child)
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(ed); ses.save(ed1);
            tx.commit();
        }
        catch(Exception e){
            tx.rollback();
        }
    }
}
```

```
@Override
public void listEmpsWithDept() {
// Get Session
Session ses=HibernateUtil.getSession();
// prepare HQL Query
Query query=ses.createQuery("from EmpDetails");
List<EmpDetails>list=query.list();
for(EmpDetails ed:list){
System.out.println("child ---->" +ed);
//get Associated parent
Department dept=ed.getDept();
System.out.println("parent---->" +dept);
}
}

@Override
public void addNewEmpToExistingDept() {
// get Session
Session ses=HibernateUtil.getSession();
//Load existing Dept (Parent obj)
Department dept=(Department)ses.get(Department.class,1001);
//create new Employee (child obj)
EmpDetails emp=new EmpDetails();
emp.setEno(654); emp.setEname("karan"); emp.setSalary(9000);
// set Existing Dept to new Employee
emp.setDept(dept);
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.save(emp);
tx.commit();
System.out.println("New Emp is added to existing Dept");
} //try
catch(Exception e){
tx.rollback();
}
} //method

@Override
public void addExistingEmpToNewDept() {
// get Session
Session ses=HibernateUtil.getSession();
// Load existing Employee
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,101);
// create new Department
Department dept=new Department();
dept.setDeptno(1111); dept.setDeptname("HR"); dept.setDeptHead("SCOTT");
```

```
// move existing employee new Department
ed.setDept(dept);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.update(ed);
    tx.commit();
}//try
catch(Exception e){
    tx.rollback();
}

}

@Override
public void changeExistingEmpToExsitingDept() {
// get Session
Session ses=HibernateUtil.getSession();
// Load existing Employee
EmpDetails ed=(EmpDetails)ses.get(EmpDetails.class,101);
//Load existing Dept (Parent obj)
Department dept=(Department)ses.get(Department.class,1001);
//set Loaded Dept(parent) to Load Emp (child)
ed.setDept(dept);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.update(ed);
    tx.commit();
    System.out.println("Employee Details are updated");
}//try
catch(Exception e){
    tx.rollback();
}
}

@Override
public void deleteAllEmpsWithDept() {
// get Session
Session ses=HibernateUtil.getSession();
// Load all childs
Query query=ses.createQuery("from EmpDetails");
List<EmpDetails>list=query.list();
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    for(EmpDetails ed:list){
```

```
ses.delete(ed);
}
tx.commit();
}//try
catch(Exception e){
tx.rollback();
}

}

@Override
public void deleteEmpFromDept() {
// get Session
Session ses=HibernateUtil.getSession();
/* // Load Emp (child)
EmpDetails emp=(EmpDetails)ses.get(EmpDetails.class,101);
//Delete child
Transaction tx=null;
try{
tx=ses.beginTransaction();
ses.delete(emp);
tx.commit();
}
catch(Exception e){
tx.rollback();
} */

Query query=ses.createQuery("delete from EmpDetails where eno=:no");
query.setInteger("no",102);
Transaction tx=null;
try{
tx=ses.beginTransaction();
int result=query.executeUpdate();
System.out.println(result+" no.of records are deleted");
tx.commit();
}
//try
catch(Exception e){
tx.rollback();
}
}//method
}//class
```

### M2ODAOFactory.java

```
package com.nt.dao;

public class M2ODAOFactory {
```

```
public static M2ODAO getInstance(){
    return new M2ODAOImpl();
}
```

### HibernateUtil.java

### ClientApp.java

```
package com.nt.test;

import com.nt.dao.M2ODAO;
import com.nt.dao.M2ODAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        M2ODAO dao=M2ODAOFactory.getInstance();
        //call methods
        //dao.saveEmpsWithDept();
        dao.listEmpsWithDept();
        //dao.addNewEmpToExistingDept();
        //dao.addExistingEmpToNewDept();
        //dao.changeExistingEmpToExsitingDept();
        //dao.deleteAllEmpsWithDept();
        //dao.deleteEmpFromDept();
        //dao.deleteEmpFromDept();

        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```

## one-to-many association (bi-directional)

- @OneToMany: To build parent to child association
- @ManyToOne: To build child to parent association
- @JoinKey

 HBProj55(Anno-One-to-Many Bi)

```

    ▲ src
      ▲ com.nt.cfgs
        x hibernate.cfg.xml
      ▲ com.nt.dao
        ▷ J OTcMDAO.java
        ▷ J OTcMDAOFactory.java
        ▷ J OTcMDAOImpl.java
      ▲ com.nt.domain
        ▷ J PhoneNumber.java
        ▷ J User.java
      ▲ com.nt.test
        ▷ J ClientApp.java
      ▲ com.nt.utility
        ▷ J HibernateUtil.java
  
```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.User"/>
    <mapping class="com.nt.domain.PhoneNumber"/>
  </session-factory>
</hibernate-configuration>
  
```

[PhoneNumber.java](#)

```

package com.nt.domain;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
  
```

```
@Entity
@Table(name="phone_numbers")
public class PhoneNumber{
    @Id
    private long phone;
    @Column(name="number_type")
    private String numberType;

    @ManyToOne(targetEntity=User.class,cascade=CascadeType.ALL,fetch=FetchType.LAZY)
    @JoinColumn(name="unid",referencedColumnName="user_id")
    private User parent;
    public PhoneNumber() {
        System.out.println("PhoneNumber:0-param constructor");
    }
    //setters and getters
    public long getPhone() {
        return phone;
    }
    public void setPhone(long phone) {
        this.phone = phone;
    }
    public String getNumberType() {
        return numberType;
    }
    public void setNumberType(String numberType) {
        this.numberType = numberType;
    }

    public User getParent() {
        return parent;
    }
    public void setParent(User parent) {
        this.parent = parent;
    }

    //toString
    @Override
    public String toString() {
        return "PhoneNumber [phone=" + phone + ", numberType=" + numberType + "]";
    }
}
```

User.java

```
package com.nt.domain;
```

```
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.GenericGenerator;

@Entity
@Table(name="user_table")
public class User{
    @Id
    @Column(name="user_id")
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int userId;

    @Column(name="first_name")
    private String firstName;

    @OneToMany(targetEntity=PhoneNumber.class,
               cascade=CascadeType.ALL,
               fetch=FetchType.LAZY,
               orphanRemoval=true,
               mappedBy="parent")
    // @JoinColumn(name="unid", referencedColumnName="user_id")
    private Set<PhoneNumber> phones;
    public User() {
        System.out.println("User:0-param constrctor");
    }
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
}
```

```

public Set<PhoneNumber> getPhones() {
    return phones;
}
public void setPhones(Set<PhoneNumber> phones) {
    this.phones = phones;
}
//toString
@Override
public String toString() {
    return "User [userId=" + userId + ", firstName=" + firstName + "]";
}
}
/*
<!--
=====parent table=====
create table user_table
(user_id number(5) primary key,
first_name varchar2(20)
);
=====child table=====
create table Phone_numbers
( phone number(10) primary key,
number_type varchar2(20),
uid number(5) references user_table(user_id)
);
-->
*/

```

**OToMDAO.java**

```

package com.nt.dao;

public interface OToMDAO {
    public void saveDataUsingUser();
    public void saveDataUsingPhoneNumber();
    public void loadDataUsingUser();
}

```

**OToMDAOImpl.java**

```

package com.nt.dao;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;

```

```
import org.hibernate.Transaction;  
  
import com.nt.domain.PhoneNumber;  
import com.nt.domain.User;  
import com.nt.utility.HibernateUtil;  
  
public class OToMDAOImpl implements OToMDAO {  
    private static final String HQL_GET_ALL_USERS_QUERY="from User";  
  
    @Override  
    public void saveDataUsingUser() {  
        Session ses=HibernateUtil.getSession();  
        //parent obj  
        User user=new User();  
        user.setFirstName("raja2");  
        //child obj1  
        PhoneNumber ph1=new PhoneNumber();  
        ph1.setPhone(999999);  
        ph1.setNumberType("office1");  
  
        //child obj1  
        PhoneNumber ph2=new PhoneNumber();  
        ph2.setPhone(888888);  
        ph2.setNumberType("residence");  
        //set multiple child objs to parent obj (1 to many)  
        Set <PhoneNumber>childs=new HashSet<PhoneNumber>();  
        childs.add(ph1); childs.add(ph2);  
        user.setPhones(childs);  
  
        //parent obj  
        User user1=new User();  
        user1.setFirstName("ravi1");  
        //child obj1  
        PhoneNumber ph3=new PhoneNumber();  
        ph3.setPhone(6666666);  
        ph3.setNumberType("office");  
  
        //child obj1  
        PhoneNumber ph4=new PhoneNumber();  
        ph4.setPhone(77777777);  
        ph4.setNumberType("residence");  
        //set multiple child objs to parent obj (1 to many)  
        Set <PhoneNumber>childs1=new HashSet<PhoneNumber>();  
        childs1.add(ph3); childs1.add(ph4);  
        user1.setPhones(childs1);  
        Transaction tx=null;  
        try{  
            tx=ses.beginTransaction();
```

```
ses.save(user);
ses.save(user1);
tx.commit();
System.out.println("Objs are saved");
}//try
catch(Exception e){
    tx.rollback();
}
}//save Data
```

**@Override**

```
public void loadDataUsingUser() {
Session ses=null;
Query query=null;
List<User>list=null;
Set<PhoneNumber> childs=null;
//Get Session
ses=HibernateUtil.getSession();
//prepare HQL Query
query=ses.createQuery(HQL_GET_ALL_USERS_QUERY);
list=query.list();
for(User user:list){
    System.out.println("parent---->" +user);
    //get all child objs of a parent
    childs=user.getPhones();
    for(PhoneNumber ph:childs){
        System.out.println("child---->" +ph);
    }
}
}
}//loadData()
```

**@Override**

```
public void saveDataUsingPhoneNumber() {
//get Session
Session ses=HibernateUtil.getSession();
User user=new User();
user.setFirstName("raja2");
//child obj1
PhoneNumber ph1=new PhoneNumber();
ph1.setPhone(999999);
ph1.setNumberType("office1");

//child obj1
PhoneNumber ph2=new PhoneNumber();
ph2.setPhone(888888);
ph2.setNumberType("residence");
//set multiple child objs to parent obj (1 to many)
```

```

Set <PhoneNumber>childs=new HashSet<PhoneNumber>();
childs.add(ph1); childs.add(ph2);
//set parent to childs
ph1.setParent(user); ph2.setParent(user);
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(ph1);
    ses.save(ph2);
    tx.commit();
    System.out.println("Objs are saved");
} //try
catch(Exception e){
    tx.rollback();
}
}
}//class

```

**OToMDAOFactory.java**

```

package com.nt.dao;

public class OToMDAOFactory {
    public static OToMDAO getInstance(){
        return new OToMDAOImpl();
    }
}

```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.OToMDAO;
import com.nt.dao.OToMDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // get DAO
        OToMDAO dao=OToMDAOFactory.getInstance();
        //call DAO methods
        //dao.saveDataUsingUser();
        dao.saveDataUsingPhoneNumber();
        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}

```

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

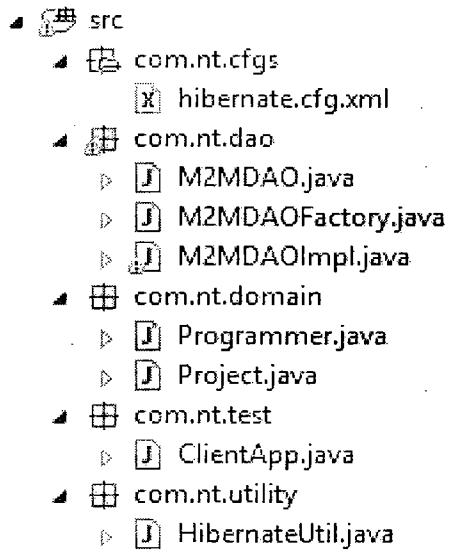
### Understating the mappedBy property/parameter in hibernate:

- While building bi-directional association ,it allows to specify the special property that owns the relationship i.e. the property on which **@JoinColumn annotation is placed** due to this there is no need of writing **@JoinColumn annotation** in the current domain on special property.
- This must be added on **non-owning side property** and specify the property that owns the relationship, this parameter is available in only **@OneToOne @oneToMany, @ManyToMany but not in @ManyToOne.**
- This **mappedBy** parameter is optional to use, when not used we should write **@JoinColumn** on special properties of **both parent and child domain classes.**

### Annotation driven many-to-many association:

- **@ManyToMany** : To build many-to-many association
- **@JoinColumn** : To specify FK column.
- **@JoinTable** : To specify third table configurations.

### HBProj56(Anno-ManyToMany-Bi)



### hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

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

```

<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>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping class="com.nt.domain.Programmer"/>
<mapping class="com.nt.domain.Project"/>
</session-factory>
</hibernate-configuration>

```

**Programmer.java**

```

package com.nt.domain;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Table(name="programmers")
@Entity
public class Programmer{
    @Id
    private int pid;
    private String pname;
    private double salary;

    @ManyToMany(targetEntity=Project.class,cascade=CascadeType.ALL)
    @JoinTable(name="programmers_projects",
    joinColumns=@JoinColumn(name="programmer_id",referencedColumnName="pid"),
    inverseJoinColumns=@JoinColumn(name="project_id",referencedColumnName="proid"))
    private Set<Project> projects=new HashSet<Project>();
    public Programmer() {
        System.out.println("Programmer:0-param consturctor");
    }
    public int getPid() {
        return pid;
    }
}

```

```
}

public void setPid(int pid) {
this.pid = pid;
}

public String getPname() {
return pname;
}

public void setPname(String pname) {
this.pname = pname;
}

public double getSalary() {
return salary;
}

public void setSalary(double salary) {
this.salary = salary;
}

public Set<Project> getProjects() {
return projects;
}

public void setProjects(Set<Project> projects) {
this.projects = projects;
}

@Override
public String toString() {
return "Programmer [pid=" + pid + ", pname=" + pname + ", salary=" + salary + "]";
}
}

/*
===== table1=====
create table programmers
( pid number(5) primary key,
pname varchar2(20),
salary number(10)
);

=====table2=====
create table projects
( proid number(5) primary key,
proname varchar2(10)
);

=====table3=====
create table programmers_projects(
programmer_id number(5) references Programmers(pid),
project_id number(5) references Projects(proid),
primary key(programmer_id,project_id)
);

*/
```

Project.java

```
package com.nt.domain;
import java.util.HashSet;
import java.util.Set;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;
@Entity
@Table(name="projects")
public class Project{
@Id
private int proid;
private String proname;
@ManyToMany(targetEntity=Programmer.class,
           cascade=CascadeType.ALL,
           mappedBy="projects")
private Set<Programmer> programmers=new HashSet<Programmer>();
public Project() {
System.out.println("Project: 0-param constructor");
}
public int getProid() {
return proid;
}
public void setProid(int proid) {
this.proid = proid;
}
public String getProname() {
return proname;
}
public void setProname(String proname) {
this.proname = proname;
}
public Set<Programmer> getProgrammers() {
return programmers;
}
public void setProgrammers(Set<Programmer> programmers) {
this.programmers = programmers;
}
@Override
public String toString() {
return "Project [proid=" + proid + ", proname=" + proname + ", programmers=" + programmers +
"]";
}
}
```

**M2MDAO.java**

```
package com.nt.dao;

public interface M2MDAO {
    public void saveDataUsingProgrammer();
    public void saveDataUsingProjects();
    public void listDataUsingProgrammer();
    public void listDataUsingProject();
}
```

**M2MDAOImpl.java**

```
package com.nt.dao;

import java.util.List;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Programmer;
import com.nt.domain.Project;
import com.nt.utility.HibernateUtil;

public class M2MDAOImpl implements M2MDAO{

    @Override
    public void saveDataUsingProgrammer() {
        // get Session
        Session ses=HibernateUtil.getSession();
        //create parent and child objs
        Programmer prgmr1=new Programmer();
        prgmr1.setPid(1); prgmr1.setPname("raja"); prgmr1.setSalary(9000);

        Programmer prgmr2=new Programmer();
        prgmr2.setPid(2); prgmr2.setPname("ravi"); prgmr2.setSalary(8000);

        Programmer prgmr3=new Programmer();
        prgmr3.setPid(3); prgmr3.setPname("jani"); prgmr3.setSalary(8000);

        Project proj1=new Project();
        proj1.setProid(1001); proj1.setProname("proj1");

        Project proj2=new Project();
        proj2.setProid(1002); proj2.setProname("proj2");

        //set Projects to Programmer (childs to Parent)
        prgmr1.getProjects().add(proj1);
    }
}
```

```
prgmr1.getProjects().add(proj2);

prgmr2.getProjects().add(proj2);

prgmr3.getProjects().add(proj1);
prgmr3.getProjects().add(proj2);

//set Programmers to Projects (parents to childs)
proj1.getProgrammers().add(prgmr1);
proj1.getProgrammers().add(prgmr3);

proj2.getProgrammers().add(prgmr1);
proj2.getProgrammers().add(prgmr2);
proj2.getProgrammers().add(prgmr3);

//save objs (parent to child)
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(prgmr1);
    ses.save(prgmr2);
    ses.save(prgmr3);
    tx.commit();
    System.out.println("Objects are saved");
}i/try
catch(Exception e){
    tx.rollback();
}
//saveDataUsingProgrammer

@Override
public void saveDataUsingProjects() {
    // get Session
Session ses=HibernateUtil.getSession();
//create parent and child objs
Programmer prgmr4=new Programmer();
prgmr4.setPid(4); prgmr4.setPname("rakesh"); prgmr4.setSalary(9000);

Project proj3=new Project();
proj3.setProid(1003); proj3.setProname("proj3");

Project proj4=new Project();
proj4.setProid(1004); proj4.setProname("proj4");

//set Projects to Programmer (childs to Parent)
prgmr4.getProjects().add(proj3);
prgmr4.getProjects().add(proj4);
```

```
//set Programmers to Projects (parents to childs)
proj3.getProgrammers().add(prgmr4);
proj4.getProgrammers().add(prgmr4);

//save objs (child to parent)
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(proj3);
    ses.save(proj4);
    tx.commit();
    System.out.println("Objects are saved");

}//try
catch(Exception e){
    tx.rollback();
}
}//method
@Override
public void listDataUsingProgrammer() {
    // GET Session
    Session ses=HibernateUtil.getSession();
    Query query=ses.createQuery("from Programmer");
    List<Programmer>list=query.list();
    for(Programmer prgmr:list){
        System.out.println("parent---->" +prgmr);
        Set<Project>childs=prgmr.getProjects();
        for(Project proj:childs){
            System.out.println("Child ----->" +proj);
        }
    }
    //for
    //for
}//method
@Override
public void listDataUsingProject() {
    // get Session
    Session ses=HibernateUtil.getSession();
    Query query=ses.createQuery("from Project");
    List<Project>list=query.list();
    for(Project proj:list){
        System.out.println("child ----->" +proj);
        Set<Programmer>childs=proj.getProgrammers();
        for(Programmer prgmr:childs){
            System.out.println("parent---->" +prgmr);
        }
    }
    //for
    //for
}//method

}//class
```

**M2MDAOFactory.java**

```
package com.nt.dao;

public class M2MDAOFactory {

    public static M2MDAO getInstance(){
        return new M2MDAOImpl();
    }

}
```

**HibernateUtil.java**

Same as previous..

**ClientApp.java**

```
package com.nt.test;

import com.nt.dao.M2MDAO;
import com.nt.dao.M2MDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        M2MDAO dao=M2MDAOFactory.getInstance();
        //dao.saveDataUsingProgrammer();
        //dao.saveDataUsingProjects();
        //dao.listDataUsingProgrammer();
        dao.listDataUsingProject();

        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

**@CollectionId (annotation based alternate to <idbag> tag:**

In many-to-many association there is no uniqueness in 3<sup>rd</sup> DATABASE table /join table. By adding one index column in third table we can bring that uniqueness to add this index column and to generate values in that column by configuring one or another generator (except assigned ,native) We can use **@CollectionId** annotation as alternate to **<idbag>** tag.

**Example:****a) Add index column in 3<sup>rd</sup> table**

Alter table programmer\_project add prgmr\_proj\_idx number (5)

- b) Change collection type List either in parent/child class or in both classes and also configure **@CollectionId** annotation.

In Programmer.java

```

@ManyToMany(targetEntity=Project.class,cascade=CascadeType.ALL)

@JoinTable(name="programmers_projects",

joinColumns=@JoinColumn(name="programmer_id",referencedColumnName="pid"),

inverseJoinColumns=@JoinColumn(name="project_id",referencedColumnName="proid")

)

@GenericGenerator(name="gen1",strategy="increment")

@CollectionId(columns=@Column(name="prgmr_proj_idx"),

type=@Type(type="int"),

generator="gen1")

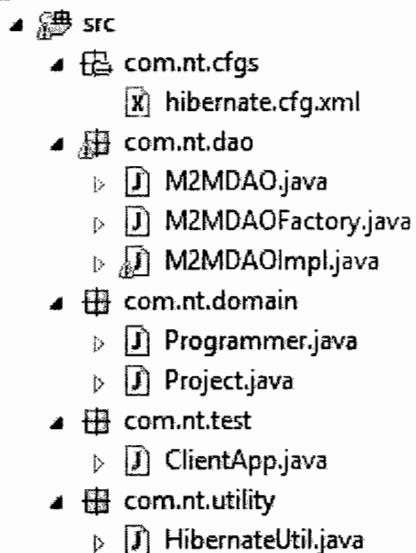
private List<Project> projects=new ArrayList<Project>();

```

programmer\_projects

<u>programmer_id</u>	<u>project_id</u>	<u>prgmr_proj_idx</u>
1	proj1	1
1	proj2	2
2	proj1	3
2	proj3	4

HBProj57(Anno-ManyToMany-Bi with CollectionId-ldbag)



hibernate.cfg.xml

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.Programmer"/>
        <mapping class="com.nt.domain.Project"/>
    </session-factory>
</hibernate-configuration>
```

Programmer.java

```
package com.nt.domain;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

import org.hibernate.annotations.CollectionId;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;

@Table(name="programmers")
@Entity
public class Programmer{
    @Id
    private int pid;
    private String pname;
    private double salary;

    @ManyToMany(targetEntity=Project.class,cascade=CascadeType.ALL)
    @JoinTable(name="programmers_projects",
```

```
joinColumns=@JoinColumn(name="programmer_id",referencedColumnName="pid"),
inverseJoinColumns=@JoinColumn(name="project_id",referencedColumnName="proid")
)
@GenericGenerator(name="gen1",strategy="increment")
@CollectionId(columns=@Column(name="prgmr_proj_idx"),
    type=@Type(type="int"),
    generator="gen1")
private List<Project> projects=new ArrayList<Project>();
public Programmer() {
System.out.println("Programmer:0-param consturctor");
}
public int getPid() {
return pid;
}
public void setPid(int pid) {
this.pid = pid;
}
public String getPname() {
return pname;
}
public void setPname(String pname) {
this.pname = pname;
}
public double getSalary() {
return salary;
}
public void setSalary(double salary) {
this.salary = salary;
}
public List<Project> getProjects() {
return projects;
}
public void setProjects(List<Project> projects) {
this.projects = projects;
}
@Override
public String toString() {
return "Programmer [pid=" + pid + ", pname=" + pname + ", salary=" + salary + "]";
}
}

/*
==== table1====
create table programmers
( pid number(5) primary key,
pname varchar2(20),
```

```
salary number(10)
);
=====table2=====
create table projects
( proid number(5) primary key,
proname varchar2(10)
);
=====table3=====
create table programmers_projects(
programmer_id number(5) references Programmers(pid),
project_id number(5) references Projects(proid),
primary key(programmer_id,project_id)
);

*/
```

#### Project.java

```
package com.nt.domain;

import java.util.HashSet;
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name="projects")
public class Project{
@Id
private int proid;
private String proname;
@ManyToMany(targetEntity=Programmer.class,
cascade=CascadeType.ALL,
mappedBy="projects")
private Set<Programmer> programmers=new HashSet<Programmer>();
public Project() {
System.out.println("Project: 0-param constructor");
}
public int getProid() {
return proid;
}
public void setProid(int proid) {
this.proid = proid;
}
public String getPrname() {
```

```
return proname;
}
public void setProname(String proname) {
this.proname = proname;
}
public Set<Programmer> getProgrammers() {
return programmers;
}
public void setProgrammers(Set<Programmer> programmers) {
this.programmers = programmers;
}
@Override
public String toString() {
return "Project [proid=" + proid + ", proname=" + proname + ", programmers=" + programmers +
"]";
}
}
```

**M2MDAO.java**

```
package com.nt.dao;

public interface M2MDAO {

    public void saveDataUsingProgrammer();
}
```

**M2MDAOImpl.java**

```
package com.nt.dao;

import java.util.List;
import java.util.Set;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.Programmer;
import com.nt.domain.Project;
import com.nt.utility.HibernateUtil;

public class M2MDAOImpl implements M2MDAO{

    @Override
    public void saveDataUsingProgrammer() {
        // get Session
        Session ses=HibernateUtil.getSession();
        //create parent and child objs
        Programmer prgmr1=new Programmer();
        prgmr1.setPid(1); prgmr1.setPname("raja"); prgmr1.setSalary(9000);
    }
}
```

```
Programmer prgmr2=new Programmer();
prgmr2.setPid(2); prgmr2.setPname("ravi"); prgmr2.setSalary(8000);

Programmer prgmr3=new Programmer();
prgmr3.setPid(3); prgmr3.setPname("jani"); prgmr3.setSalary(8000);

Project proj1=new Project();
proj1.setProid(1001); proj1.setProname("proj1");

Project proj2=new Project();
proj2.setProid(1002); proj2.setProname("proj2");

//set Projects to Programmer (childs to Parent)
prgmr1.getProjects().add(proj1);
prgmr1.getProjects().add(proj2);

prgmr2.getProjects().add(proj2);

prgmr3.getProjects().add(proj1);
prgmr3.getProjects().add(proj2);
//set Programmers to Projects (parents to childs)
proj1.getProgrammers().add(prgmr1);
proj1.getProgrammers().add(prgmr3);

proj2.getProgrammers().add(prgmr1);
proj2.getProgrammers().add(prgmr2);
proj2.getProgrammers().add(prgmr3);

//save objs (parent to child)
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(prgmr1);
    ses.save(prgmr2);
    ses.save(prgmr3);
    tx.commit();
    System.out.println("Objects are saved");
} //try
catch(Exception e){
    tx.rollback();
}
HibernateUtil.closeSession();
//saveDataUsingProgrammer

//class
```

**M2MDAOFactory.java**

```
package com.nt.dao;

public class M2MDAOFactory {
    public static M2MDAO getInstance(){
        return new M2MDAOImpl();
    }
}
```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```
package com.nt.test;

import com.nt.dao.M2MDAO;
import com.nt.dao.M2MDAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        M2MDAO dao=M2MDAOFactory.getInstance();
        dao.saveDataUsingProgrammer();

        //close SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```

**Annotation driven one-to-one association**

It can be implemented in two ways

- As one-to-one FK (using FK column)
- As one-to-one PK( without using FK column)

**one-to-one(FK)**

- @OneToOne : To build association.
- @JoinColumn: To specify FK column.

 HBProj58(Anno-One -One FK -Uni)

```

    ▲ src
      ▲ com.nt.cfgs
        X hibernate.cfg.xml
      ▲ com.nt.dao
        J O2ODAO.java
        J O2ODAOFactory.java
        J O2ODAOImpl.java
      ▲ com.nt.domain
        J License.java
        J Person.java
      ▲ com.nt.test
        J ClientApp.java
      ▲ com.nt.utility
        J HibernateUtil.java

```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Person"/>
    <mapping class="com.nt.domain.License"/>
  </session-factory>
</hibernate-configuration>

```

[Person.java](#)

```

package com.nt.domain;

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

import org.hibernate.annotations.GenericGenerator;

```

```

@Entity
@Table

```

```
public class Person{
    @Id
    @Column(name="person_id")
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int id;
    private String firstName;
    private String lastName;
    private byte age;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getFirstName() {
        return firstName;
    }
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public byte getAge() {
        return age;
    }
    public void setAge(byte age) {
        this.age = age;
    }
    @Override
    public String toString() {
        return "Person [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", age=" + age +
    "]";
    }
}
/*
create table Person(person_id number(5) primary key,
                   firstName varchar2(20),
                   lastName varchar2(20),
                   age number(5));
create table license(license_id number(5) primary key,
                     type varchar2(20),
                     valid_from date,
```

```
valid_to_date,  
license_holder_number(5) references Person(person_id));  
create sequence lic_seq start with 1000 increment by 1;  
*/
```

**License.java**

```
package com.nt.domain;  
  
import java.util.Date;  
import javax.persistence.CascadeType;  
import javax.persistence.Column;  
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.Id;  
import javax.persistence.JoinColumn;  
import javax.persistence.OneToOne;  
import javax.persistence.Table;  
import org.hibernate.annotations.GenericGenerator;  
import org.hibernate.annotations.Parameter;  
  
@Table  
@Entity  
public class License{  
    @Id  
    @Column(name="license_id")  
    @GenericGenerator(name="gen1",strategy="sequence",  
                      parameters=@Parameter(name="sequence",value="lic_seq")  
    )  
    @GeneratedValue(generator="gen1")  
    private int id;  
    private String type;  
    @Column(name="valid_from")  
    private Date dateFrom;  
    @Column(name="valid_to")  
    private Date dateTo;  
    @OneToOne(targetEntity=Person.class,  
               cascade=CascadeType.ALL)  
    @JoinColumn(name="license_holder",referencedColumnName="person_id")  
    private Person licenseHolder;  
  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }
```

```
public String getType() {
    return type;
}
public void setType(String type) {
    this.type = type;
}
public Date getDateFrom() {
    return dateFrom;
}
public void setDateFrom(Date dateFrom) {
    this.dateFrom = dateFrom;
}
public Date getDateTo() {
    return dateTo;
}
public void setDateTo(Date dateTo) {
    this.dateTo = dateTo;
}
public Person getLicenseHolder() {
    return licenseHolder;
}
public void setLicenseHolder(Person licenseHolder) {
    this.licenseHolder = licenseHolder;
}

@Override
public String toString() {
    return "License [id=" + id + ", type=" + type + ", dateFrom=" + dateFrom + ", dateTo=" + dateTo +
    "]";
}
```

### O2ODAO.java

```
package com.nt.dao;

public interface O2ODAO {

    public void saveDataUsingLicense();
    public void listDataUsingLicense();

}
```

### O2ODAOImpl.java

```
package com.nt.dao;

import java.util.Date;
import java.util.List;
```

```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.License;
import com.nt.domain.Person;
import com.nt.utility.HibernateUtil;

public class O2ODAOImpl implements O2ODAO{

    @Override
    public void saveDataUsingLicense() {
        // get Session
        Session ses=HibernateUtil.getSession();
        //prepare parent,child objs
        Person person1=new Person();
        person1.setFirstName("raja");
        person1.setLastName("Rao");
        person1.setAge((byte)30);

        License license=new License();
        license.setType("2-wheeler");
        license.setDateFrom(new Date());
        license.setDateTo(new Date(146,3,28));
        license.setLicenseHolder(person1);

        Person person2=new Person();
        person2.setFirstName("ravi");
        person2.setLastName("rathod");
        person2.setAge((byte)35);
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(person2);
            ses.save(license);
            tx.commit();
        } //try
        catch(Exception e){
            tx.rollback();
        }
        System.out.println("objs are saved");
    } //method

    @Override
    public void listDataUsingLicense() {
        // Get Session
    }
}
```

```

Session ses=HibernateUtil.getSession();
//prepare Query
Query query=ses.createQuery("from License");
List<License> list=query.list();
for(License license:list){
System.out.println("Child ---->" +license);
Person person=license.getLicenseHolder();
System.out.println("Parent---->" +person);
}//for
}//method

}//class

```

**O2ODAOFactory.java**

```

package com.nt.dao;

public class O2ODAOFactory {

public static O2ODAO getInstance(){
    return new O2ODAOImpl();
}

}

```

**HibernateUtil.java**

Same previous

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.O2ODAO;
import com.nt.dao.O2ODAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

public static void main(String[] args) {
//get DAO
O2ODAO dao=O2ODAOFactory.getInstance();
//dao.saveDataUsingLicense();
dao.listDataUsingLicense();

//close SessionFactory,Session
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class

```

**Note:**

In annotation driven environment, if you write **@OneToOne** with **@JoinColumn** then it builds OneToOne Fk association and if we write **@OneToOne** with **@PrimaryKeyJoinColumn** then it builds OneToOne PK association.

**one-to-one Pk association (annotation driven)**

- **@OneToOne** : To build the association
- **@PrimaryKeyJoinColumn**: To take pk value of one DATABASE table and to make it as the PK value of another DATABASE table. There is no need of FK column to build relationship.

HBProj59(Anna - One - One PK - Bi)

```

- src
  - com.nt.cfgs
    - hibernate.cfg.xml
  - com.nt.dao
    - O2ODAO.java
    - O2ODAOFactory.java
    - O2ODAOImpl.java
  - com.nt.domain
    - LibraryMembership.java
    - Student.java
  - com.nt.test
    - ClientApp.java
  - com.nt.utility
    - HibernateUtil.java
  
```

**hibernate.cfg.xml**

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <mapping class="com.nt.domain.Student"/>
    <mapping class="com.nt.domain.LibraryMembership"/>
  </session-factory>
</hibernate-configuration>
  
```

**Student.java**

```
package com.nt.domain;
```

```
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;

@Entity
@Table
public class Student{
    @Id
    @Column(name="sid")
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int id;
    private String name;
    private String address;
    @OneToOne(targetEntity=LibraryMembership.class,
               cascade=CascadeType.ALL,
               mappedBy="studentDetails")
    private LibraryMembership libraryDetails;
    public Student() {
        System.out.println("Student:0-param constructor");
    }
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getAddress() {
        return address;
    }
    public void setAddress(String address) {
        this.address = address;
    }
    public LibraryMembership getLibraryDetails() {
        return libraryDetails;
    }
}
```

```

public void setLibraryDetails(LibraryMembership libraryDetails) {
this.libraryDetails = libraryDetails;
}
@Override
public String toString() {
return "Student [id=" + id + ", name=" + name + ", address=" + address + "]";
}

/*SQL> create table student(sid number(5) primary key,name varchar2(20),
address varchar2(20));
SQL> create table lib_membership(lid number(5) primary key,doj date);
*/
}

```

***LibraryMembership.java***

```

package com.nt.domain;

import java.util.Date;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;

import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name="lib_membership")
public class LibraryMembership{
@Id
@Column(name="lid")
@GenericGenerator(name="gen1",
    strategy="foreign",
    parameters=@Parameter(name="property",value="studentDetails")
)
@GeneratedValue(generator="gen1")
private int id;

@Column(name="doj")
private Date joiningDate;

@OneToOne(targetEntity=Student.class,cascade=CascadeType.ALL)
@PrimaryKeyJoinColumn(name="lid", referencedColumnName="sid")

```

```

private Student studentDetails;
public LibraryMembership() {
System.out.println("LibraryMembership:0-param constructor");
}
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public Date getJoiningDate() {
return joiningDate;
}
public void setJoiningDate(Date joiningDate) {
this.joiningDate = joiningDate;
}
public Student getStudentDetails() {
return studentDetails;
}
public void setStudentDetails(Student studentDetails) {
this.studentDetails = studentDetails;
}
@Override
public String toString() {
return "LibraryMembership [id=" + id + ", joiningDate=" + joiningDate + "]";
}

}

```

**O2ODAO.java**

```

package com.nt.dao;

public interface O2ODAO {

public void saveDataUsingStudent();
public void saveDataUsingLibraryMembeship();
public void listDataUsingStudent();
public void listDataUsingLibraryMembership();

}

```

**O2ODAOImpl.java**

```

package com.nt.dao;

import java.util.Date;
import java.util.List;

```

```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.nt.domain.LibraryMembership;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class O2ODAOImpl implements O2ODAO{

    @Override
    public void saveDataUsingStudent() {
        // get Session
        Session ses=HibernateUtil.getSession();
        //prepare parent,child objs
        Student raja=new Student();
        raja.setName("raja"); raja.setAddress("hyd");

        LibraryMembership rajaLib=new LibraryMembership();
        rajaLib.setJoiningDate(new Date());
        //set parent to child and child to parent
        raja.setLibraryDetails(rajaLib);
        rajaLib.setStudentDetails(raja);
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            ses.save(raja);
            tx.commit();
        }catch(Exception e){
            tx.rollback();
        }
    }//method

    @Override
    public void saveDataUsingLibraryMembeship() {
        // Get Session
        Session ses=HibernateUtil.getSession();
        //prepare parent,child objs
        Student ravi=new Student();
        ravi.setName("ravi"); ravi.setAddress("hyd");

        LibraryMembership raviLib=new LibraryMembership();
        raviLib.setJoiningDate(new Date());
        //set parent to child and child to parent
        ravi.setLibraryDetails(raviLib);
        raviLib.setStudentDetails(ravi);
        //save objs( child to parent)
    }
}
```

```
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    ses.save(raviLib);
    tx.commit();
} //try
catch(Exception e){
    tx.rollback();
}

}//method

@Override
public void listDataUsingStudent() {
//get Session
Session ses=HibernateUtil.getSession();
//prepare Query
Query query=ses.createQuery("from Student");
List<Student> list=query.list();
for(Student st:list){
    System.out.println("parent---->" +st);
//get associated child
    LibraryMembership lib=st.getLibraryDetails();
    System.out.println("child---->" +lib);
}
}//fior
}//method

@Override
public void listDataUsingLibraryMembership() {
//get Session
Session ses=HibernateUtil.getSession();
//prepare Query
Query query=ses.createQuery("from LibraryMembership");
List<LibraryMembership> list=query.list();
for(LibraryMembership lib:list){
    System.out.println("child---->" +lib);
//get associated child
    Student st=lib.getStudentDetails();
    System.out.println("parent---->" +st);
}
}//fior
}//method

}//class
```

#### O2ODAOFactory.java

```
package com.nt.dao;

public class O2ODAOFactory {
```

```
public static O2ODAO getInstance(){
return new O2ODAOImpl();
}
```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

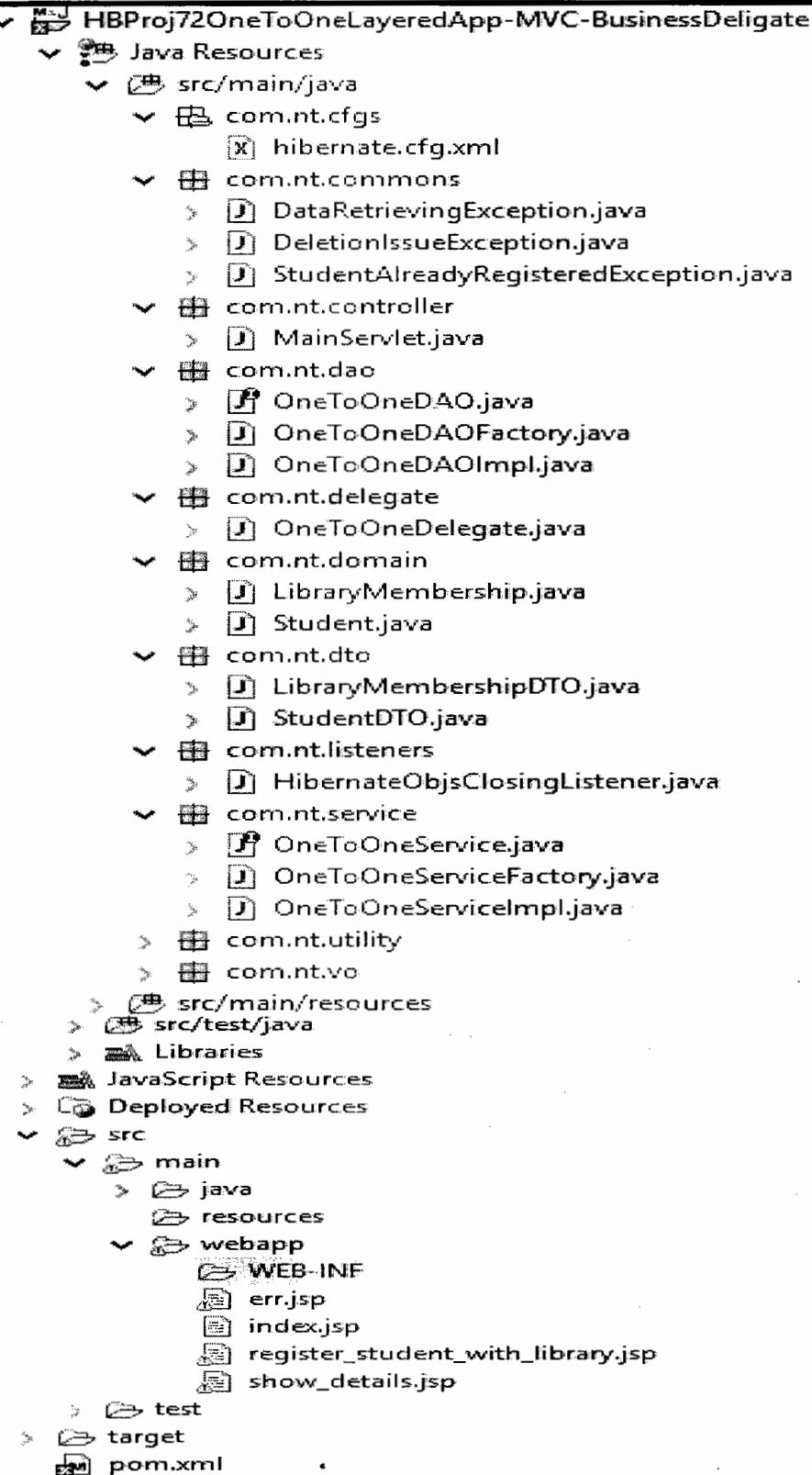
```
package com.nt.test;

import com.nt.dao.O2ODAO;
import com.nt.dao.O2ODAOFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        //get DAO
        O2ODAO dao=O2ODAOFactory.getInstance();
        //dao.saveDataUsingStudent();
        //dao.saveDataUsingLibraryMembeship();
        dao.listDataUsingStudent();
        dao.listDataUsingLibraryMembership();

        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }//main
}//class
```



**hibernate.cfg.xml:-**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="format_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.Student"/>
        <mapping class="com.nt.domain.LibraryMembership"/>
    </session-factory>

</hibernate-configuration>
```

**DataRetrievingException.java:-**

```
package com.nt.commons;

public class DataRetrievingException extends Exception {

    public DataRetrievingException(String msg) {
        super(msg);
    }

}
```

**DeletionIssueException.java:-**

```
package com.nt.commons;

public class DeletionIssueException extends Exception {

    public DeletionIssueException(String msg) {
        super(msg);
    }

}
```

**StudentAlreadyRegisteredException.java:-**

```
package com.nt.commons;

public class StudentAlreadyRegisteredException extends Exception {
```

```
public StudentAlreadyRegisteredException(String msg) {
    super(msg);
}

}
```

**MainServlet.java:-**

```
package com.nt.controller;

import java.io.IOException;
import java.text.ParseException;
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 com.nt.commons.DataRetrievingException;
import com.nt.commons.DeletionIssueException;
import com.nt.commons.StudentAlreadyRegisteredException;
import com.nt.delegate.OneToOneDelegate;
import com.nt.dto.StudentDTO;
import com.nt.vo.LibraryMembershipVO;
import com.nt.vo.StudentVO;

@WebServlet("/controller")
public class MainServlet extends HttpServlet {

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
        String param=null;
        OneToOneDelegate delegate=null;
        List<StudentDTO> listDTO=null;
        RequestDispatcher rd=null;
        StudentVO stVO=null;
        LibraryMembershipVO libVO=null;
        String sname=null, sadd=null, doj=null;
        String result=null;
        String sno=null;
        //get BusinessDelegate object
        delegate=new OneToOneDelegate();
        //read Addtional request parameter (param) value
        param=req.getParameter("param");
        if(param.equals("link1")){
            try{
                listDTO=delegate.getAllStudentsAndLibraryMembershipDetailsDelegate();
                req.setAttribute("listDTO", listDTO);
            }
        }
    }
}
```

```
rd=req.getRequestDispatcher("/show_details.jsp");
rd.forward(req,res);
}
catch(DataRetrievingException dre){
    req.setAttribute("errMsg",dre.getMessage());
    rd=req.getRequestDispatcher("/err.jsp");
    rd.forward(req,res);
}//catch
}//if
else if(param.equals("register")){
//read form data
sname=req.getParameter("sname");
sadd=req.getParameter("sadd");
doj=req.getParameter("doj");
//create StudentVO class having LibraryMembershipVO
class object
{
    libVO=new LibraryMembershipVO();
    libVO.setDoj(doj);
    stVO=new StudentVO();
    stVO.setSname(sname);
    stVO.setSadd(sadd);
    stVO.setLibraryDetails(libVO);
    libVO.setStudentDetails(stVO);
//use BusinessDelegate
try{
    result=delegate.registerDelegate(stVO);
    req.setAttribute("result",result);

listDTO=delegate.getAllStudentsAndLibraryMembershipDetailsDelegate();
req.setAttribute("listDTO",listDTO);

rd=req.getRequestDispatcher("/show_details.jsp");
rd.forward(req,res);
}
catch(StudentAlreadyRegisteredException sare){
    req.setAttribute("errMsg","Student
AlreadyRegistrered??");
    rd=req.getRequestDispatcher("/err.jsp");
    rd.forward(req,res);
}
catch(ParseException pe){
    req.setAttribute("errMsg","Date conversion
problem");
    rd=req.getRequestDispatcher("/err.jsp");
    rd.forward(req,res);
}
catch(Exception e){
    req.setAttribute("errMsg","Internal problem");
    rd=req.getRequestDispatcher("/err.jsp");
}
```

```

        rd.forward(req,res);
    }
} //else if
else if(param.equals("link2")){
    //read student number
    sno=req.getParameter("sno");
    try{
        //use BusinessDelegate
        result=delegate.deleteStudentWithLibraryDelegate(sno);
        req.setAttribute("result",result);

listDTO=delegate.getAllStudentsAndLibraryMembershipDetailsDelegate();
        req.setAttribute("listDTO",listDTO);
        //forward request
        rd=req.getRequestDispatcher("/show_details.jsp");
        rd.forward(req,res);
    } //try
    catch(DeletionIssueException die){
        req.setAttribute("errMsg","Student Not
Deleted");
        rd=req.getRequestDispatcher("/err.jsp");
        rd.forward(req,res);
    }
    catch(DataRetrievingException dre){
        req.setAttribute("errMsg",dre.getMessage());
        rd=req.getRequestDispatcher("/err.jsp");
        rd.forward(req,res);
    } //catch
    catch(Exception e){
        req.setAttribute("errMsg","Internal problem");
        rd=req.getRequestDispatcher("/err.jsp");
        rd.forward(req,res);
    } //catch

}
} // doGet(-,-)

@Override
public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    doGet(req, res);
} // doPost(-,-)

} // class

```

**OneToOneDAO.java:-**

```

package com.nt.dao;

import java.util.List;

```

```
import com.nt.domain.Student;

public interface OneToOneDAO {
    public List<Student>
getAllStudentsAndTheirLibraryMembershipDetails();
    public int insertStudentWithLibrary(Student student);
    public int deleteStudentWithLibrary(int no);
}
```

**OneToOneDAOFactory.java:-**

```
package com.nt.dao;

public class OneToOneDAOFactory {
    public static OneToOneDAO getInstance(){
        return new OneToOneDAOImpl();
    }
}
```

**OneToOneDAOImpl.java:-**

```
package com.nt.dao;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class OneToOneDAOImpl implements OneToOneDAO {
    private static final String
HQL_GET_ALL_STUDENTS_AND_LIBRARYDETAILS="from Student";

    @Override
    public List<Student>
getAllStudentsAndTheirLibraryMembershipDetails() {
        Session ses=null;
        Query query=null;
        List<Student> listDomains=null;
        //get Session
        ses=HibernateUtil.getSession();
        //prepare Query

        query=ses.createQuery(HQL_GET_ALL_STUDENTS_AND_LIBRARYDETAILS);
        //execute the Query
        listDomains=query.list();
        return listDomains;
    }

    @Override
```

```
public int insertStudentWithLibrary(Student student) {
    Session ses=null;
    Transaction tx=null;
    int sid=0;
    //get Session
    ses=HibernateUtil.getSession();
    //save object
    try{
        tx=ses.beginTransaction();
        sid=(Integer)ses.save(student);
        tx.commit();
    }
    catch(Exception e){
        tx.rollback();
    }
    return sid;
}//method

@Override
public int deleteStudentWithLibrary(int no) {
    Session ses=null;
    Student stud=null;
    Transaction tx=null;
    // get Session
    ses=HibernateUtil.getSession();
    //load the object
    stud=ses.get(Student.class,no);
    //delete the object
    try{
        tx=ses.beginTransaction();
        ses.delete(stud);
        tx.commit();
        return 1;
    }
    catch(Exception e){
        tx.rollback();
        return 0;
    }
}//method
}//class
```

OneToOneDelegate.java:-

```
package com.nt.delegate;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.List;
import com.nt.commons.DataRetrievingException;
import com.nt.commons.DeletionIssueException;
import com.nt.commons.StudentAlreadyRegisteredException;
```

```
import com.nt.dto.LibraryMembershipDTO;
import com.nt.dto.StudentDTO;
import com.nt.service.OneToOneService;
import com.nt.service.OneToOneServiceFactory;
import com.nt.vo.LibraryMembershipVO;
import com.nt.vo.StudentVO;

public class OneToOneDelegate {

    public List<StudentDTO>
    getAllStudentsAndLibraryMembershipDetailsDelegate()throws
    DataRetrievingException{
        OneToOneService service=null;
        List<StudentDTO> listDTO=null;
        //get SErvie class object
        service=OneToOneServiceFactory.getInstance();
        try{

            listDTO=service.findAllStudentsAndLibraryMemebershipDetails();
        }
        catch(Exception e){
            throw new DataRetrievingException(e.getMessage());
        }
        return listDTO;
    }//method
    public String registerDelegate(StudentVO stVO) throws
    StudentAlreadyRegisteredException,ParseException{
        OneToOneService service=null;
        String result=null;
        LibraryMembershipVO libVO=null;
        StudentDTO stDTO=null;
        LibraryMembershipDTO libDTO=null;
        //convert StVO class object stDTO class object
        libVO=stVO.getLibraryDetails();
        libDTO=new LibraryMembershipDTO();
        libDTO.setDoj(new SimpleDateFormat("MM-dd-
yyyy").parse(libVO.getDoj()));
        stDTO=new StudentDTO();
        stDTO.setSname(stVO.getSname());
        stDTO.setSadd(stVO.getSadd());
        stDTO.setLibraryDetails(libDTO);
        libDTO.setStudentDetails(stDTO);
        //get Service class object
        service=OneToOneServiceFactory.getInstance();
        try{
            result=service.register(stDTO);
        }
        catch(Exception e){
            e.printStackTrace();
            throw new
```

```

StudentAlreadyRegisteredException(e.getMessage());
    }
    return result;
}//method
public String deleteStudentWithLibraryDelegate(String no) throws
DeletionIssueException{
    OneToOneService service=null;
    String result;
    int sno=0;
    //get Service class
    service=OneToOneServiceFactory.getInstance();
    //convert Student number to int
    sno=Integer.parseInt(no);
    //use Service class
    try{
        result=service.removeStudentWithLibrary(sno);
    }
    catch(Exception e){
        throw new DeletionIssueException(e.getMessage());
    }
    return result;
}//method
}//class

```

**LibraryMembership.java:-**

```

package com.nt.domain;

import java.util.Date;
import javax.persistence.CascadeType;
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 org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;

@Entity
@Table(name="Lib_Membership")
public class LibraryMembership {
    @Id
    @GenericGenerator(name="gen1", strategy="foreign",
                      parameters=@Parameter(name="property",
                                             value="studentDetails"))
    @GeneratedValue(generator="gen1")
    private int lid;

    private Date doj;
}

```

```
@OneToOne(targetEntity=Student.class,cascade=CascadeType.ALL,
           fetch=FetchType.LAZY)
@PrimaryKeyJoinColumn(name="lid",referencedColumnName="sno")
private Student studentDetails;
public int getLid() {
    return lid;
}
public void setLid(int lid) {
    this.lid = lid;
}
public Date getDoj() {
    return doj;
}
public void setDoj(Date doj) {
    this.doj = doj;
}
public Student getStudentDetails() {
    return studentDetails;
}
public void setStudentDetails(Student studentDetails) {
    this.studentDetails = studentDetails;
}
@Override
public String toString() {
    return "LibraryMembership [lid=" + lid + ", doj=" + doj + "]";
}
}
```

**Student.java:-**

```
package com.nt.domain;
import javax.persistence.CascadeType;
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;
@Entity
@Table(name="student_tab")
public class Student {
    @Id
    @GenericGenerator(name="gen1",strategy="increment")
    @GeneratedValue(generator="gen1")
    private int sno;
    private String sname;
    private String sadd;
    @OneToOne(targetEntity=LibraryMembership.class,
               cascade=CascadeType.ALL,fetch=FetchType.LAZY,
```

```
        mappedBy="studentDetails")
private LibraryMembership libraryDetails;
public int getSno() {
    return sno;
}
public void setSno(int sno) {
    this.sno = sno;
}
public String getSname() {
    return sname;
}
public void setSname(String sname) {
    this.sname = sname;
}
public String getSadd() {
    return sadd;
}
public void setSadd(String sadd) {
    this.sadd = sadd;
}
public LibraryMembership getLibraryDetails() {
    return libraryDetails;
}
public void setLibraryDetails(LibraryMembership libraryDetails) {
    this.libraryDetails = libraryDetails;
}
@Override
public String toString() {
    return "Student [sno=" + sno + ", sname=" + sname + ", sadd="
+ sadd + "]";
}
}//class
```

**LibraryMembershipDTO.java:-**

```
package com.nt.dto;
import java.util.Date;
import com.nt.domain.Student;

public class LibraryMembershipDTO {
    private int lid;
    private Date doj;
    private StudentDTO studentDetails;
    public int getLid() {
        return lid;
    }
    public void setLid(int lid) {
        this.lid = lid;
    }
    public Date getDoj() {
        return doj;
    }
}
```

```
}

public void setDoj(Date doj) {
    this.doj = doj;
}

public StudentDTO getStudentDetails() {
    return studentDetails;
}

public void setStudentDetails(StudentDTO studentDetails) {
    this.studentDetails = studentDetails;
}

}
```

**StudentDTO.java:-**

```
package com.nt.dto;

import com.nt.domain.LibraryMembership;

public class StudentDTO {
    private int sno;
    private String sname;
    private String sadd;

    private LibraryMembershipDTO libraryDetails;
    public int getSno() {
        return sno;
    }
    public void setSno(int sno) {
        this.sno = sno;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
    public LibraryMembershipDTO getLibraryDetails() {
        return libraryDetails;
    }
    public void setLibraryDetails(LibraryMembershipDTO libraryDetails) {
        this.libraryDetails = libraryDetails;
    }
}
```

HibernateObjClosingListener.java:-

```
package com.nt.listeners;

import javax.servlet.ServletContainerInitializer;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;
import com.nt.utility.HibernateUtil;
@WebListener
public class HibernateObjClosingListener implements
ServletContextListener{

    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        System.out.println("contextDestroyed(-)");
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
    @Override
    public void contextInitialized(ServletContextEvent sce) {
        System.out.println("contextInitialized(-)");

    }
}
```

OneToOneService.java:-

```
package com.nt.service;

import java.util.List;
import com.nt.dto.StudentDTO;

public interface OneToOneService {
    public List<StudentDTO>
findAllStudentsAndLibraryMemebershipDetails();
    public String register(StudentDTO dto);
    public String removeStudentWithLibrary(int no);

}
```

OneToOneServiceFactory.java:-

```
package com.nt.service;
public class OneToOneServiceFactory {

    public static OneToOneService getInstance(){
        return new OneToOneServiceImpl();
    }
}
```

**OneToOneServiceImpl.java:-**

```
package com.nt.service;

import java.util.ArrayList;
import java.util.List;
import com.nt.dao.OneToOneDAO;
import com.nt.dao.OneToOneDAOFactory;
import com.nt.domain.LibraryMembership;
import com.nt.domain.Student;
import com.nt.dto.LibraryMembershipDTO;
import com.nt.dto.StudentDTO;

public class OneToOneServiceImpl implements OneToOneService {

    @Override
    public List<StudentDTO>
    findAllStudentsAndLibraryMemebershipDetails() {
        OneToOneDAO dao=null;
        List<Student> listDomains=null;
        LibraryMembership lib=null;
        List<StudentDTO> listDTO=null;
        StudentDTO studDTO=null;
        LibraryMembershipDTO libDTO=null;
        //get DAO
        dao=OneToOneDAOFactory.getInstance();
        //use DAO

        listDomains=dao.getAllStudentsAndTheirLibraryMembershipDetails();
        //Convert ListDomains to ListDTO
        listDTO=new ArrayList();
        for(Student st:listDomains){
            studDTO=new StudentDTO();
            studDTO.setSno(st.getSno());
            studDTO.setSname(st.getSname());
            studDTO.setSadd(st.getSadd());
            lib=st.getLibraryDetails();
            libDTO=new LibraryMembershipDTO();
            libDTO.setLid(lib.getLid());
            libDTO.setDoj(lib.getDoj());
            studDTO.setLibraryDetails(libDTO);
            libDTO.setStudentDetails(studDTO);
            listDTO.add(studDTO);
        }//for
        return listDTO;
    }//method

    @Override
    public String register(StudentDTO dto) {
        OneToOneDAO dao=null;
        Student st=null;
```

```
LibraryMembership lib=null;
LibraryMembershipDTO libDTO=null;
int sno=0;
//get DAO
dao=OneToOneDAOFactory.getInstance();
//Convert DTO to Domain class object
st=new Student();
st.setSname(dto.getSname());
st.setSadd(dto.getSadd());
libDTO=dto.getLibraryDetails();
lib=new LibraryMembership();
lib.setDoj(libDTO.getDoj());
st.setLibraryDetails(lib);
lib.setStudentDetails(st);
//use DAO
sno=dao.insertStudentWithLibrary(st);
return "Student is registered with sno="+sno;
}//method

@Override
public String removeStudentWithLibrary(int no) {
    OneToOneDAO dao=null;
    int count=0;
    //get DAO
    dao=OneToOneDAOFactory.getInstance();
    //use DAO
    count=dao.deleteStudentWithLibrary(no);
    if(count==0)
        return "Student with Library is not deleted";
    else
        return "Student with Library is deleted";
}//method
}//class
```

**HibernateUtil.java:-**

```
package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").
buildSessionFactory();
        System.out.println("SessionFactory
```

```
hashCode::"+factory.hashCode());
}
public static Session getSession(){
    Session ses=null;
    ses=threadLocal.get();
    if(ses==null){
        ses=factory.openSession();
        threadLocal.set(ses);
    }
    System.out.println("Session obj hashCode::"+ses.hashCode());
    return ses;
}//getSession()

public static void closeSession(){
    Session ses=null;

    ses=threadLocal.get();
    if(ses!=null){
        ses.close();
        threadLocal.remove();
    }
}

public static void closeSessionFactory(){
    factory.close();
}

}//class
```

**LibraryMembershipVO.java:-**

```
package com.nt.vo;
import java.util.Date;
import com.nt.domain.Student;
public class LibraryMembershipVO {
    private String doj;
    private StudentVO studentDetails;
    public String getDoj() {
        return doj;
    }
    public void setDoj(String doj) {
        this.doj = doj;
    }
    public StudentVO getStudentDetails() {
        return studentDetails;
    }
    public void setStudentDetails(StudentVO studentDetails) {
        this.studentDetails = studentDetails;
    }
}
```

**StudentVO.java:-**

```

package com.nt.vo;

public class StudentVO {
    private String sname;
    private String sadd;
    private LibraryMembershipVO libraryDetails;
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSadd() {
        return sadd;
    }
    public void setSadd(String sadd) {
        this.sadd = sadd;
    }
    public LibraryMembershipVO getLibraryDetails() {
        return libraryDetails;
    }
    public void setLibraryDetails(LibraryMembershipVO libraryDetails) {
        this.libraryDetails = libraryDetails;
    }
}

```

**err.jsp:-**

```

<%@page isELIgnored="false" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:if test="${errMsg ne null }">
    ${errMsg}
</c:if>

```

**index.jsp:-**

```

<a href="controller?param=Link1">GetAllStudentsAndLibraryMembershipDetails</a>

```

**register student with library.jsp:-**

```

<h1 style="text-align:center"> Provide Student and Library Details</h1>

<form action="controller" method="post">
    <h3><p> Student Details </p></h3>
    name :: <input type="text" name="sname"><br>

```

```
Address:: <input type="text" name="sadd"><br>
<h3><p> Library Details </p></h3>
Date Of Joining <input type="text" name="doj"/> (MM-dd-yyyy) <br>
<input type="submit" value="register" name="param">
</form>
```

show details.jsp:-

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<c:choose>
<c:when test="${!empty listDTO}">
    <c:forEach var="st" items="${listDTO}">
        <h2 style="color:red"> ${st.sno} &ampnbsp&ampnbsp&ampnbsp
${st.sname} &ampnbsp&ampnbsp&ampnbsp ${st.sadd} &ampnbsp&ampnbsp&ampnbsp<a
href="controller?param=Link2&sno=${st.sno}">Delete student with Library
</a> </h3>
        <h5
style="color:blue">${st.libraryDetails.lid}&ampnbsp&ampnbsp&ampnbsp
${st.libraryDetails.doj} &ampnbsp&ampnbsp <a
href="controller?param=Link3">Delete only Library </a></h4>
    </c:forEach>
        <a href="register_student_with_Library.jsp">Add Student with
LibraryMembership</a>
    </c:when>
    <c:otherwise>
        <h1 style="color:red">Records not found </h1>
    </c:otherwise>
</c:choose>
<br><br><br>
<c:if test="${result ne null }">
    <h1 style='color:cyan'>${result} </h1>
</c:if>
```

pom.xml:-

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>nit</groupId>
    <artifactId>HBProj20LayeredWebApp</artifactId>
    <packaging>war</packaging>
    <version>0.0.1-SNAPSHOT</version>
    <name>HBProj20LayeredWebApp Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>
```

```

<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-core</artifactId>
    <version>5.1.0.Final</version>
</dependency>
<dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.1.0</version>
</dependency>
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
    <version>1.2</version>
</dependency>

<dependency>
    <groupId>taglibs</groupId>
    <artifactId>standard</artifactId>
    <version>1.1.2</version>
</dependency>

<!--
https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl -->

</dependencies>
<build>
    <finalName>HBProj20LayeredWebApp</finalName>
</build>
</project>

```

#### Q. What is different between **FetchMode** and **FetchType**?

Ans:

**FetchType:** Talks about whether associated object should be loaded lazily or eagerly along with main object. The possible fetch types are **EAGER, LAZY**.

**FetchMode:** Talks about number of select queries that should be generated while loading child objects along with parent objects, the possible FetchModes are **SELECT, SUBSELECT, JOIN**.

- In xml file we use **lazy** attribute to specify fetch type and fetch attribute to specify **FetchMode**.  
→ In annotations we use **fetch** attribute/parameter of **@XxxToYyy annotations** to specify **FetchType**. Similarly we use Hibernate supplied **@Fetch annotation** to specify **FetchMode**.

```
@Fetch(value=FetchMode.SELECT/SUBSELECT/JOIN)
private List<PhoneNumber>phones;
```

### Caching/Buffering

Cache or buffer is a temporary memory that holds the data for temporary period. In client-server environment the cache at client side remembers sever side data and uses it across the multiple same requests and reduces the network round trips between client and server. To get good benefit of cache, it should be emptied at regular intervals.

### Hibernate supports two level of caching:

#### 1. Level1/first level/L1 cache:

- It is built-in cache associated with session object.
- It is one per session object.

#### Two benefits of Level1/First Level cache:

1. Keeps loaded objects from DATABASE software and uses it across the multiple same requests.
2. Keeps track of the changes happened on the objects in the coarse of Transaction and generates single SQL Query to DATABASE software reflecting all the changes when Transaction is committed.

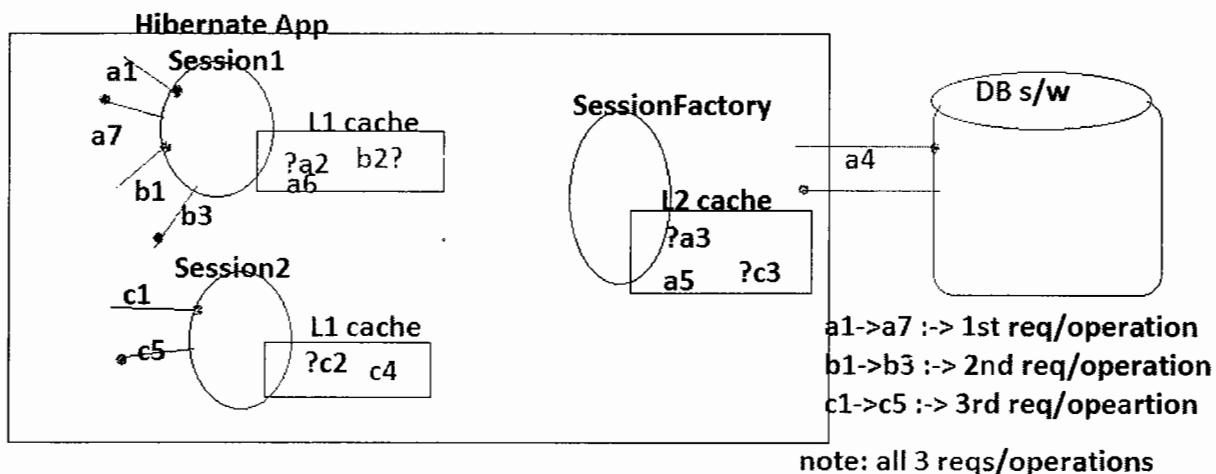
#### 2. Level2/Second Level/L2 cache:

- It is associated with SessionFactory object i.e. it is One per SessionFactory object .
- It is configurable cache, there are multiple second level cache provider softwares, we need to configure one of them to enable Second level/L2 cache in the application.

#### Flow in the caches

We ask hibernate framework to load objects from DATABASE software → Checks in Level1 cache of current session object to the load the object, if available takes the object → otherwise checks in Level2 cache to get object, if available object it takes the object from Level2 cache → otherwise Application interacts with Database software and load sobject/record from Database software → Keeps the object in Level2 cache → Keeps the object in Level1 cache → Returns the object to client application.

=>Hibernate App first searches required obj in L1 cache, if not available then it searches in L2 cache, if there also not available it loads from DB s/w and keeps in L2 cache ,L1 cache before using in the App.



=>Hibernate Caches can maintain domain class objs/  
collections having domain class objs

### There are multiple second level caching software

- EH cache
- OS cache
- Swaram cache
- JbossTree cache
- Tangosal cache(commercial) and etc...

#### **EHCache**

- Fast, lightweight, and easy-to-use in-process cache
- Supports read-only and read/write caching, and memory- and disk-based caching.
- However, it does not support clustering

#### **OSCache**

- Part of a larger package, which also provides caching functionalities for JSP pages or arbitrary objects.
- It is a powerful and flexible package, which, like EHCache, supports read-only and read/write caching, and memory- and disk-based caching.
- It also provides basic support for clustering via either JavaGroups or JMS

#### **SwarmCache**

- Is a simple cluster-based caching solution based on JavaGroups.
- Supports read-only or nonstrict read/write caching (the next section explains this term).
- This type of cache is appropriate for applications that typically have many more read operations than write operations.

#### **JBoss TreeCache**

- Is a powerful replicated (synchronous or asynchronous) and transactional cache.
- Use this solution if you really need a true transaction-capable caching architecture

Cache	Provider Class	Type	Cluster Safe	Query Cache Supported
EHCACHE	net.sf.hibernate.ehcache.hibernate.Provider	Memory, disk	No	Yes
OSCache	net.sf.hibernate.cache.OSCacheProvider	Memory, disk	No	Yes
SwarmCache	net.sf.hibernate.cache.SwarmCacheProvider	Clustered (IP multicast)	Yes (clustered invalidation)	No
TreeCache	net.sf.hibernate.cache.TreeCacheProvider	Clustered (IP multicast), transactional	Yes (replication)	No

### Caching Strategies for Second-level Cache

- Read-only
- Read/write
- Nonstrict read/write
- Transactional

#### Caching Strategy: Read-only

- Useful for data that is read frequently but never updated.
- Simplest and best-performing cache strategy.

<hibernate-mapping>

```

<class name="Country" table="COUNTRY" dynamic-update="true">
    <meta attribute="implement-equals">true</meta>
    <cache usage="read-only"/>
    <id name="id" type="long" unsaved-value="null" >
        <column name="cn_id" not-null="true"/>
        <generator class="increment"/>
    </id>
    <property column="cn_code" name="code" type="string"/>
    <property column="cn_name" name="name" type="string"/>
    <set name="airports" >
        <cache usage="read-only"/>
        <key column="cn_id"/>
        <one-to-many class="Airport"/>
    </set>
</class>
</hibernate-mapping>

```

#### Caching Strategy: Read/Write

- Appropriate if your data needs to be updated.
- They carry more overhead than read-only caches.
- In non-JTA environments, each transaction should be completed when `Session.close()` or `Session.disconnect()` is called.

<hibernate-mapping>

```

<class name="Employee" table="EMPLOYEE" dynamic-update="true">
    <meta attribute="implement-equals">true</meta>
    <cache usage="read-write"/>

    <id name="id" type="long" unsaved-value="null" >
        <column name="emp_id" not-null="true"/>
        <generator class="increment"/>
    </id>

    <property column="emp_surname" name="surname" type="string"/>
    <property column="emp_firstname" name="firstname" type="string"/>

    <many-to-one name="country" column="cn_id"
        class="com.wakaleo.articles.caching.businessobjects.Country"
        not-null="true" />

    <set name="languages" table="EMPLOYEE_SPEAKS_LANGUAGE" lazy="false">
        <cache usage="read-write"/>
        <key column="emp_id"/>
        <many-to-many column="lan_id" class="Language"/>
    </set>
</class>
</hibernate-mapping>

```

#### Caching Strategy: Nonstrict Cache

- Does not guarantee that two transactions won't simultaneously modify the same data.
- Therefore, it may be most appropriate for data that is read often but only occasionally modified.

#### Caching Strategy: Transactional

- A fully transactional cache that may be used
- only in a JTA environment

#### Procedure to work with second level cache by taking EHCache implementation

- a) Keep any application ready.
- b) Add EHCache related jar files to the build path of project.
- c) All 3 jar files of <hibernate\_home>\lib\optional\ehcache folder.
- d) Configure the cache provider class enable second level cache in hibernate.cfg.xml file

```

<property
    name="cache.region.factory_class">org.hibernate.cache.EhCacheRegionFactory</pro
    perty>
    <property name="cache.use_second_level_cache">true</property>
    <property name="cache.use_query_cache">true</property>

```

- e) Develop xml file(ehcache.xml) having ehcache configurations

```

<ehcache>

    <diskStore path="java.io.tmpdir"/>

    <defaultCache

        maxElementsInMemory="100"

        eternal="false"

        timeToIdleSeconds="5"

        timeToLiveSeconds="20"

        overflowToDisk="true"

    />

</ehcache>

```

**Note:**

→ Disk caching mean once the Level2 cache memory of RAM is filled up then it will use hard disk for the caching.

- f) Configure the above xml in hibernate.cfg.xml file

```

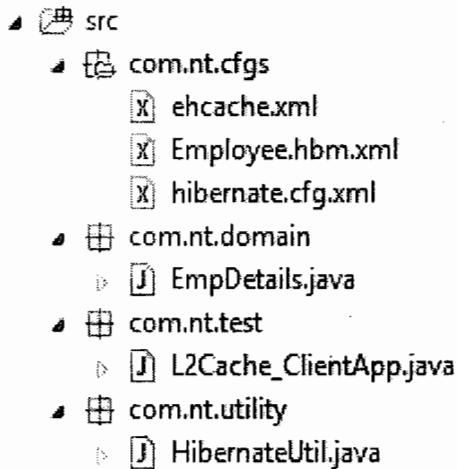
<property
    name="net.sf.ehcache.configurationResourceName">/com/nt/cfgs/ehcache.xml</proper
    ty>

```

- g) Specify the caching strategy in mapping file(before <id> tag)

```
<cache usage="read-only"/>
```

HBProj60(SecondLevel Cache-EH Cache & QueryCache)



**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <property
            name="cache.region.factory_class">org.hibernate.cache.EhCacheRegionFactory</property>
        <property name="cache.use_second_level_cache">true</property>
        <property name="cache.use_query_cache">true</property>
        <property
            name="net.sf.ehcache.configurationResourceName">/com/nt/cfgs/ehcache.xml</property>

        <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
    </session-factory>

</hibernate-configuration>
```

**EmpDetails.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
        <cache usage="read-only"/>
        <id name="no" column="ENO"><!-- Singular Id field -->
            <generator class="native"/>
        </id>
        <property name="fname" >
            <column name="FIRSTNAME"/>
        </property>
        <property name="lname" >
            <column name="LASTNAME"/>
        </property>
        <property name="mail" >
            <column name="EMAIL"/>
        </property>
    </class>
</hibernate-mapping>
```

**ehcache.xml**

```
<ehcache>
<diskStore path="java.io.tmpdir"/>
<defaultCache
    maxElementsInMemory="100"
    eternal="false"
    timeToIdleSeconds="5"
    timeToLiveSeconds="20"
    overflowToDisk="true"
/>

</ehcache>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
    //domain class properties
    private int no;
    private String fname;
    private String lname;
    private String mail;
    public EmpDetails(){
        System.out.println("EmpDetails:0-param constructor");
    }

    public int getNo() {
        return no;
    }
    public void setNo(int no) {
        this.no = no;
    }
    public String getFname() {
        return fname;
    }
    public void setFname(String fname) {
        this.fname = fname;
    }
    public String getLname() {
        return lname;
    }
    public void setLname(String lname) {
```

```

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}
}

```

**HibernateUtil.java**

Same as previous

**L2Cache\_ClientApp.java**

```

package com.nt.test;
import org.hibernate.Session;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;
public class L2Cache_ClientApp {

public static void main(String[] args) throws Exception{
//Get HB Session obj
Session ses=HibernateUtil.getSession();

// Loads from DATABASE software and keeps in Level2,Level1 cache
EmpDetails ed1=(EmpDetails)ses.get(EmpDetails.class,101);
System.out.println(ed1);

//Loads from Level1 cache
EmpDetails ed2=(EmpDetails)ses.get(EmpDetails.class,101);
System.out.println(ed2);

ses.clear(); //removes obj from Level1 cache
//Loads from Level2 cache and also keeps in L1cache
EmpDetails ed3=(EmpDetails)ses.get(EmpDetails.class,101);
System.out.println(ed3);
ses.clear(); // removes obj from Level1 cache
Thread.sleep(10000);
//Removes obj from Level2 cache becoz of idle time out period

// Loads from DATABASE software and keeps in Level2,Level1 cache

```

```
EmpDetails ed4=(EmpDetails)ses.get(EmpDetails.class,101);
System.out.println(ed4);

//close Session,sessionfactory
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}//main
}//class
```

## Query Cache

- It is part of second level cache/Level2 cache and it maintains domain class objects that comes because of HQL query execution.
- When same HQL query is executed for multiple times, this **Query cache** maintains the objects and uses the objects across the multiple same executions to reduce network rounds trips between java application and Database software.

### Example on Query Cache

By default the HQL query result doesn't participant in caching to make them participating in caching, we need to go for **Query Cache**.

- a) Keep previous application ready on which Level2/SecondLevel is cache enabled
- b) Enable Query cache through hibernate.cfg.xml file  
`<property name="cache.use_query_cache">true</property>`
- c) Write the following code in client app to keep HQL Query results in Query cache

#### QueryCacheClient.java

```
package com.nt.test;

import java.util.List;
import org.hibernate.CacheMode;
import org.hibernate.Query;
import org.hibernate.Session;
import com.nt.domain.EmpDetails;
import com.nt.utility.HibernateUtil;

public class QueryCacheClient {

    public static void main(String[] args) {
        // get Session
        Session ses=HibernateUtil.getSession();

        Query query=ses.createQuery("from EmpDetails where eno>=:no");
        //enable query cache
        query.setCacheable(true);
```

```
query.setCacheMode(CacheMode.NORMAL);
query.setInteger("no",100);
//execute the Query
List<EmpDetails> list=query.list(); // gets from DATABASE software and keeps in Query cache
//process the Result
for(EmpDetails ed:list){
System.out.println(ed);
}
//execute Query
List<EmpDetails> list1=query.list(); // gets from Query cache.
//process the Result
for(EmpDetails ed:list1){
System.out.println(ed);
}//for

//close Session and session factory
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();
}
```

#### To control second level cache method:

- **factory.close()** : Level2/Second Level cache will be cleaned up.
- **factory.evict(EmpDetais.class)**: removes EmpDetails class objects from Level2/Second Level cache.
- **factory.evictCollection("phones")** : Remove "phones" related collections from Level2 cache.
- **factory.evictQueries()**: Remove HQL Queries results from Query cache from default region.
- **factory.evictQueries("region1")**: Removes HQL results from Query cache region whose name is **region1**.

evict() and evictXxx() methods are deprecated. So use Cache.evictXxx() methods they are:

- cache.evictEntityRegion(EmpDetails.class)
- cache.evictEntityCollectionRegion("phones")
- cache.evictDefaultQueryRegion()
- cache.evictQueryRegion("region1");

#### To get Cache Object

```
Cache cache =factory.getCache();
```

### Transaction Management:

- The process of combining related operations into single unit and executing them by applying **do everything or nothing principle** is called Transaction management.
- It is recommended to enable Transaction management on sensitive operations like transfer money, employee registration in both HR, Finance Database table and etc...
- Transfer money operation is combination of two tasks:
  - a) Withdraw amount from source account
  - b) Deposit amount into destination accountIf completed, both operations should be completed or both operations should be failed (do everything or nothing)

#### Transaction management is all about 3 operations:

Begin transaction

->Operation 1

->Operation 2

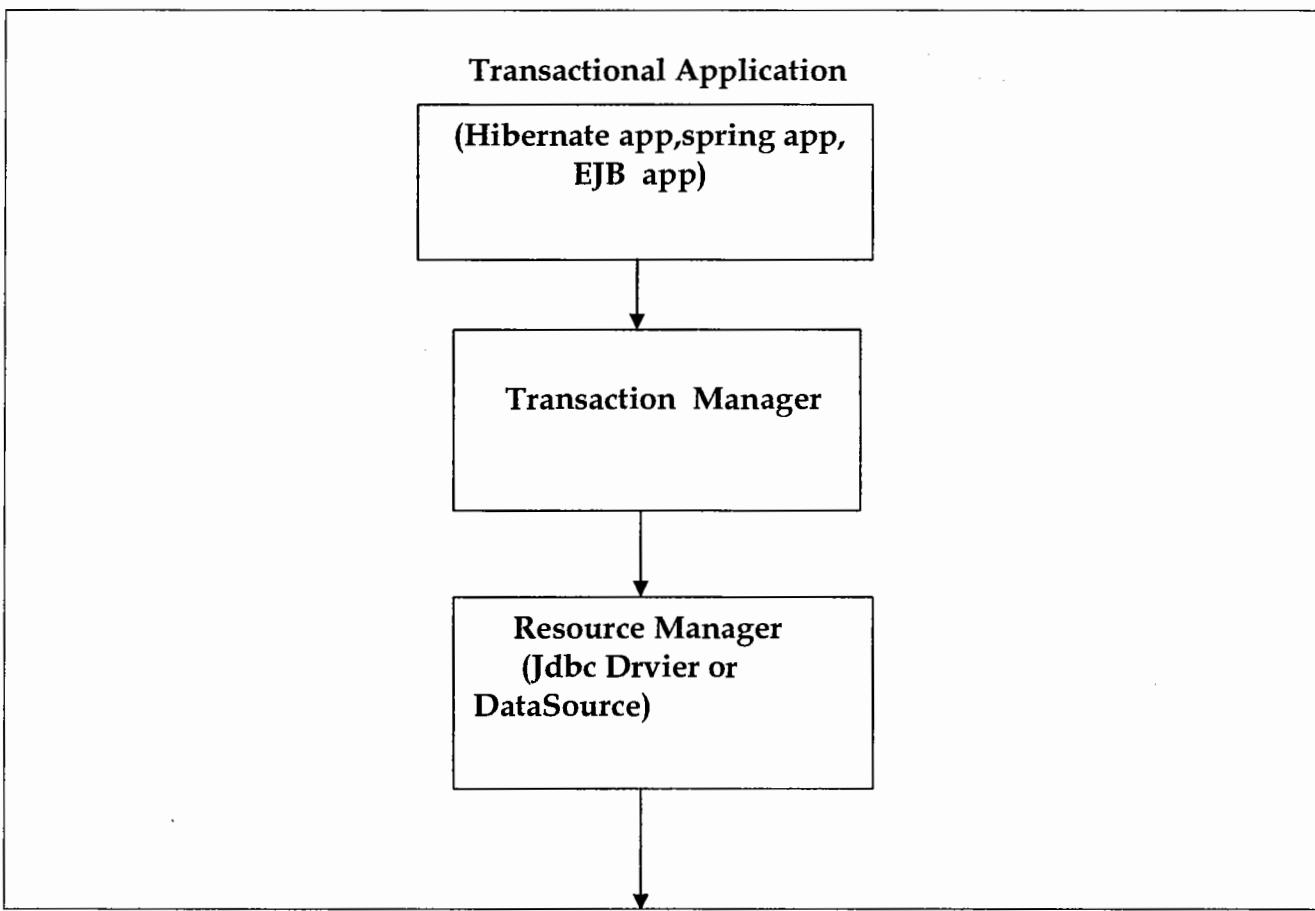
.....

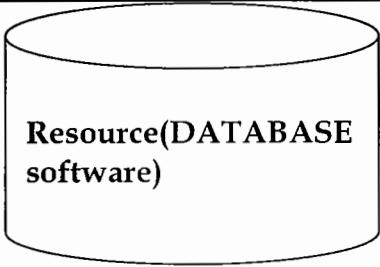
.....

->Operation n

Commit or Rollback the transaction

### Transactional Architecture





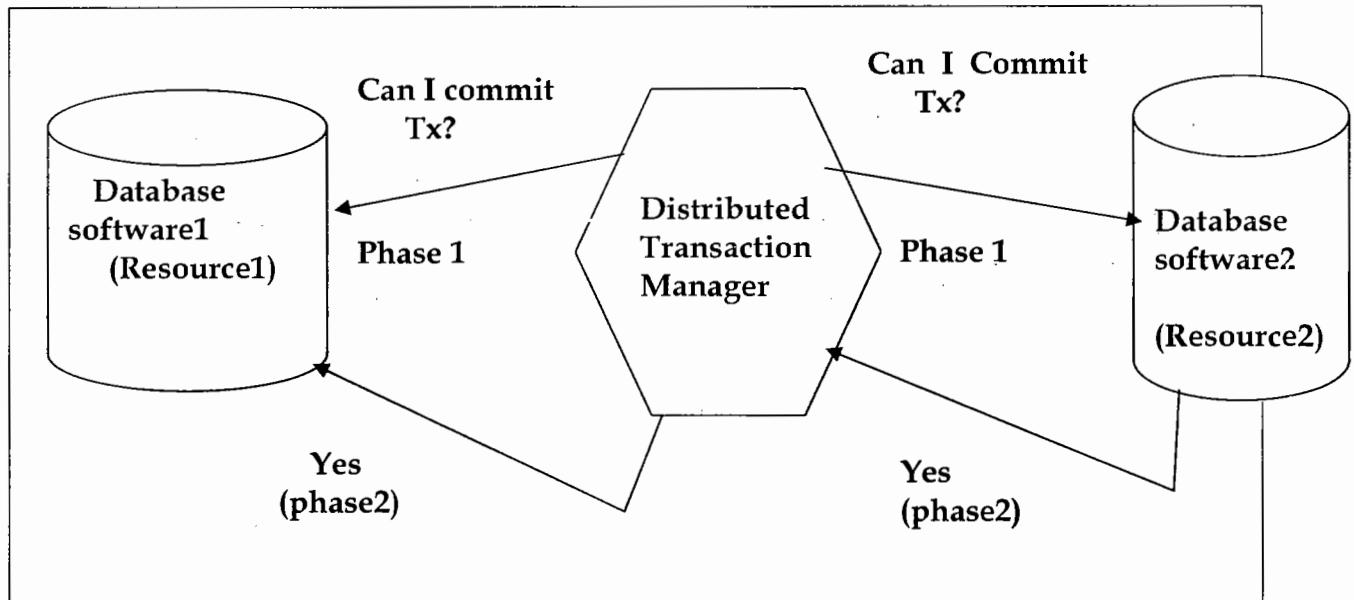
The Transactional application will give commit or rollback instruction to **TransactionManager** after executing business logic or persistence logic, the **TransactionManager** commits or rollbacks resource or Database software data through **Resource manager** like JDBC driver or Data Source.

There are two types Transactions

- a) Local Transaction ( All operations takes place in single Resource/DB software)
- b) Global Transaction (All

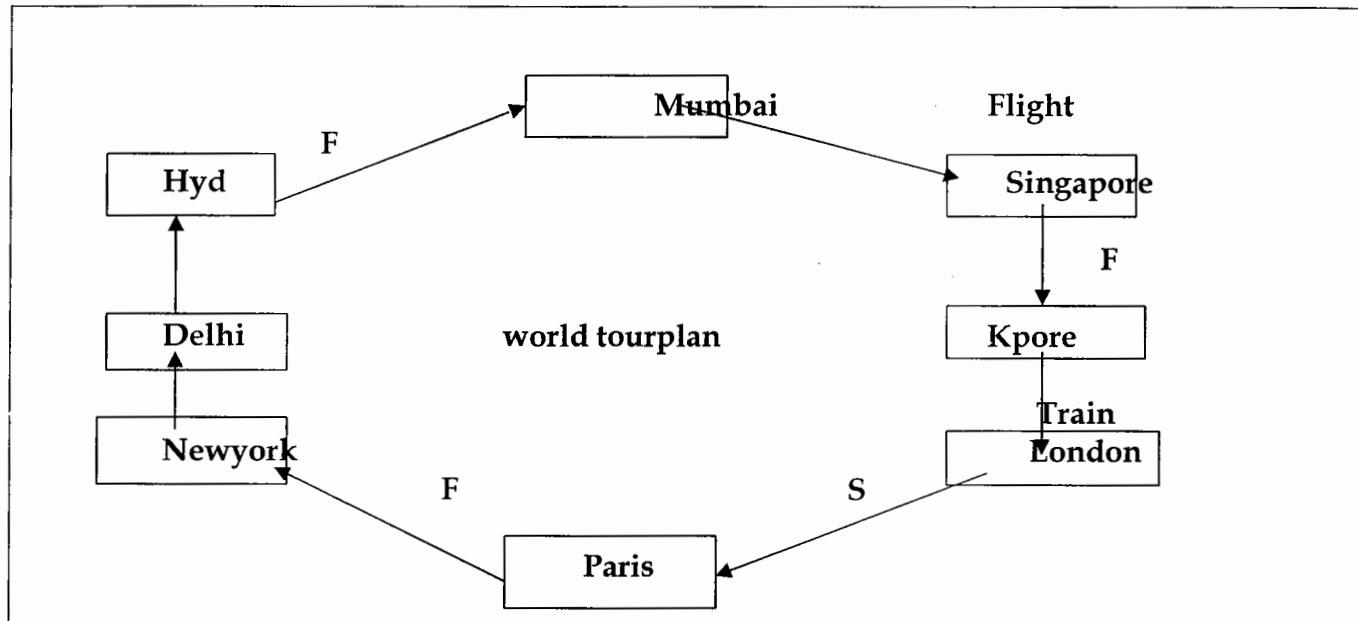
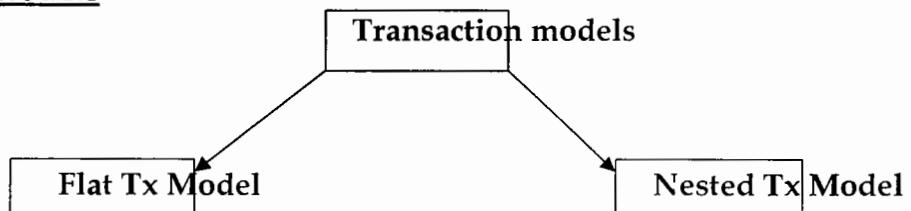
#### To work with Global/XA/Distributed Transactions:

- We need to use **XA Jdbc driver**
- Distributed transaction runs based on **2pc** protocol or principle
- **2pc**: 2-phase commit protocol/principle.



- In phase1 the **Distributed Transaction manager** will contact all the resources/DB s/w's that are involved in transaction and seeks the permission to commit the transaction?
- In phase2 if all Database s/w's or Resources responds and says yes then the Distributed Transaction will be committed otherwise will not be committed.(rollback)
- EJB, Spring, hibernate supports Local transactions
- Spring ,EJB supports distributed transactions but hibernate does not support distributed transaction because we create Transaction object based on HB session object which is specific to one DATABASE software
- Using spring with hibernate (Spring ORM) we can develop Distributed Transaction Application.

## Transaction Models



### Flat Transaction Model

```

Main Transaction{
    ->Journey 1 ticket booking
    ->Journey2 ticket booking
    .....
    .....
    .....
    .....
    ->Journeyn ticket booking
}
  
```

### Nested Transaction Model

```

MainTx{
    subTx1{
        // Journey 1 ticket booking
    }
    .....
    subTx2{
        // Journey2 ticket booking
    }
}
  
```

- When the above tour plan is given to a software application that runs with **Flat Transaction model**, the Transaction will be rollback, if one or other journey ticket is not available because in Flat Transaction model all journey tickets booking operations will be taken as the direct operations of Main Transaction. if the above tour plan given to a software application that runs with nested Transaction model, if one or other journey tickets are not available it will confirm remaining journey tickets because all journey ticket bookings will be taken as the sub Transactions of Main Transaction. So the success or failure of one sub Transaction does not affect other sub Transactions.

- Spring, EJB, Hibernate supports Flat Transaction model.
- EJB, Hibernate does not support nested Transactions, where as spring supports nested transaction.
- To apply nested transaction on hibernate logic use spring with hibernate (**spring ORM**)

### HBProj61(TXMgrt)

```

    └── src
        ├── com.nt.cfgs
        │   └── hibernate.cfg.xml
        ├── com.nt.dao
        │   ├── BankDao.java
        │   ├── BankDaoFactory.java
        │   └── BankDaoImpl.java
        ├── com.nt.domain
        │   └── Account.java
        ├── com.nt.test
        │   └── ClientApp.java
        └── com.nt.utility
            └── HibernateUtil.java

```

#### hibernate.cfg.xml

```

<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>

        <mapping class="com.nt.domain.Account"/>
    </session-factory>
</hibernate-configuration>

```

#### Account.java

```

package com.nt.domain;

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

```

```

@Entity
public class Account {
@Id
private int acno;
private String holder;
private double balance;
public Account() {
System.out.println("Account:0-param constructor");
}
public int getAcno() {
return acno;
}
public void setAcno(int acno) {
this.acno = acno;
}
public String getHolder() {
return holder;
}
public void setHolder(String holder) {
this.holder = holder;
}
public double getBalance() {
return balance;
}
public void setBalance(double balance) {
this.balance = balance;
}
@Override
public String toString() {
return "Account [acno=" + acno + ", holder=" + holder + ", balance=" + balance + "]";
}
}

```

**BankDao.java**

```

package com.nt.dao;

public interface BankDao {
public int withdraw(int acno,double amount);
public int deposite(int acno,double amount);
public boolean transferMoney(int srcNo,int destNo,double amount);
}

```

**BankDaolmpl.java**

```

package com.nt.dao;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

```

```
import com.nt.utility.HibernateUtil;

public class BankDaoImpl implements BankDao {
    private static final String WITHDRAW_QUERY="UPDATE Account SET balance=balance-:amt
WHERE acno=:srcNo";
    private static final String DEPOSITE_QUERY="UPDATE Account SET balance=balance+:amt WHERE
acno=:destNo";

    @Override
    public int withdraw(int acno, double amount) {
        // Session
        Session ses=HibernateUtil.getSession();
        // perform withdraw operation
        Query query=ses.createQuery(WITHDRAW_QUERY);
        query.setDouble("amt", amount);
        query.setInteger("srcNo", acno);
        int result=query.executeUpdate();
        return result;
    }
    @Override
    public int deposite(int acno, double amount) {
        Session ses=HibernateUtil.getSession();
        // perform withdraw operation
        Query query=ses.createQuery(DEPOSITE_QUERY);
        query.setDouble("amt", amount);
        query.setInteger("destNo", acno);
        int result=query.executeUpdate();
        return result;
    }
    @Override
    public boolean transferMoney(int srcNo, int destNo, double amount) {
        // get Session
        Session ses=HibernateUtil.getSession();
        // perform transfer Money operation
        Transaction tx=null;
        try{
            tx=ses.beginTransaction();
            // with operation
            int result1=withdraw(srcNo,amount);
            int result2=deposite(destNo,amount);
            if(result1==0 || result2==0){
                tx.rollback();
                System.out.println("Transaction rolled -Money not transferred");
                return false;
            }
        else{
```

```

tx.commit();
System.out.println("Transaction committed -Money transferred");
return true;
}
}//try
catch(Exception e){
tx.rollback();
return false;
}

}//method

}//class

```

**BankDaoFactory.java**

```

package com.nt.dao;

public class BankDaoFactory {

    public static BankDao getInstance(){
        return new BankDaoImpl();
    }

}

```

**HibernateUtil.java**

Same as previous

**ClientApp.java**

```

package com.nt.test;

import com.nt.dao.BankDao;
import com.nt.dao.BankDaoFactory;
import com.nt.utility.HibernateUtil;

public class ClientApp {

    public static void main(String[] args) {
        // Get DAO
        BankDao dao=BankDaoFactory.getInstance();
        // perform trasfer money
        boolean status=dao.transferMoney(101,102, 30000);
        System.out.println("Money Transferred?"+status);
        //close Session,SessionFactory
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}

```

**Transaction management gives support to:**

- 1) Atomicity:
- 2) Consistency
- 3) Isolation
- 4) Durability

When we enable Transaction management on database we can implement **ACID** properties.

**Atomicity:**

- The process of combining individual operations single unit and executing them by applying do everything or nothing principle is called **atomicity**.
- Atomicity word is created from the word **atom**. Transaction management is more of **Atomicity**.

**Consistency:**

- Even though the rules kept on Database software are violated during the coarse of Transaction (like balance must not be negative). If there is a guarantee that no rule will be violated at the end of Transaction (like when Transaction is committed or rolledback) then we can say that data is **consistent data**.

**Isolation:**

- Applying lock on DATABASE software at various levels and allowing only one user/one application at a time to manipulate Database data comes under **isolation**. It protects the data .

**Durability:**

- Bringing the database back to normal stage with support log files and backup files when it is crashed makes DATABASE data as durable data.
- If database software data is manipulated simultaneously and concurrently then there is possibility of getting data corruption and the following problems may occur they are
  - a) **Dirty read problem**
  - b) **Non repeatable read problem**
  - c) **phantom read problem**

To solve problem we can use various locks on DATABASE software with the support of isolation levels,they are:

- a) **Read committed isolation level solves dirty read problem**
- b) **Repeatable read isolation level solves dirty read problem, non-repeatable problems**
- c) **Serializable isolation level solve dirty read problem,non-repeatable read, phantom read problems**

**Dirty Read Problem**

User A	User B
<p>(1) User a begins Transaction and reads the balance rs:15000.</p> <p>(2) User A deposits the rs:10000 into the Joint Account  <math>bal = bal + amt(15000 + 10000) = 25000</math></p> <p>(4) User A aborts the Transaction, so original balance 15000 comes back to Account.  Note: Here the withdraw operation done by user B is <b>dirty read operation</b></p>	<p>(3) User B withdraw Rs 20000 from account  <math>bal = bal - amt(25000 - 20000 = 5000)</math></p> <ul style="list-style-type: none"> <li>➤ User A, user B are joint Account holders</li> <li>➤ Let us assume Database software is allowing uncommitted data read operations</li> </ul>

To solve this dirty read problem use **Read committed** isolation level that makes Data base to allow read operations only on committed data.

**Non-repeated problem:**

User A	User B
<p>(1) At beginning of the Transaction user A issues select query and get 10 records</p> <p>(3) At the end of Transaction user A issues same select query that is issued above, but gets modified 10 records(<b>non repeatable problem</b>)</p>	<p>(2) User B executes update query that modifies various values of user A retrieved 10 records</p>

- The data that is read at the beginning of the Transaction cannot be used at the end of Transaction by the same user this raises **non repeatable read problem**.
- To solve above problem apply Repeatable read isolation level that applies **write locks** on certain columns of the Database table.

**Phantom read problem:**

User A	User B
<p>(1)At the beginning of the Tx, user A issues select SQL query and gets 5 records.</p> <p>(3)At the end of Transaction, user A issues same select query and gets more records(<math>5+4=8</math>) records. Getting unexpected more records is nothing but <b>phantom</b> read problem</p>	<p>(2)User B inserts more 4 records in Database table matching the select query condition of user A</p>

**Note:**

Non-- repeatable read problem raises because of simultaneously updation is done by other user whereas phantom read problem raises because simultaneous insertion is done by other user.

To solve phantom read problem use serialization isolation level that apply read and write locks on whole Database.

**To applying Transaction isolation level on Database s/w from JDBC code we can use**

```
con.setTransactionIsolation(Connection.TRANSACTION_SERIALIZABLE)
con.setTransactionIsolation(8);
```

**Other isolation level are:**

```
Connection.TRANSACTION_READ_COMMITTED(2);
Connection.TRANSACTION_REPEATABLE_READ(4);
```

**To applying transaction isolation level from hibernate application:****In hibernate.cfg.xml**

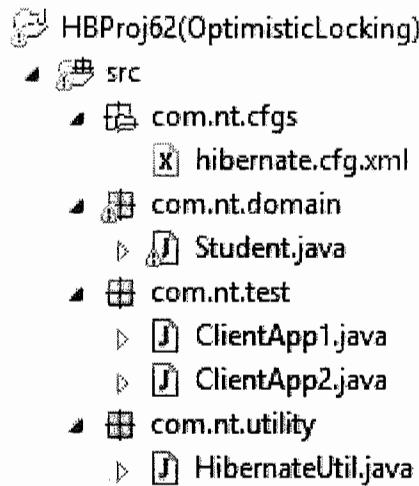
```
<property name="connection.isolation_level">8</property>
```

## Locking in Hibernate:

- Isolation level on configured from the configuration file applies locks on whole Database and kills the performance.
- To apply locks on specific record, we need to use hibernate locking mechanism.
- If multiple applications/users/threads access and manipulates Database table records simultaneously or concurrently then there is possibility of getting data corruption. In hibernate we can enable locking in two ways
  - 1) Optimistic locking: locking through versioning
  - 2) Pessimistic locking: locking through ses.get(--,LockMode.UPGRADE\_NOWAIT);

### Optimistic Locking:

- Once we enable the versioning the optimistic lock will come automatically on Database.
- It will check whether version of the object maintained by hibernate and version of the record maintained by database software are **matching or not**, If not matching, it will throw StaleStateException/DirtyStateException.
- If the state of the object/record that is there with hibernate not matching with the state of record in database table then object is called **Dirty state object or Stale state object**.



**hibernate.hbm.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <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>
        <property
            name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
        <mapping class="com.nt.domain.Student"/>
    </session-factory>
</hibernate-configuration>
```

**Student.java**

```
package com.nt.domain;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;

@Entity
@Table(name="student_Tab")
public class Student {
    @Id
    @Column(name="stno")
    private int sno;
    @Column(name="stname")
    private String sname;
    @Column(name="stadd")
    private String address;
    @Version
    @Column(name="ver_col")
    private int ver;

    public int getVer() {
```

```
return ver;
}
public void setVer(int ver) {
this.ver = ver;
}

//write setters and getters
public int getSno() {
return sno;
}

public void setSno(int sno) {
this.sno = sno;
}

public String getSname() {
return sname;
}
public void setSname(String sname) {
this.sname = sname;
}

public String getAddress() {
return address;
}
public void setAddress(String address) {
this.address = address;
}
@Override
public String toString() {
return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";
}
}
```

#### HibernateUtil.java

Same as above..

#### ClientApp1.java

```
package com.nt.test;

import org.hibernate.Transaction;

import org.hibernate.Session;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;
```

```
public class ClientApp1 {  
    public static void main(String[] args) throws Exception {  
        // get Session  
        Session ses = HibernateUtil.getSession();  
  
        //Load obj operation  
        Student st = (Student) ses.get(Student.class, 101);  
        if (st != null)  
            System.out.println(st);  
        //modify obj  
        Thread.sleep(40000); // Run Client2 in this sleep period  
        Transaction tx = null;  
        try {  
            tx = ses.beginTransaction();  
            st.setAddress("new delhi1");  
            tx.commit();  
            System.out.println("Object modified");  
        }  
        catch (Exception e) {  
            tx.rollback();  
            e.printStackTrace();  
        }  
  
        //close objs  
        HibernateUtil.closeSession();  
        HibernateUtil.closeSessionFactory();  
  
    }  
}
```

### ClientApp2.java

```
package com.nt.test;  
  
import org.hibernate.Transaction;  
  
import org.hibernate.Session;  
  
import com.nt.domain.Student;  
import com.nt.utility.HibernateUtil;  
  
public class ClientApp2 {  
    public static void main(String[] args) {  
        // get Session  
        Session ses = HibernateUtil.getSession();  
        //Load obj operation
```

```
Student st=(Student)ses.get(Student.class,101);
if(st!=null)
    System.out.println(st);
//modify obj
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    st.setAddress("new delhi2");
    tx.commit();
    System.out.println("Obj modified");
}
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
}

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}
```

### Execution procedure:

1. Run ClientApp1 that loads 101 object from DATABASE table whose version is 0 and Client Application gets into sleep state.
2. During this sleep period run Client application2 to load 101 object and modify the object(changes the object's version to 1)
3. When ClientApp1 completes sleep state and attempts to modify 101 object then hibernate notices the version of the object that it is having 0 not matching **with the version of the record (1) in Database table.**

#### Note:

Optimistic locking is bad practice it does not stop simultaneously access and modification of the record, rather it raises exception once the object or record is modified simultaneously.

### Pessimistic locking:

- This is proper locking mechanism of hibernate:
- It makes application/user acquiring the lock on the object, if any other client or user tries to access the same record simultaneously or concurrently then exception will be raised. For this we should call  
`ses.get(-,-)with LockMode.UPGRADE_NOWAIT`  
or  
`LockMode.PESSIMISTIC_READ/WRITE`

 HBProj63(PessimisticLocking)

```

    ▲ src
      ▲ com.nt.cfgs
        ☐ hibernate.cfg.xml
      ▲ com.nt.domain
        ▶ Student.java
      ▲ com.nt.test
        ▶ ClientApp1.java
        ▶ ClientApp2.java
      ▲ com.nt.utility
        ▶ HibernateUtil.java

```

[hibernate.cfg.xml](#)

```

<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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>
    <property
      name="connection.provider_class">org.hibernate.c3p0.internal.C3P0ConnectionProvider</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
      <mapping class="com.nt.domain.Student"/>
  </session-factory>
</hibernate-configuration>

```

[Student.java](#)

```

package com.nt.domain;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Version;

@Entity
@Table(name="student_Tab")

```

```
public class Student {  
    @Id  
    @Column(name="stno")  
    private int sno;  
    @Column(name="stname")  
    private String sname;  
    @Column(name="stadd")  
    private String address;  
  
    //write setters and getters  
    public int getSno() {  
        return sno;  
    }  
  
    public void setSno(int sno) {  
        this.sno = sno;  
    }  
  
    public String getName() {  
        return sname;  
    }  
    public void setName(String sname) {  
        this.sname = sname;  
    }  
  
    public String getAddress() {  
        return address;  
    }  
    public void setAddress(String address) {  
        this.address = address;  
    }  
    @Override  
    public String toString() {  
        return "Student [sno=" + sno + ", sname=" + sname + ", address=" + address + "]";  
    }  
}
```

**HibernateUtil.java**

Same as previous

**ClientApp1.java**

```
package com.nt.test;  
  
import org.hibernate.Transaction;  
import org.hibernate.LockMode;  
import org.hibernate.Session;
```

```
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class ClientApp1 {
    public static void main(String[] args) throws Exception {
        // get Session
        Session ses = HibernateUtil.getSession();

        //Load obj operation
        Student st = (Student) ses.get(Student.class, 101, LockMode.PESSIMISTIC_READ);
        if(st!=null)
            System.out.println(st);
        //modify obj
        Thread.sleep(40000); // Run Client2 in this sleep period
        Transaction tx = null;
        try{
            tx = ses.beginTransaction();
            st.setAddress("new delhi3");
            tx.commit();
            System.out.println("Object modified");
        }
        catch(Exception e){
            tx.rollback();
            e.printStackTrace();
        }

        //close objs
        HibernateUtil.closeSession();
        HibernateUtil.closeSessionFactory();
    }
}
```

### ClientApp2.java

```
package com.nt.test;

import org.hibernate.Transaction;
import org.hibernate.LockMode;
import org.hibernate.Session;

import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class ClientApp2 {
    public static void main(String[] args) {
        // get Session
        Session ses = HibernateUtil.getSession();
```

```

//Load obj operation
Student st=(Student)ses.get(Student.class,101,LockMode.UPGRADE_NOWAIT);
if(st!=null)
    System.out.println(st);
//modify obj
Transaction tx=null;
try{
    tx=ses.beginTransaction();
    st.setAddress("new delhi4");
    tx.commit();
    System.out.println("Obj modified");
}
catch(Exception e){
    tx.rollback();
    e.printStackTrace();
}

//close objs
HibernateUtil.closeSession();
HibernateUtil.closeSessionFactory();

}
}

```

**Q. Can you explain different ORM level?**

- a) Pure relational (eg: stored procedures /functions)
- b) Light object mapping (eg: JDBC)
- c) Medium object mapping (eg: EJB2.x entity beans)
- d) Full object mapping(eg: Hibernate)

**Supports inheritance, polymorphism, composition etc...**

**Different levels of ORM quality: -**

- **Pure relational:** entire application, including the user interface, is designed around the relational model and SQL-based relational operations
- **Light object mapping:** The entities are represented as classes that are mapped manually to the relational tables
- **Medium object mapping:** The application is designed around an object model. The SQL code is generated at build time. And the associations between objects are supported by the persistence mechanism, and queries are specified using an object-oriented expression language
- **Full object mapping:** supports sophisticated object modeling: composition, inheritance, polymorphism and persistence

**Q. Can u explain mismatches between java and RDBMS how hibernate manages these mismatches?**

Ans:

1. **Sub types problem:**

Java classes can be placed in inheritance ,but Database tables can't be placed in inheritance so java classes that are inheritance can't be mapped with Database table.To solve this problem we use **inheritance mapping**

## 2. Problem of Granularity:

Java classes can be designed through composition and these classes want to map with single DATABASE table i.e. more granuals wants to map with single Database table Which is not possible directly. To solve this problem use **component mapping**.

## 3. Identity problem:

In java we can do two types of comparisons

1. Reference comparisons(==)
2. Data comparisons>equals(-)method)

In Database software the record will be identified with pk value and will compared with other record using pk value, so we use <id> tag to map pk value Database table record with java object for **identity**.

## 4. Navigation problem:

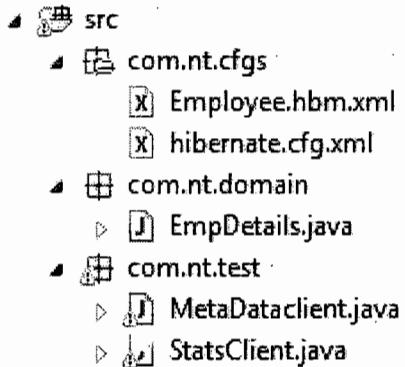
In Database software we can navigate from one table and another table using Fk column. But for object level navigation we need to use reference variables, so to synchronize FK column based navigation with referenced based navigation in Objects, we need to work with **Association Mapping**.

## Hibernate statics:

- It helps us to find out various statics with respect to SessionFactory object like. No of session objects are opened, transactions opened, no of session object closed and etc...
- To get statics object :
 

```
Statistics stats=factory.getStatistics();
stats.setStatisticsEnabled(true);
```

HBProj64(HB Stastitics & MetaData)



**hibernate.cfg.xml**

```
<!DOCTYPE hibernate-configuration PUBLIC
  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
  <session-factory>
    <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>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
    <property name="dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <property name="generate_statistics"> true</property>
    <mapping resource="com/nt/cfgs/Employee.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

**Employee.hbm.xml**

```
<!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.domain.EmpDetails" table="Employee">
    <id name="no" column="ENO"> <!-- Singular Id field -->
      <generator class="native"/>
    </id>
    <property name="fname" >
      <column name="FIRSTNAME"/>
    </property>
    <property name="lname" >
      <column name="LASTNAME"/>
    </property>
    <property name="mail" >
      <column name="EMAIL"/>
    </property>
  </class>
</hibernate-mapping>
```

**EmpDetails.java**

```
package com.nt.domain;

public class EmpDetails {
  //domain class properties
  private int no;
  private String fname;
```

```
private String lname;
private String mail;

public EmpDetails(){
System.out.println("EmpDetails:0-param constructor");
}

public int getNo() {

return no;
}
public void setNo(int no) {

this.no = no;
}
public String getFname() {

return fname;
}
public void setFname(String fname) {

this.fname = fname;
}
public String getLname() {

return lname;
}
public void setLname(String lname) {

this.lname = lname;
}
public String getMail() {

return mail;
}
public void setMail(String mail) {

this.mail = mail;
}

@Override
public String toString() {
return "EmpDetails [no=" + no + ", fname=" + fname + ", lname=" + lname + ", mail=" + mail + "]";
}

}
```

StatsClient.java

```
package com.nt.test;

import java.util.Date;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.stat.EntityStatistics;
import org.hibernate.stat.Statistics;

import com.nt.domain.EmpDetails;

public class StatsClient {

    public static void main(String[] args) {
        Configuration cfg=new Configuration();
        cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");

        SessionFactory factory=cfg.buildSessionFactory();

        Session ses1=factory.openSession();
        Session ses2=factory.openSession();
        Session ses3=factory.openSession();

        Transaction tx=null;
        try{
            tx=ses1.beginTransaction();
            EmpDetails ed=new EmpDetails();
            ed.setNo(101); ed.setFname("raja");
            ed.setLname("rao"); ed.setMail("rao@123.com");
            ses1.save(ed);

            tx.commit();
        }
        catch(Exception e){
            e.printStackTrace();
        }
        ses1.close();

        Statistics stats=factory.getStatistics();
        stats.setStatisticsEnabled(true);

        System.out.println("Stats enabled ?"+stats.isStatisticsEnabled());
        System.out.println("SEssion Start Time"+new Date(stats.getStartTime()));
        System.out.println("no.of opened sessions"+stats.getSessionOpenCount());
        System.out.println("no.of closed sessions"+stats.getSessionCloseCount());
    }
}
```

```

System.out.println("no.of Transaction count"+stats.getTransactionCount());
System.out.println("no.of Transaction successful close
count"+stats.getSuccessfulTransactionCount());

//Entity statics
EntityStatistics es=stats.getEntityStatistics("com.nt.domain.EmpDetails");
System.out.println("Delete count"+es.getDeleteCount());
System.out.println("insert count"+es.getInsertCount());
System.out.println("Load count"+es.getLoadCount());
System.out.println("Update count"+es.getUpdateCount());

factory.close();

}
}

```

## Hibernate Metadata:

In JDBC we use Database Metadata support or ResultSetMetaData support to get column Names and Data Type, same thing can be done in hibernate by suing ClassMetadata and AbstractEntityPersister objects.

### MetaDataclient.java

```

package com.nt.test;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import org.hibernate.metadata.ClassMetadata;
import org.hibernate.persister.entity.AbstractEntityPersister;
import org.hibernate.type.Type;

import com.nt.domain.EmpDetails;

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

Configuration cfg=new Configuration();
cfg=cfg.configure("/com/nt/cfgs/hibernate.cfg.xml");
SessionFactory factory=cfg.buildSessionFactory();

ClassMetadata mdata = factory.getClassMetadata(EmpDetails.class);
AbstractEntityPersister persister = (AbstractEntityPersister) mdata;

String tableName = persister.getTableName(); // tablename
String idCol = persister.getIdentifierColumnNames()[0]; // id col name
Type idColType = persister.getIdentifierType(); // id col type
String[] columnNames = persister.getPropertyNames(); //other col names
Type[] columnTypes = persister.getPropertyTypes(); // other col types

```

```
System.out.println("Table Name :" + tableName);
System.out.print("Id Column =" + idCol);
System.out.println("\n Identifier type=" + idColType.getName());

System.out.println("\n Other columns are");
for (int i = 0; i < columnNames.length; i++) {
    System.out.println(columnNames[i] + " : "
        + columnTypes[i].getName());
}
}
```

**Note:**

By using ClassMetadata and AdstractEntityPersister we can get underlying tablename, column name,column types.

## How to implement a soft delete with Hibernate

In some applications, you don't want to, or you are not allowed to permanently remove a record from the database. But you still need to remove or hide records which are no longer active. One example could be a user account which you want to keep because it is linked to other business objects which are still used.

You have 2 basic options to keep this information in your system. You can either keep an audit log that documents all changes or perform a soft delete that hides the removed records. I will explain the audit log option in another blog post. Today I want to show you how to implement a soft delete with Hibernate. But before I do that, let me quickly explain what a soft delete is.

### **What is a soft delete?**

A soft delete performs an update to mark a record as deleted instead of removing it from the database table. Common ways to model a soft delete are:

- a boolean that indicates if the record is active or deleted,
- an Enumerated which models the state of the record,
- a timestamp that stores the date and time when the soft delete was performed.
- Not deleting records physically and making records as deleted records is called soft deletion.
- @SQLDelete,@ SQLUpdate,@SQLInsert queries are given to place customSQLQueries to perform sql operations on database.

**usecase:**

Useful while deleting bank accounts, Email accounts and etc...

## How to implement a soft delete with Hibernate

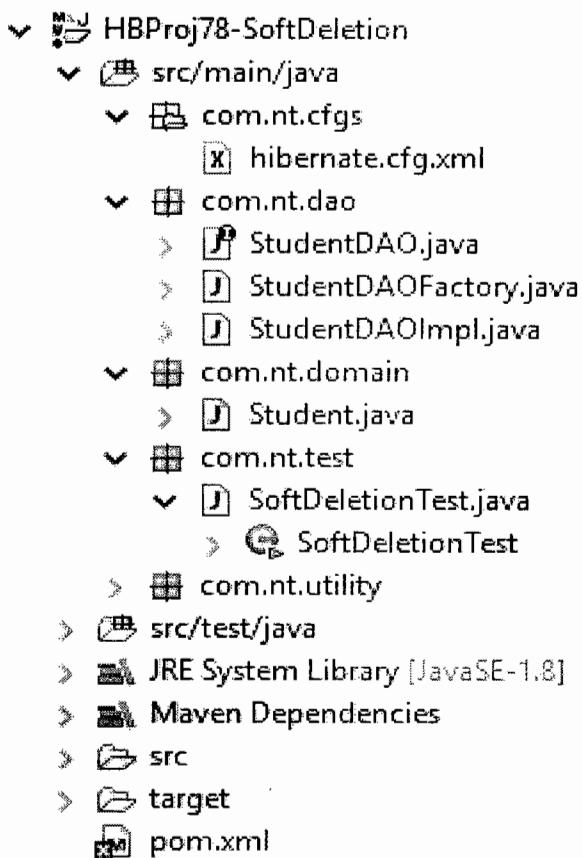
It's not that difficult to implement a soft delete with Hibernate. You just have to:

1. tell Hibernate to perform a SQL UPDATE instead of a DELETE operation and
2. exclude all "deleted" records from your query results.

I'll show you how you can easily do that in this post. All examples will use the following Account entity which uses the *AccountState* enum to indicate if an account is *INACTIVE*, *ACTIVE* or *DELETED*.

```
@Entity
@NamedQuery(name = "Account.FindByName", query = "SELECT a FROM Account a WHERE name like
:name")
public class Account {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Column(name = "id", updatable = false, nullable = false)
    private Long id;
    @Column
    private String name;
    @Column
    @Enumerated(EnumType.STRING)
    private AccountState state;
    ...
}
```

}

hibernate.cfg.xml:-

```
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- connection 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>
        <property name="connection.pool_size">40</property>
```

```
<!-- hibernate properties -->
    <property name="show_sql">true</property>
    <property name="format_sql">true</property>
    <!-- Dialect cfg -->
    <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
    <!-- mapping file -->
    <mapping class="com.nt.domain.Student"/>
</session-factory>

</hibernate-configuration>
```

**StudentDAO.java:-**

```
package com.nt.dao;

public interface StudentDAO {
    public int deleteStudentByNo(int sno);
    public void loadAllStudents();

}
```

**StudentDAOFactory.java:-**

```
package com.nt.dao;

public class StudentDAOFactory {

    public static StudentDAO getInstance(){
        return new StudentDAOImpl();
    } //method
} //class
```

**StudentDAOImpl.java:-**

```
package com.nt.dao;

import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.nt.domain.Student;
import com.nt.utility.HibernateUtil;

public class StudentDAOImpl implements StudentDAO {

    @Override
    public int deleteStudentByNo(int sno) {
        Session ses=null;
        Transaction tx=null;
        Student stud=null;
        // get Session
        ses=HibernateUtil.getSession();
```

```

//load object
stud=ses.get(Student.class,sno);
//delete the object
try{
    tx=ses.beginTransaction();
    ses.delete(stud);
    tx.commit();
    return 1;
}
catch(Exception e){
    tx.rollback();
    return 0;
}//catch

}//method
@Override
public void loadAllStudents() {
    Session ses=null;
    Student stud=null;
    Query query=null;
    List<Student> list=null;
    // get Session
    ses=HibernateUtil.getSession();
    //prepare Query
    query=ses.createQuery("from Student");
    //execute Query
    list=query.list();
    //process the ReusltSet
    list.forEach(st->{
        System.out.println(st);
    });
}//method
}//class

```

**Student.java:-**

```

package com.nt.domain;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import org.hibernate.annotations.SQLDelete;
import org.hibernate.annotations.Where;

@Entity
@Table(name="Student_tab_softdeletion")
@SQLDelete(sql="UPDATE student_Tab_softdeletion set status='DELETED' where sno=?")
@Where(clause="status is NULL")
public class Student {
    @Id

```

```

private int sno;
private String sname;
private String sadd;
public int getSno() {
    return sno;
}
public void setSno(int sno) {
    this.sno = sno;
}
public String getSname() {
    return sname;
}
public void setSname(String sname) {
    this.sname = sname;
}
public String getSadd() {
    return sadd;
}
public void setSadd(String sadd) {
    this.sadd = sadd;
}
@Override
public String toString() {
    return "Student [sno=" + sno + ", sname=" + sname + ", sadd=" +
+ sadd + "]";
}
}

```

**SoftDeletionTest.java:-**

```

package com.nt.test;

import com.nt.dao.StudentDAO;
import com.nt.dao.StudentDAOFactory;
import com.nt.utility.HibernateUtil;

public class SoftDeletionTest {

    public static void main(String[] args) {
        StudentDAO dao=null;
        int count=0;
        //get DAO
        dao=StudentDAOFactory.getInstance();
        //invoke methods
        /*count=dao.deleteStudentByNo(102);
        if(count==0)
            System.out.println("Record not deleted");
        else
            System.out.println("Record deleted");*/
    }
}

```

```
        dao.loadAllStudents();

    //close Hibernate objects
    HibernateUtil.closeSession();
    HibernateUtil.closeSessionFactory();
} //main
}//class
```

**HibernateUtil.java:-**

```
package com.nt.utility;

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

public class HibernateUtil {
    private static ThreadLocal<Session> threadLocal=new ThreadLocal();
    private static SessionFactory factory=null;
    static{
        factory=new
Configuration().configure("/com/nt/cfgs/hibernate.cfg.xml").

buildSessionFactory();
    }
    public static Session getSession(){
        Session ses=null;
        ses=threadLocal.get();
        if(ses==null){
            ses=factory.openSession();
            threadLocal.set(ses);
        }
        return ses;
    } //getSession()

    public static void closeSession(){
        Session ses=null;

        ses=threadLocal.get();
        if(ses!=null){
            ses.close();
            threadLocal.remove();
        }
    }

    public static void closeSessionFactory(){
        factory.close();
    }
} //class
```

**pom.xml:-**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>nit</groupId>
  <artifactId>HBProj20LayeredWebApp</artifactId>
  <packaging>war</packaging>
  <version>0.0.1-SNAPSHOT</version>
  <name>HBProj20LayeredWebApp Maven Webapp</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.hibernate</groupId>
      <artifactId>hibernate-core</artifactId>
      <version>5.1.0.Final</version>
    </dependency>
    <dependency>
      <groupId>com.oracle</groupId>
      <artifactId>ojdbc6</artifactId>
      <version>11.2.0</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.1.0</version>
    </dependency>
    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>jstl</artifactId>
      <version>1.2</version>
    </dependency>
    <dependency>
      <groupId>taglibs</groupId>
      <artifactId>standard</artifactId>
      <version>1.1.2</version>
    </dependency>
  </dependencies>
  <build>
    <finalName>HBProj20LayeredWebApp</finalName>
  </build>
</project>
```

## Update the record instead of deleting it

To implement a soft delete, you need to override Hibernate's default remove operation. You can do that with an `@SQLDelete` annotation. This annotation allows you to define a custom, native SQL query that Hibernate will execute when you delete the entity. You can see an example of it in the following code snippet.

```
@Entity  
@SQLDelete(sql = "UPDATE account SET state = 'DELETED' WHERE id = ?", check =  
ResultCheckStyle.COUNT)  
public class Account { ... }
```

The `@SQLDelete` annotation in the previous code snippet tells Hibernate to execute the given SQL UPDATE statement instead of the default SQL DELETE statement. It changes the state of the account to *DELETED* and you can use the *state* property in all queries to exclude the deleted accounts.

```
Account a = ses.get(Account.class, a.getId());  
ses.delete(a);
```

That is all you need to do to create a basic soft delete implementation. But there are 2 other things you need to handle:

1. When you delete an Account entity, Hibernate doesn't update the value of its *state* attribute in the current session.
2. You need to adapt all queries to exclude the deleted entities.

## Update state property in current session

Hibernate doesn't parse the native query you provide to the `@SQLDelete` annotation. It just sets the values of the bind parameters and executes it. It, therefore, doesn't know that you provided an SQL UPDATE statement instead of a DELETE statement to the `@SQLDelete` annotation. It also doesn't know that the value of the *state* attribute is outdated after it performed the delete operation.

In most cases, this is not an issue. As soon as Hibernate executes the SQL statement, the database record gets updated and all queries use the new *state* value. But what about the Account entity that you provided to the `EntityManager.remove(Object entity)` operation?

The *state* property of that entity is outdated. That's not a big deal if you release the reference immediately after removing it. In all other cases, you should update the attribute yourself.

The easiest way to do that is to use a lifecycle callback, as I do in the following code snippet. The `@PreRemove` annotation on the *deleteUser* method tells Hibernate to call this method before it performs the remove operation. I use it to set the value of the *state* property to *DELETED*.

```

@Entity
@SQLDelete(sql = "UPDATE account SET state = 'DELETED' WHERE id = ?", check =
ResultCheckStyle.COUNT)
public class Account {
...
@PreRemove
public void deleteUser() {
this.state = AccountState.DELETED;
}
}

```

## Exclude “deleted” entities in queries

You need to check the *state* attribute in all queries to exclude the deleted database records from the query results. This is an error-prone task if you do it manually and it forces you to define all queries yourself. The *EntityManager.find(Class entityClass, Object primaryKey)* method and the corresponding methods on the Hibernate *Session* don't know about the semantic of the *state* attribute and don't take it into account.

Hibernate's *@Where* annotation provides a better way to exclude all deleted entities. It allows to define an SQL snippet which Hibernate adds to the WHERE clause of all queries. The following code snippet shows a *@Where* annotation that excludes a record if its *state* is *DELETED*.

```

@Entity
@SQLDelete(sql = "UPDATE account SET state = 'DELETED' WHERE id = ?", check =
ResultCheckStyle.COUNT)
@Where(clause = "state <> 'DELETED'")
@NamedQuery(name = "Account.FindByName", query = "SELECT a FROM Account a WHERE
name like :name")
public class Account { ... }

```

As you can see in the following code snippets, Hibernate adds the defined WHERE clause when you perform a JPQL query or call the *EntityManager.find(Class entityClass, Object primaryKey)* method.

```

TypedQuery<Account> q =
em.createNamedQuery("Account.FindByName", Account.class);
q.setParameter("name", "%ans%");
Account a = q.getSingleResult();

16:07:59,511 DEBUG SQL:92 - select account0_.id as id1_0_, account0_.name as name2_0_, accou
account0_.state <> 'DELETED') and (account0_.name like ?)
Account a = em.find(Account.class, a.getId());

```

## Summary

As you've seen, it's quite simple to implement a soft delete with Hibernate. You just have to use a *@SQLDelete* annotation to define a custom SQL statement for the delete operation. You should also use Hibernate's *@Where* annotation to define a predicate that excludes all deleted records by default.

## Hibernate FAQs

### 1. What is ORM ?

ORM stands for object/relational mapping. ORM is the automated persistence of objects in a Java application to the tables in a relational database.

### 2. What does ORM consists of ?

An ORM solution consists of the followig four pieces:

- API for performing basic CRUD operations
- API to express queries refering to classes
- Facilities to specify metadata
- Optimization facilities : dirty checking,lazy associations fetching

### 3. What are the ORM levels ?

The ORM levels are:

- Pure relational (stored procedure.)
- Light objects mapping (JDBC)
- Medium object mapping
- Full object Mapping (composition,inheritance, polymorphism, persistence by reachability)

### 4. What is Hibernate?

Hibernate is a pure Java object-relational mapping (ORM) and persistence framework that allows you to map plain old Java objects to relational database tables using (XML) configuration files. Its purpose is to relieve the developer from a significant amount of relational data persistence-related programming tasks.

### 5. Why do you need ORM tools like hibernate?

The main advantage of ORM like hibernate is that it shields developers from messy SQL. Apart from this, ORM provides following benefits:

- **Improved productivity**
  - High-level object-oriented API
  - Less Java code to write
  - No SQL to write
- **Improved performance**
  - Sophisticated caching
  - Lazy loading
  - Eager loading
- **Improved maintainability**
  - A lot less code to write
- **Improved portability**
  - ORM framework generates database-specific SQL for you

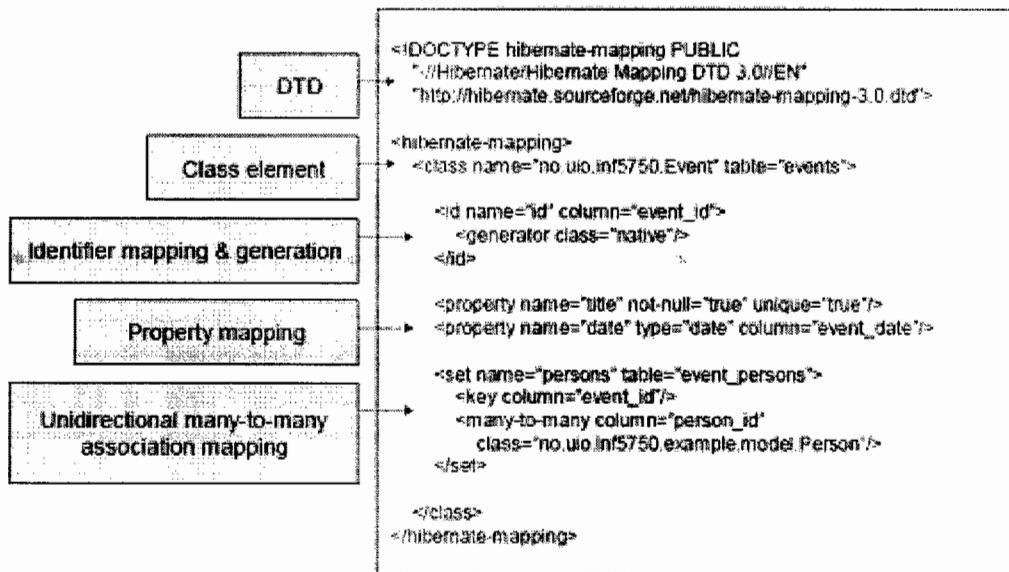
### 6. What Does Hibernate Simplify?

Hibernate simplifies:

- Saving and retrieving your domain objects
- Making database column and table name changes
- Centralizing pre save and post retrieve logic
- Complex joins for retrieving related items
- Schema creation from object model

## 7. What is the need for Hibernate xml mapping file?

- Hibernate mapping file tells Hibernate which tables and columns to use to load and store objects. Typical mapping file look as follows:



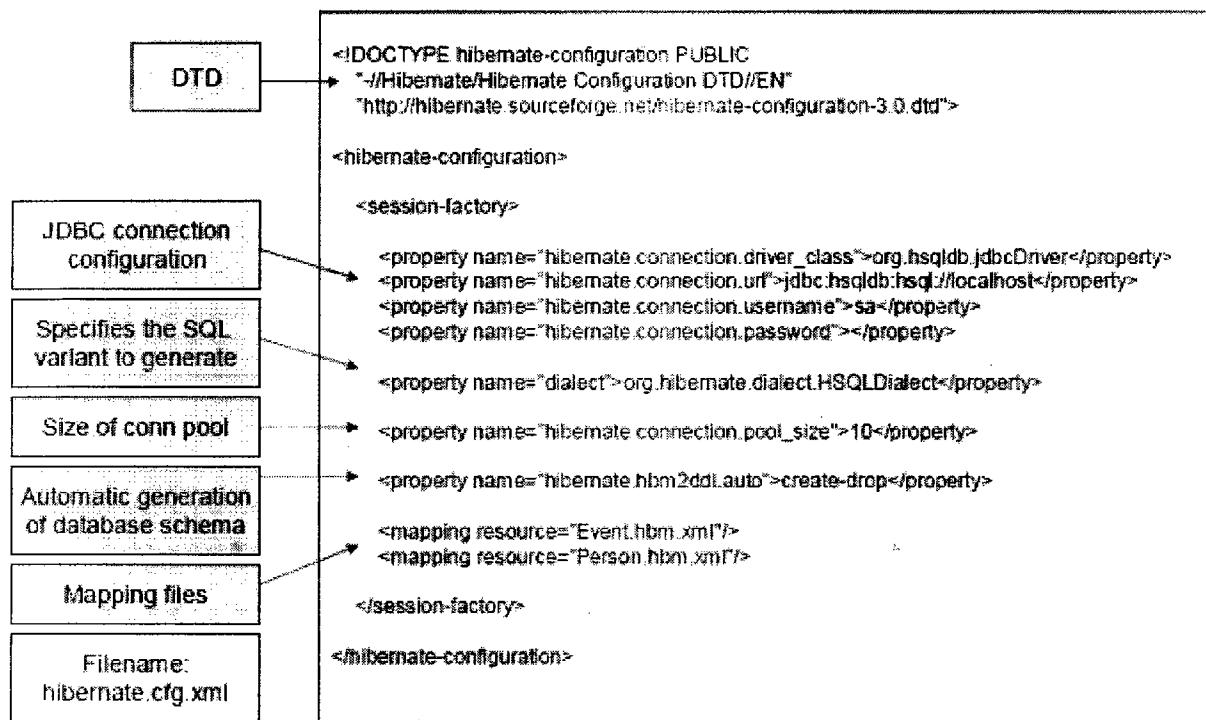
## 8. What are the most common methods of Hibernate configuration?

The most common methods of Hibernate configuration are:

- Programmatic configuration
- XML configuration (`hibernate.cfg.xml`)

## 9. What are the important tags of hibernate.cfg.xml?

Following are the important tags of hibernate.cfg.xml:



## 10. What are the Core interfaces are of Hibernate framework?

The five core interfaces are used in just about every Hibernate application. Using these interfaces, you can store and retrieve persistent objects and control transactions.

- Session interface
- SessionFactory interface
- Configuration interface
- Transaction interface
- Query and Criteria interfaces

## 11. What role does the Session interface play in Hibernate?

The Session interface is the primary interface used by Hibernate applications. It is a single-threaded, short-lived object representing a conversation between the application and the persistent store. It allows you to create query objects to retrieve persistent objects.

Session session = sessionFactory.openSession();

### Session interface role:

- Wraps a JDBC connection
- Factory for Transaction
- Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier

## 12. What role does the SessionFactory interface play in Hibernate?

The application obtains Session instances from a SessionFactory. There is typically a single SessionFactory for the whole application created during application initialization. The SessionFactory caches generated SQL statements and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and may be reused in a future unit of work.

```
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

### 13. What is the general flow of Hibernate communication with RDBMS?

The general flow of Hibernate communication with RDBMS is :

- Load the Hibernate configuration file and create configuration object. It will automatically load all hbm mapping files
- Create session factory from configuration object
- Get one session from this session factory
- Create HQL Query
- Execute query to get list containing Java objects

### 14. What is Hibernate Query Language (HQL)?

Hibernate offers a query language that embodies a very powerful and flexible mechanism to query, store, update, and retrieve objects from a database. This language, the Hibernate query Language (HQL), is an object-oriented extension to SQL.

### 15. How do you map Java Objects with Database tables?

- First we need to write Java domain objects (beans with setter and getter).
- Write hbm.xml, where we map java class to table and database columns to Java class variables.

**Example :**

```
<hibernate-mapping>
<class name="com.test.User" table="user">
<property column="USER_NAME" length="255"
  name="userName" not-null="true" type="java.lang.String"/>
<property column="USER_PASSWORD" length="255"
  name="userPassword" not-null="true" type="java.lang.String"/>
</class>
</hibernate-mapping>
```

### 16. What is the difference between load() and get()?

**load() vs. get() :-**

load()	get()
Only use the load() method if you are sure that the object exists.	If you are not sure that the object exists, then use one of the get() methods.
load() method will throw an exception if the unique id is not found in the database.	get() method will return null if the unique id is not found in the database.
load() just returns a proxy by default and database won't be hit until the proxy is first invoked.	get() will hit the database immediately.

**17. What is the difference between and merge and update ?**

Use update() if you are sure that the session does not contain an already persistent instance with the same identifier, and merge() if you want to merge your modifications at any time without consideration of the state of the session.

**18. How do you define sequence generated primary key in hibernate?**

Using <generator> tag.

**Example:-**

```
<id column="USER_ID" name="id" type="java.lang.Long">
  <generator class="sequence">
    <param name="table">SEQUENCE_NAME</param>
  <generator>
</id>
```

**19. Define cascade and inverse option in one-many mapping?**

cascade - enable operations to cascade to child entities.

cascade="all|none|save-update|delete|all-delete-orphan"

inverse - mark this collection as the "inverse" end of a bidirectional association.

inverse="true|false"

Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**20. What do you mean by Named SQL query?**

Named SQL queries are defined in the mapping xml document and called wherever required.

**Example:**

```
<sql-query name = "empdetails">
  <return alias="emp" class="com.test.Employee"/>
  SELECT emp.EMP_ID AS {emp.empid},
    emp.EMP_ADDRESS AS {emp.address},
    emp.EMP_NAME AS {emp.name}
  FROM Employee EMP WHERE emp.NAME LIKE :name
</sql-query>
```

**Invoke Named Query :**

```
List people = session.getNamedQuery("empdetails")
  .setString("TomBrady", name)
  .setMaxResults(50)
  .list();
```

**21. How do you invoke Stored Procedures?**

```
<sql-query name="selectAllEmployees_SP" callable="true">
<return alias="emp" class="employee">
<return-property name="empid" column="EMP_ID"/>

<return-property name="name" column="EMP_NAME"/>
<return-property name="address" column="EMP_ADDRESS"/>
{ ? = call selectAllEmployees() }
</return>
</sql-query>
```

**22. Explain Criteria API**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

**Example :**

```
List employees = session.createCriteria(Employee.class)
    .add(Restrictions.like("name", "a%"))
    .add(Restrictions.like("address", "Boston"))
    .addOrder(Order.asc("name"))
    .list();
```

**23. Define HibernateTemplate?**

org.springframework.orm.hibernate.HibernateTemplate is a helper class which provides different methods for querying/retrieving data from the database. It also converts checked HibernateExceptions into unchecked DataAccessExceptions.

**24. What are the benefits does HibernateTemplate provide?**

The benefits of HibernateTemplate are :

- HibernateTemplate, a Spring Template class simplifies interactions with Hibernate Session.
- Common functions are simplified to single method calls.
- Sessions are automatically closed.
- Exceptions are automatically caught and converted to runtime exceptions.

**25. How do you switch between relational databases without code changes?**

Using Hibernate SQL Dialects , we can switch databases. Hibernate will generate appropriate hql queries based on the dialect defined.

**26. If you want to see the Hibernate generated SQL statements on console, what should we do?**

In Hibernate configuration file set as follows:

```
<property name="show_sql">true</property>
```

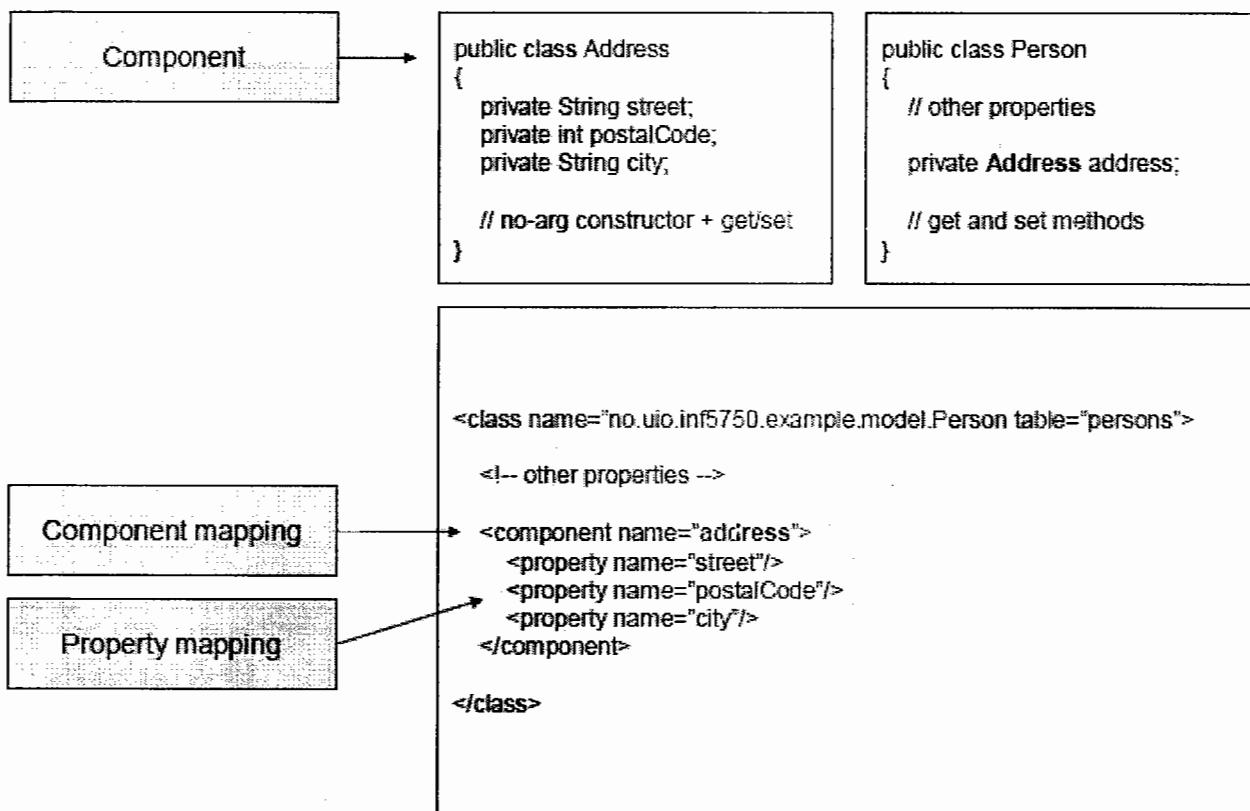
## 27. What are derived properties?

The properties that are not mapped to a column, but calculated at runtime by evaluation of an expression are called derived properties. The expression can be defined using the formula attribute of the element.

## 28. What is component mapping in Hibernate?

- A component is an object saved as a value, not as a reference
- A component can be saved directly without needing to declare interfaces or identifier properties
- Required to define an empty constructor
- Shared references not supported

**Example:**



## 29. What is the difference between sorted and ordered collection in hibernate?

**sorted collection vs. order collection :-**

### sorted collection

A sorted collection is sorting a collection by utilizing the sorting features provided by the Java collections framework. The sorting occurs in the memory of JVM which running Hibernate, after the data being read from database using java comparator.

### order collection

Order collection is sorting a collection by specifying the order-by clause for sorting this collection when retrieval.

If your collection is not large, it will be more

If your collection is very large, it will be

efficient way to sort it.

more efficient way to sort it .

### 31. What is the advantage of Hibernate over jdbc?

Hibernate Vs. JDBC :-

#### JDBC

With JDBC, developer has to write code to map an object model's data representation to a relational data model and its corresponding database schema.

With JDBC, the automatic mapping of Java objects with database tables and vice versa conversion is to be taken care of by the developer manually with lines of code.

JDBC supports only native Structured Query Language (SQL). Developer has to find out the efficient way to access database, i.e. to select effective query from a number of queries to perform same task.

Application using JDBC to handle persistent data (database tables) having database specific code in large amount. The code written to map table data to application objects and vice versa is actually to map table fields to object properties. As table changed or database changed then it's essential to change object structure as well as to change code written to map table-to-object/object-to-table.

With JDBC, it is developer's responsibility to handle JDBC result set and convert it to Java objects through code to use this persistent data in application. So with JDBC, mapping between Java objects and database tables is done manually.

With JDBC, caching is maintained by hand-coding.

#### Hibernate

Hibernate is flexible and powerful ORM solution to map Java classes to database tables. Hibernate itself takes care of this mapping using XML files so developer does not need to write code for this.

Hibernate provides transparent persistence and developer does not need to write code explicitly to map database tables tuples to application objects during interaction with RDBMS.

Hibernate provides a powerful query language Hibernate Query Language (independent from type of database) that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application.

Hibernate provides this mapping itself. The actual mapping between tables and application objects is done in XML files. If there is change in Database or in any table then the only need to change XML file properties.

Hibernate reduces lines of code by maintaining object-table mapping itself and returns result to application in form of Java objects. It relieves programmer from manual handling of persistent data, hence reducing the development time and maintenance cost.

Hibernate, with Transparent Persistence, cache is set to application work space.

In JDBC there is no check that always every user has updated data. This check has to be added by the developer.

Relational tuples are moved to this cache as a result of query. It improves performance if client application reads same data many times for same write. Automatic Transparent Persistence allows the developer to concentrate more on business logic rather than this application code.

Hibernate enables developer to define version type field to application, due to this defined field Hibernate updates version field of database table every time relational tuple is updated in form of Java class object to that table. So if two users retrieve same tuple and then modify it and one user save this modified tuple to database, version is automatically updated for this tuple by Hibernate. When other user tries to save updated tuple to database then it does not allow saving it because this user does not have updated data.

### 32. What are the Collection types in Hibernate ?

- Bag
- Set
- List
- Array
- Map

### 33. What are the ways to express joins in HQL?

HQL provides four ways of expressing (inner and outer) joins:-

- An *implicit* association join
- An ordinary join in the FROM clause
- A fetch join in the FROM clause.
- A *theta-style* join in the WHERE clause.

### 34. Define cascade and inverse option in one-many mapping?

**cascade** - enable operations to cascade to child entities.

`cascade="all|none|save-update|delete|all-delete-orphan"`

**inverse** - mark this collection as the "inverse" end of a bidirectional association.

`inverse="true|false"`

Essentially "inverse" indicates which end of a relationship should be ignored, so when persisting a parent who has a collection of children, should you ask the parent for its list of children, or ask the children who the parents are?

**35. What is Hibernate proxy?**

The proxy attribute enables lazy initialization of persistent instances of the class. Hibernate will initially return CGLIB proxies which implement the named interface. The actual persistent object will be loaded when a method of the proxy is invoked.

**36 .How can Hibernate be configured to access an instance variable directly and not through a setter method ?**

By mapping the property with access="field" in Hibernate metadata. This forces hibernate to bypass the setter method and access the instance variable directly while initializing a newly loaded object.

**37. How can a whole class be mapped as immutable?**

Mark the class as mutable="false" (Default is true)., This specifies that instances of the class are (not) mutable. Immutable classes, may not be updated or deleted by the application.

**38. What is the use of dynamic-insert and dynamic-update attributes in a class mapping?**

Criteria is a simplified API for retrieving entities by composing Criterion objects. This is a very convenient approach for functionality like "search" screens where there is a variable number of conditions to be placed upon the result set.

- dynamic-update (defaults to false): Specifies that UPDATE SQL should be generated at runtime and contain only those columns whose values have changed
- dynamic-insert (defaults to false): Specifies that INSERT SQL should be generated at runtime and contain only the columns whose values are not null.

**39. What do you mean by fetching strategy ?**

A *fetching strategy* is the strategy Hibernate will use for retrieving associated objects if the application needs to navigate the association. Fetch strategies may be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query.

**40. What is automatic dirty checking?**

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.

**41. What is transactional write-behind?**

Hibernate uses a sophisticated algorithm to determine an efficient ordering that avoids database foreign key constraint violations but is still sufficiently predictable to the user. This feature is called transactional write-behind.

**42. What are Callback interfaces?**

Callback interfaces allow the application to receive a notification when something interesting happens to an object—for example, when an object is loaded, saved, or deleted. Hibernate applications don't need to implement these callbacks, but they're useful for implementing certain kinds of generic functionality.

### 43. What are the types of Hibernate instance states ?

Three types of instance states:

- Transient -The instance is not associated with any persistence context
- Persistent -The instance is associated with a persistence context
- Detached -The instance was associated with a persistence context which has been closed – currently not associated

### 44. What are the differences between EJB 3.0 & Hibernate

Hibernate Vs EJB 3.0 :-

Hibernate	EJB 3.0
<b>Session</b> —Cache or collection of loaded objects relating to a single unit of work	<b>Persistence Context</b> -Set of entities that can be managed by a given EntityManager is defined by a persistence unit
<b>XDoclet Annotations</b> used to support Attribute Oriented Programming	<b>Java 5.0 Annotations</b> used to support Attribute Oriented Programming
<b>Defines HQL</b> for expressing queries to the database	<b>Defines EJB QL</b> for expressing queries
<b>Supports Entity Relationships</b> through mapping files and annotations in JavaDoc	<b>Support Entity Relationships</b> through Java 5.0 annotations
<b>Provides a Persistence Manager API</b> exposed via the Session, Query, Criteria, and Transaction API	<b>Provides and Entity Manager Interface</b> for managing CRUD operations for an Entity
<b>Provides callback support</b> through lifecycle, interceptor, and validatable interfaces	<b>Provides callback support</b> through Entity Listener and Callback methods
<b>Entity Relationships are unidirectional.</b> Bidirectional relationships are implemented by two unidirectional relationships	<b>Entity Relationships are bidirectional or unidirectional</b>

### 45. What are the types of inheritance models in Hibernate?

There are three types of inheritance models in Hibernate:

- Table per class hierarchy
- Table per subclass
- Table per concrete class

### What is Hibernate?

- Hibernate is an ORM (Object Relational Mapping) and persistent framework.
- The framework helps to map plain java object to relational database table using xml configuration file.
- The framework helps to perform following things.

  1. Perform basic CURD operations.
  2. Write queries referring to java classes (HQL queries).
  3. Facilities to specify metadata.
  4. Dirty checking, lazy association fetching.

### Why hibernate and how does it help in the programming?

The main advantage of Hibernate (ORM) framework is that it shields developer to write a messy SQL. Apart from that ORM provides following benefits.

1. Improve Productivity of the developer by providing high level object oriented API (e.g. API for easily maintaining the connection to data base, mapping java classes to relational database tables), less java code to write, helps to avoid writing SQL query.
2. Improved performance by providing sophisticated caching, lazy loading and eager loading features.
3. Provide portability, the framework helps to generate database specific SQL for you.

### How will you configure Hibernate?

- To configure hibernate, you need hibernate.cfg.xml or hibernate.properties file and \*.hbm.xml files, all these files are used by configuration class to create sessionFactory, which in turn creates the session instances.
- Session instances are the primary interface for persistence services.
- The hibernate.cfg.xml or hibernate.properties files are used to configure the hibernate service (database connection driver class, database connection URL, connection username, connection password, dialect, mapping resources etc.).
- The \*.hbm.xml files are used for mapping persistent objects to relational database.
- From Java 5 onwards you can configure and map persistent objects through annotations.

### Which settings will be loaded if both hibernate.properties and hibernat.cf.xml files are present in the classpath?

If both hibernate.properties and hibernate.cfg.xml files are present in the classpath then hibernate.cfg.xml file will override the settings found in hibernate.properties. So please make sure that your project should include either hibernate.properties or hibernate.cfg.xml file.

### What are the Core interfaces of Hibernate framework?

There are five core interfaces being used extensively in every Hibernate application. Using these interfaces you can store or retrieve any persistent objects and also control transactions.

1. Session interface
2. SessionFactory interface
3. Configuration interface

- 
- 4. Transaction interface
  - 5. Query and Criteria interfaces

### What is the role of Session interface in Hibernate?

The session interface is the primary interface used in Hibernate Application.

- It is a single threaded short-lived object and represents conversation between application and the persistent store.
- It helps to create query objects to retrieve persistent objects.

- You can get the session object from session factory :

```
Session session = sessionFactory.openSession();
```

- Session Interface role :

1. Wraps a JDBC connection.
2. Factory for Transaction.
3. Holds a mandatory (first-level) cache of persistent objects, used when navigating the object graph or looking up objects by identifier.

### What is the role of SessionFactory?

The application obtains session object from SessionFactory interface. Typically there should be only one sessionFactory for whole application and is loaded during application initialization. The SessionFactory caches generate SQL Statement and other mapping metadata that Hibernate uses at runtime. It also holds cached data that has been read in one unit of work and can be reused in a future unit of work.

You can get the instance of SessionFactory by the configuration object as below :

```
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

### How do you implement one to one relationship in Hibernate with XML mapping?

- For example you have table emp(employee table) and emp\_detail(employee details) and assuming there is a one to one relationship between them. For the above tables you have to create corresponding POJO classes and hbm.xml files.
- So for emp table the java class is Employee.java with property empld and xml file is emp.hbm.xml.
- And for emp\_details the java class is EmployeeDetail.java (properties are name, address etc.) and xml file is empdetail.hbm.xml.
- So the final code will look like as below :

```
package com.test.onetoone.mapping  
public class Employee implements java.io.Serializable  
{  
    private Integer empld;  
    private EmployeeDetail empDetail;  
}  
package com.test.onetoone.mapping  
public class EmployeeDetail implements java.io.Serializable  
{
```

```
private Integer empld;
private Employee emp;
private String name;
private String address;
}

----- emp.hbm.xml -----
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.test.onetoone.mapping.Employee" table="emp">
<id name="empld" type="java.lang.Integer">
<column name="EMP_ID" />
<generator class="identity" />
</id>
<one-to-one name="empDetail" class="com.test.onetoone.mapping.EmployeeDetail"
cascade="save-update"></one-to-one>
</class>
</hibernate-mapping>
```

\*\*\*\*\*empdetails.hbm.xml\*\*\*\*\*

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
<class name="com.test.onetoone.mapping.EmployeeDetail" table="emp_detail"
catalog="mkyongdb">
<id name="stockId" type="java.lang.Integer">
<column name="EMP_ID" />
<generator class="foreign">
<param name="property">emp</param>
</generator>
</id>
<one-to-one name="emp" class="com.test.onetoone.mapping.Employee"
constrained="true"></one-to-one>
<property name="name" type="string">
<column name="EMP_NAME" length="100" not-null="true" />
</property>
<property name="address" type="string">
<column name="EMP_ADDR" not-null="true" />
</property>
</class>
</hibernate-mapping>
```

How do you implement one to one relationship in Hibernate with java annotation?

Taking the same Employee and Employee Details example of the above question.

```
Employee.java
package com.test.onetoone.mapping
import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.OneToOne;
import javax.persistence.Table;
@Entity
@Table(name = "emp")
public class Employee implements java.io.Serializable
{
    private Integer empld;
    private EmployeeDetail empDetail;
    public Employee(){}
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "EMP_ID", unique = true, nullable = false)
    public Integer getEmpld()
    {
        return this.empld;
    }
    public void setEmpld(Integer empld)
    {
        this.empld = empld;
    }
    @OneToOne(fetch = FetchType.LAZY, mappedBy = "emp", cascade = CascadeType.ALL)
    public EmployeeDetail getEmpDetail()
    {
        return this.empDetail;
    }
    public void setEmpDetail(EmployeeDetail empDetail)
    {
        this.empDetail = empDetail;
    }
}
```

**File: EmployeeDetail.java**

```
package com.test.onetoone.mapping
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 javax.persistence.Temporal;
import javax.persistence.TemporalType;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;
@Entity
@Table(name = "emp_detail")
public class EmployeeDetail implements java.io.Serializable
{
    private Integer empld;
    private Employee emp;
    private String name;
    private String address;
    public EmployeeDetail() {}
    public EmployeeDetail(Employee emp, String name, String address)
    {
        this.emp = emp;
        this.name = name;
        this.address = address;
    }
    @GenericGenerator(name = "generator", strategy = "foreign",
parameters = @Parameter(name = "property", value = "emp"))
@Id
@GeneratedValue(generator = "generator")
@Column(name = "EMP_ID", unique = true, nullable = false)
public Integer getEmpld()
{
    return this.empld;
}
public void setEmpld(Integer empld)
{
    this.empld = empld;
}
@OneToOne(fetch = FetchType.LAZY)
@PrimaryKeyJoinColumn
public Employee getEmp()
{
    return this.emp;
}
public void setEmp(Employee emp)
{
    this.emp = emp;
}
@Column(name = "ADDRESS", nullable = false, length = 400)
public String getAddress()
{
```

```

    return this.address;
}
public void setAddress(String address)
{
    this.address = address;
}
@Column(name = "EMP_NAME", nullable = false)
public String getName()
{
    return this.name;
}
public void setName(String name)
{
    this.name = name;
}
}

```

**3. Hibernate Configuration File**

- Puts annotated classes Employee.java and EmployeeDetail.java in your Hibernate configuration file, and also MySQL connection details.

File : hibernate.cfg.xml

```

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

```

<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mytestdb</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">password</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="show_sql">true</property>
<mapping class="com.test.onetoone.mapping.Employee" />
<mapping class="com.test.onetoone.mapping.EmployeeDetail" />
</session-factory>
</hibernate-configuration>
```

**How do you implement one to many relationships in Hibernate?**

- For one to many relationships we can consider the example Author and Books (i.e. one Author can have written many books).
- Below code will help you to understand how you can implement one to many relationship in hibernate.

**File: Author.java**

```

package com.test.one.to.many;
java.util.Set;
public class Author implements java.io.Serializable
```

```
{  
    private Integer authorId;  
    private String authorName;  
    private Set books;  
    // getter and setter  
}
```

**File: Book.java**

```
package com.test.one.to.many;  
public class Book implements java.io.Serializable  
{  
    private Author author;  
    private Integer bookId;  
    private String bookName;  
    // getter and setter  
}
```

**File: Author.hbm.xml**

```
<?xml version="1.0"?>  
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
<class name="com.test.one.to.many.Author" table="author">  
<id name="authorId" type="java.lang.Integer">  
<column name="AUTHOR_ID" />  
<generator class="identity" />  
</id>  
<property name="authorName" type="string">  
<column name="AUTHOR_NAME" length="100" not-null="true" unique="true" />  
</property>  
<set name="books" table="book" inverse="true" lazy="true" fetch="select">  
<key>  
<column name="AUTHOR_ID" not-null="true" />  
</key>  
<one-to-many class="com.test.one.to.many.Book" />  
</set>  
</class>  
</hibernate-mapping>
```

**File: Book.hbm.xml**

```
<?xml version="1.0"?>  
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">  
<hibernate-mapping>  
<class name="com.test.one.to.many.Book" table="book">  
<id name="bookId" type="java.lang.Integer">  
<column name="BOOK_ID" />  
<generator class="identity" />  
</id>  
<many-to-one name="author" class="com.test.one.to.many.Author" fetch="select">
```

```
<column name="AUTHOR_ID" not-null="true" />
</many-to-one>
<property name="bookName" type="string">
<column name="BOOK_NAME" />
</property>
</class>
</hibernate-mapping>
File: hibernate.cfg.xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mytestdb</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">password</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="show_sql">true</property>
<property name="format_sql">true</property>
<mapping resource="com/test/one/to/many/Author.hbm.xml" />
<mapping resource="com/test/one/to/many/Book.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

#### How to implement one to many relationships with Annotation?

- You can implement one to many relationship with the help Annotation also.
- Below code will help you to understand the one to many relationship with java Annotation.

#### File : Author.java

```
package com.test.one.to.many;
import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.OneToMany;
import javax.persistence.Table;
@Entity
@Table(name = "Author")
public class Author implements java.io.Serializable
{
    private Integer authorId;
    private String authorName;
    private Set<Book> books;
```

```
// getter and setter
public Author() {}
@Id
@GeneratedValue(strategy = IDENTITY)
@Column(name = "AUTHOR_ID", unique = true, nullable = false)
public Integer getAuthorId()
{
    return this.authorId;
}
public void setAuthorId(Integer authorId)
{
    this.authorId = authorId;
}
@Column(name = "AUTHOR_NAME", nullable = false, length = 100)
public String getAuthorName()
{
    return this.authorName;
}
public void setAuthorName(String authorName)
{
    this.authorName = authorName;
}
@OneToMany(fetch = FetchType.LAZY, mappedBy = "author")
public Set<Book> getBooks()
{
    return this.books;
}
public void setBooks(Set<Book> books)
{
    this.books = books;
}
```

**File : Book.java**

```
package com.test.one.to.many;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import static javax.persistence.GenerationType.IDENTITY;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
import javax.persistence.UniqueConstraint;
```

@Entity

```
@Table(name = "book")
public class Book implements java.io.Serializable
{
    private Author author;
    private Integer bookId;
    private String bookName;
    public Book() {}
    @Id
    @GeneratedValue(strategy = IDENTITY)
    @Column(name = "BOOK_ID", unique = true, nullable = false)
    public Integer getBookId()
    {
        return this.bookId;
    }
    public void setBookId(Integer bookId)
    {
        this.bookId = bookId;
    }
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "AUTHOR_ID", nullable = false)
    public Author getAuthor()
    {
        return this.author;
    }
    public void setAuthor(Author author)
    {
        this.author = author;
    }
    @Column(name = "BOOK_NAME", length = 400, nullable=false)
    public Float getBookName()
    {
        return this.bookName;
    }
    public void setBookName(String bookName)
    {
        this.bookName = bookName;
    }
}
```

### 3. Hibernate Configuration File

- Puts annotated classes Author.java and Book.java in hibernate.cfg.xml like this :

File : hibernate.cfg.xml

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

```
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mytestdb</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">password</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="show_sql">true</property>
<mapping class="com.test.one.to.many.Author" />
<mapping class="com.test.one.to.many.Book" />
</session-factory>
</hibernate-configuration>
```

How will you integrate Hibernate with spring framework?

- To integrate hibernate with spring framework, the hibernate configuration will go in spring configuration file.

- The configuration file will look like as below :

```
<beans>
<bean id="propertyConfigurer"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"
p:location="/WEB-INF/jdbc.properties"></bean>
<!--jdbc.properties database related properties --?>
```

```
<bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"
p:driverClassName="${jdbc.driverClassName}"
p:url="${jdbc.databaseurl}" p:username="${jdbc.username}"
p:password="${jdbc.password}"></bean>
```

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
<property name="dataSource" ref="dataSource"></property>
<property name="configLocation">
<value>classpath:hibernate.cfg.xml</value>
</property>
<property name="configurationClass">
<value>org.hibernate.cfg.AnnotationConfiguration</value>
</property>
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">${jdbc.dialect}</prop>
<prop key="hibernate.show_sql">true</prop>
</props>
</property>
</bean>
```

```
<bean id="employeeDAO" class="com.test.dao.EmployeeDaoImpl"></bean>
```

```
<bean id="employeeManager" class="com.test.service.EmployeeManagerImpl"></bean>

<tx:annotation-driven />

<bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
<property name="sessionFactory" ref="sessionFactory"></property>
</bean>
</beans>
```

### What are the Collection types in Hibernate?

1. Bag
2. Set
3. List
4. Array
5. Map

### What is HibernateTemplate?

The spring framework provides

HibernateTemplate(org.springframework.orm.hibernate.HibernateTemplate) which is kind of helper class and provides following benefits.

- HibernateTemplate class simplifies interaction with Hibernate session.
- Common functions are simplified to single method calls.
- Sessions are automatically closed.
- Exception are automatically caught and converted to runtime exceptions.

### What is the difference between load() and get() method?

#### load() :

- Use load() method only when you are sure that object you want to read already exists.
- If unique Id of an object does not exists in database then load() method will throw an exception.
- load() method return proxy object default and database won't be hit until the proxy is first invoked.

#### get() :

- Use get() method when you are not sure about the object existance in the database.
- If object does not exists in the database, the get() method will return null.
- get() method will hit database immediately.

### What is lazy fetching in Hibernate?

In Hibernate Lazy fetching is associated with child objects loading for its parents. Through Hibernate mapping file (.hbm.xml) you can specified the selection of loading child objects. By

---

default Hibernate does not load child objects. Lazy=true means not to load the child objects.

What is the difference between merge and update method?

Use update() method when you are sure that session does not contain an already persistent instance with the same identifier, and merge() if you want to merge your modifications at any time without consideration of the state of the session.

How do you define sequence generated primary key in hibernate?

Using <generator> tag.

- Example :

```
<id column="CUST_ID" name="id" type="java.lang.Long">
<generator class="sequence">
<param name="table">SEQUENCE_NAME</param>
<generator>
</id>
```

What are the different types of caches in Hibernate?

Hibernate uses two different type of caches for objects: first-level cache and second-level cache. First level of cache is associated with Session object, while second-level of cache is associated with the SessionFactory object. By default, Hibernate uses first-level of cache on a per-transaction basis. Hibernate mainly use this cache to reduce the number of SQL queries it needs to generate within a given transaction.

What do you mean by Named – SQL query?

- Named SQL queries are defined in the mapping xml document and called wherever required.

- Example :

```
<sql-query name = "empdetails">
<return alias="emp" class="com.test.Employee"/>
SELECT emp.EMP_ID AS {emp.empid},
emp.EMP_ADDRESS AS {emp.address},
emp.EMP_NAME AS {emp.name}
FROM Employee EMP WHERE emp.NAME LIKE :name
</sql-query>
```

Invoke Named Query :

```
List people = session.getNamedQuery("empdetails")
.setString("Deepak", name)
.setMaxResults(50)
.list();
```

How does Hibernate useful in mapping object and its relations?

- Hibernate is a framework for Java that is used to map the object and its relations.
- It is used to build persistent objects using the terminologies of JAVA.
- It uses the following properties to perform the function of mapping :

1. **Association** : Where the one object can be associated with another object of the same or different class.
2. **Inheritance** : It is a property of inheriting the properties and features of the parent class.
3. **Polymorphism** : It is a feature that allows more than one data type to be handled by a single function.
4. **Composition** : It is used to combine the simple data types into complex data types.

#### What is the function performed by code generator in Hibernate?

Code generator is a framework that allow simple and easy mapping of objects using pre-defined tags. It follows Model driven architecture which uses models of the objects separately and link them with each other. Code generator adds attributes to the source code. It uses the java-doc tags to generate the hibernate mapping. It is also used to generate the database definition language from the source code to map the objects correctly. It allows easy mapping of objects from an existing database and allow it to be connected to some different database model as well.

#### What is the need of hibernate tools when ORM tools can be used?

The use of hibernate tools is to allow the programmer to code efficiently and saves them from being involved with the SQL code directly. It is used to increase the productivity of the work and uses high level object oriented APIs to increase the efficiency and performance of the work. It allows the programming to be more objects oriented and the code in Java has to be written less. It also save programmers time to write the SQL code as there is no need to write the SQL code in it. These tools provide very good caching system that keeps all the files and they provide it when it is required. It allows easy maintenance of the overall system and the code that is being written in it.

#### What are the tags that are importantly used in Hibernate configuration file?

Hibernate configuration file is the most important file. The configuration file consists of many tags and they are as follows :

**DTD** : This is also called as Document type definition and is written as :

```
<!DOCTYPE hibernate-configuration PUBLIC  
"-//Hibernate/Hibernate Configuration DTD/EN"  
" http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

**JDBC connection configuration tag** : JDBC stands for Java Database connectivity tags that gives the driver name and the database related information and represented as :

```
<property name= "hibernate.connection.driver_class">org.hsqldb.jdbcDriver</property>
```

**SQL variant** : It specifies the SQL statement to generate the particular property in the database. It is given as :

```
<property name= "dialect">org.hibernate.dialect.HSQLDialect</property>
```

**Conn pool** : This tag is very important to set the size of the conn pool for the correct connection. It is given as :

```
<property name= "hibernate.connection.pool_size">10</property>
```

**Property to generate the database schema** : This property tag generates the database schema automatically. It is given as :

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

**Mapping files** : These files are important to map the resources of the same or different objects. It

is given as :

```
<mapping resource= "Event.hbm.xml">
```

### What is the role played by Session interface in Hibernate?

Session interface is the primary interface used by Hibernate. It is the important interface for the applications that uses single thread, and a short lived object representing a conversation between application and the objects that are stored. This object allows creating query objects that can be used to call persistent objects from the database. This interface allows the wrapping of JDBC connection and keeps the transactions at one place. It holds the persistent objects in the cache for easy navigation and to increase the performance of the overall system. The interface is represented as :

```
Session session = sessionFactory.openSession();
```

### What is the role played by SessionFactory interface in Hibernate?

The session instance is being obtained by the application from SessionFactory. It keeps the session instances at one place for easy access and dynamic allocation of the resources. There is always a single SessionFactory created for the complete application. This is created during the application initialization. It creates the cache that is used to generate the SQL statements and metadata information about the mapping of the Hibernate functions that gets used in runtime. It holds the data that has been cached and then read into it. It focuses on reusability of the work and the functions that are stored in the SessionFactory. This is represented as :

```
SessionFactory sessionFactory = configuration.buildSessionFactory();
```

### What is the process of communication between Hibernate with RDBMS?

To create the communication between the Hibernate and RDBMS there are some steps that are involved in the process. These are as follows :

- To create the communication first loading of the Hibernate configuration file takes place. When this file is loaded then a configuration object gets created. This object loads the .hbm files used to map the objects automatically.
- A session gets created by taking the interface from the SessionFactory from the configuration object.
- A HQL query is being creation after taking the session from the session factory and starting the session for the user.
- In the end an execution of the query gets listed that contain the Java Objects of the file. This allows the communication being done between different objects of Hibernate and RDBMS.

### What is the purpose of HQL in communication between Hibernate and RDBMS?

HQL stands for Hibernate Query language and is used to set the communication between the Hibernate and RDBMS. It is a powerful language that uses flexible mechanism for query, store, and update to take the objects from the database. It uses the objects for making a connection between the Hibernate and the database. It is an extension to the SQL that uses an Object oriented paradigm to communicate and retrieve the data from the database. It uses inheritance, polymorphism and abstraction features to handle the request and respond to the queries coming

from the user.

### Write the code that maps the Java Object with the database tables?

To map the Java Object with the database tables there is a need to write the code of Java domain objects. Java domain objects use the methods setter and getter to retrieve the Java objects and maps it accordingly. The file that is used to be written to map java class or object to the database table is hbm.xml. This file also maps the database columns to Java class variables. The code that is being written to perform the above mentioned operation is given below :

```
<hibernate-mapping>
<class name="com.test.User" table="user">
<property column="USER_NAME" length="255"
name="userName" not-null="true" type="java.lang.String"/>
<property column="USER_PASSWORD" length="255"
name="userPassword" not-null="true" type="java.lang.String"/>
</class>
</hibernate-mapping>
```

### What is the purpose of using HibernateTemplate?

HibernateTemplate is a spring template that is used as a class to simplify the interaction with the Hibernate Session. HibernateTemplate includes simplified single method calls that are used to close the sessions automatically. It includes the exception handler that catches the exception automatically and converted it in runtime. There are classes used in Hibernate that provides different methods for query and to retrieve the data from the database. The helper class that is provided is org.springframework.orm.hibernate. HibernateTemplate is used to check and convert the HibernateExceptions into unchecked DataAccessExceptions.

### How components are used in Hibernate?

Components allow the object to be saved as a value and not as a reference. It is used to be saved directly without the need to declare the properties of the interfaces or identifier in the application. Components in Hibernate define the empty constructor in which the references that are shared are not supported. The components class will be defined as :

```
public class hello
{
    private String a;
    private int b;
    private string c;
    //write your own code here
}
```

### Write a code to do the component mapping in Hibernate.

To do the component mapping there is a requirement for two component classes that has to be made to establish the connection between them. So, the first component class will be like this :

```
public class hello
```

```
{  
    private String a;  
    private int b;  
    private string c;  
    //write your own code here  
}
```

Another class will be written like this :

```
public class comp  
{  
    // properties  
    private String a;  
    //write get and set methods  
    //write your own code here  
}
```

The class that will be used to map the two components are written as :

```
<class name= "no.uio.inf.model.table=hello">  
<!—other properties-->  
<component name= "comp">  
<property name = "a"/>  
<property name = "b"/>  
<property name = "c"/>  
</component>  
</class>
```

### Why Hibernate is preferred over JDBC?

The preference of Hibernate is more due to the transparency it provides to the programmer for writing their code. It doesn't allow the programmer to write explicit code to map the database tables to the application objects that are used during the interaction with RDBMS. It provides an advanced ORM solution to map the Java class files to the database tables for easy access and retrieval of the data. Hibernate uses Hibernate Query Language that is independent of the type of database used. It allows polymorphic queries to be retrieved and selects the way to perform the database manipulation tasks. Hibernate allow the storing of large amount of data in small files that can be easily accessed and the performance can be faster. It also provides the actually mapping to be performed between the tables and objects in the XML files.

### What is the use of Hibernate proxy in Hibernate?

The proxies are easy to create and it is created dynamically by sub-classing the objects at runtime. This sub-class consists of all the parent methods and allows easy accessing of them. The proxy allows the loading of this object in real time from the database and calls the methods from there. Proxy is used as a mechanism in Hibernate allowing breaking of the objects in the database in smaller parts that can easily be accessed and use the memory efficiently. It is easy for a class to be mapped to a proxy than to a table. When there is a call load on a session then the value that is returned is the proxy. The proxy consists of the methods to load the data. The sample code to call the proxy is given as :

```
<class name="a" proxy="com.ch01.a">  
</class>
```

**What is the difference between managed and non-managed environment in Hibernate?**

It is important to configure the Hibernate file properly so that it will be easy to manage and access the database. The difference that has to be known for the Hibernate is between Managed and non-managed environment :

- Managed environment allows the pooled resources of database connections to be loaded on run time. It also allows the transaction to be taken place smoothly in the specified boundaries, whereas non-managed environment provides the basic concurrency management using the method of thread pooling.
- Managed environment allows automatic transaction of the resources, whereas non-managed environment doesn't provide any automatic transaction of the resources.
- Managed environment provides automatic management of security infrastructure, whereas non-managed environment doesn't provide this.

**What are the different types of interfaces present in Hibernate?**

Hibernate consists of many interfaces to keep everything dynamic and up-to-date. The interfaces are as follows :

- Configuration interface: this interface includes the configuration object that is used to configure and bootstrap the Hibernate applications. It allows the need to map the documents and other related properties. It also allows the creation of the SessionFactory.
- Transaction interface: is an optional interface API that is used to manage the transaction of the infrastructure code. It allows the portability of the application that is executed in different environments and containers.
- Query and Criteria interfaces: it allows the performance of different queries that is against the database and it controls the execution of the queries as well. The queries are written either in HQL or SQL that depends on the database being used.

**Why are callback interfaces useful in Hibernate?**

Callback interfaces are useful as it allows the application to receive important notification about the objects that are in execution phase. It includes the objects that are loaded, saved or deleted from the Hibernate. Callback interfaces are very important to implement the general functionality like audit records, etc. It sends a notification when any object event occurs. It allows the programmer to get the error information or exception handling can be done in a better way to notify the user on run time in case of any problem in the programming code.

**What is the importance of Lifecycle and validatable interface in Hibernate?**

Lifecycle and validatable interface allows the object that is persistent to react to certain events that occurs during their own persistence lifecycle. The lifecycle is managed by the CRUD operations which handles the object lifecycle as well. Hibernate uses ORM solution that are similar in functionality of the callback interface. It allows easy portable code for the transaction to be

saved and persistent classes and objects can be validated and sent over for implementation and use.

What are the different types of extension interface that is present in Hibernate?

Extension interface allows easy implementation of other interfaces according to the user's requirements. It allows different functionality to be provided when the built in functionality is not sufficient for the users. The extensions that are provided for the use is as follows :

**ProxyFactory interface** : It is used to create proxies to help in maintaining the mapping of the objects.

**ConnectionProvider interface** : It is used to provide the JDBC connection management for easy mapping of the object with the database.

**TransactionFactory interface** : It is used to provide the transaction management.

**TransactionManagementLookup interface** : It is used to provide the way to look up the transactions that is being managed by transaction management interface.

**Cache interface** : It is used to provide the caching techniques and strategies for the objects to speed up the process.

**IdentifierGenerator interface** : It is used to provide a way to generate the keys like primary key to uniquely identify the objects.

#### What is JDBC?

JDBC stands for Java Database Connectivity and provides a set of Java API for accessing the relational databases from Java program. These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database.

#### What is ORM?

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C# etc.

What are the advantages of ORM over JDBC?

An ORM system has following advantages over plain JDBC

#### S.N. Advantages

- 1 Lets business code access objects rather than DB tables.
- 2 Hides details of SQL queries from OO logic.
- 3 Based on JDBC 'under the hood'
- 4 No need to deal with the database implementation.
- 5 Entities based on business concepts rather than database structure.
- 6 Transaction management and automatic key generation.

7 Fast development of application.

Name some of the ORM frameworks based on JAVA.

There are several persistent frameworks and ORM options in Java.

- Enterprise JavaBeans Entity Beans
- Java Data Objects
- Castor
- TopLink
- Spring DAO
- Hibernate

**What is Hibernate?**

Hibernate is an Object-Relational Mapping(ORM) solution for JAVA and it raised as an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.

Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

**What are the advantages of using Hibernate?**

Following are the advantages of using Hibernate.

- Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
- Provides simple APIs for storing and retrieving Java objects directly to and from the database.
- If there is change in Database or in any table then the only need to change XML file properties.
- Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
- Hibernate does not require an application server to operate.
- Manipulates Complex associations of objects of your database.
- Minimize database access with smart fetching strategies.
- Provides Simple querying of data.

Name some of the databases that hibernate supports.

Hibernate supports almost all the major RDBMS. Following is list of few of the database engines supported by Hibernate.

- HSQL Database Engine
- DB2/NT
- MySQL
- PostgreSQL
- FrontBase
- Oracle
- Microsoft SQL Server Database
- Sybase SQL Server
- Informix Dynamic Server

Name some of the java based tools/frameworks that supports hibernate integration.

Hibernate supports a variety of other technologies, including the following:

- XDoclet Spring
- J2EE
- Eclipse plug-ins
- Maven

What are the key components/objects of hibernate?

Following are the key components/objects of Hibernate:

- **Configuration** - Represents a configuration or properties file required by the Hibernate.
- **SessionFactory** - Configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated.
- **Session** - Used to get a physical connection with a database.
- **Transaction** - Represents a unit of work with the database and most of the RDBMS supports transaction functionality.
- **Query** - Uses SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
- **Criteria** - Used to create and execute object oriented criteria queries to retrieve objects.

What are the two key components of a hibernate configuration object?

The Configuration object provides two keys components:

- **Database Connection:** This is handled through one or more configuration files supported by

Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.

- **Class Mapping Setup**

This component creates the connection between the Java classes and database tables.

#### **What is a configuration object in hibernate?**

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

#### **What is a SessionFactory in hibernate?**

Configuration object is used to create a SessionFactory object which inturn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

#### **What is Session in hibernate?**

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

#### **What is Transaction in hibernate?**

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

#### **What is Query in hibernate?**

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

#### **What is Criteria in hibernate?**

Criteria object are used to create and execute object oriented criteria queries to retrieve objects.

Name some of the properties you would require to configure for a databases in a standalone situation.

S.N.	Properties and Description
------	----------------------------

**hibernate.dialect**

1 This property makes Hibernate generate the appropriate SQL for the chosen database.

**hibernate.connection.driver\_class**

2 The JDBC driver class.

**hibernate.connection.url**

3 The JDBC URL to the database instance.

**hibernate.connection.username**

4 The database username.

**hibernate.connection.password**

5 The database password.

**hibernate.connection.pool\_size**

6 Limits the number of connections waiting in the Hibernate database connection pool.

**hibernate.connection.autocommit**

7 Allows autocommit mode to be used for the JDBC connection.

What are the three states of a persistent entity at a given point in time?

Instances may exist in one of the following three states at a given point in time:

- **transient:** A new instance of a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.
- **persistent:** You can make a transient instance persistent by associating it with a Session. A persistent instance has a representation in the database, an identifier value and is associated with a Session.
- **detached:** Once we close the Hibernate Session, the persistent instance will become a detached instance.

**What is the purpose of Session.beginTransaction() method?**

Session.beginTransaction method begins a unit of work and returns the associated Transaction object.

**Which method is used to add a criteria to a query?**

Session.createCriteria creates a new Criteria instance, for the given entity class, or a superclass of an entity class.

**Which method is used to create a HQL query?**

Session.createQuery creates a new instance of Query for the given HQL query string.

**Which method is used to create a SQL query?**

---

Session.createSQLQuery creates a new instance of SQLQuery for the given SQL query string.

**Which method is used to remove a persistent instance from the datastore?**

Session.delete removes a persistent instance from the datastore.

**Which method is used to get a persistent instance from the datastore?**

Session.get returns the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.

**Which method is used to re-read the state of the given instance from the underlying database?**

Session.refresh re-reads the state of the given instance from the underlying database.

**Which method is used to save the state of the given instance from the underlying database?**

Session.save saves the state of the given instance from the underlying database.

**Which method is used to update the state of the given instance from the underlying database?**

Session.update updates the state of the given instance from the underlying database.

**Which method is used to save or update the state of the given instance from the underlying database?**

Session.saveOrUpdate either saves(Object) or updates(Object) the given instance.

**What are persistent classes in hibernate?**

Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate.

What are the best practices that hibernate recommends for persistent classes.

There are following main rules of persistent classes, however, none of these rules are hard requirements.

- All Java classes that will be persisted need a default constructor.
- All classes should contain an ID in order to allow easy identification of your objects within Hibernate and the database. This property maps to the primary key column of a database table.
- All attributes that will be persisted should be declared private and have **getXXX** and **setXXX** methods defined in the JavaBean style.
- A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.
- All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

**Where Object/relational mappings are defined in hibernate?**

An Object/relational mappings are usually defined in an XML document. This mapping file instructs Hibernate how to map the defined class or classes to the database tables. We should save the mapping document in a file with the format <classname>.hbm.xml.

**What is root node of hbm.xml?**

The mapping document is an XML document having <hibernate-mapping> as the root element which contains all the <class> elements.

**Which element of hbm.xml defines a specific mappings from a Java classes to the database tables?**

The <class> elements are used to define specific mappings from a Java classes to the database tables. The Java class name is specified using the **name** attribute of the class element and the database table name is specified using the **table** attribute.

**Which element of hbm.xml defines maps the unique ID attribute in class to the primary key of the database table?**

The <id> element maps the unique ID attribute in class to the primary key of the database table. The **name** attribute of the id element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

**Which element of hbm.xml is used to automatically generate the primary key values?**

The <generator> element within the id element is used to automatically generate the primary key values. Set the **class** attribute of the generator element is set to **native** to let hibernate pick up either **identity**, **sequence** or **hilo** algorithm to create primary key depending upon the capabilities of the underlying database.

**Which element of hbm.xml is used to map a Java class property to a column in the database table?**

The <property> element is used to map a Java class property to a column in the database table. The **name** attribute of the element refers to the property in the class and the **column** attribute refers to the column in the database table. The **type** attribute holds the hibernate mapping type, this mapping types will convert from Java to SQL data type.

**Which element of hbm.xml is used to map a java.util.Set property in hibernate?**

This is mapped with a <set> element and initialized with java.util.HashSet.

**Which element of hbm.xml is used to map a java.util.SortedSet property in hibernate?**

This is mapped with a <set> element and initialized with java.util.TreeSet. The sort attribute can be set to either a comparator or natural ordering.

**Which element of hbm.xml is used to map a java.util.List property in hibernate?**

This is mapped with a <list> element and initialized with java.util.ArrayList.

**Which element of hbm.xml is used to map a java.util.Collection property in hibernate?**

This is mapped with a <bag> or <ibag> element and initialized with java.util.ArrayList.

**Which element of hbm.xml is used to map a java.util.Map property in hibernate?**

This is mapped with a <map> element and initialized with java.util.HashMap.

**Which element of hbm.xml is used to map a java.util.SortedMap property in hibernate?**

This is mapped with a <map> element and initialized with java.util.TreeMap. The sort attribute can be set to either a comparator or natural ordering.

**What is many-to-one association?**

---

A many-to-one association is the most common kind of association where an Object can be associated with multiple objects. For example a same address object can be associated with multiple employee objects.

<many-to-one> element is used to define many-to-one association. The name attribute is set to the defined variable in the parent class. The column attribute is used to set the column name in the parent table.

#### **What is one-to-one association?**

A one-to-one association is similar to many-to-one association with a difference that the column will be set as unique. For example an address object can be associated with a single employee object.

<many-to-one> element is used to define one-to-one association. The name attribute is set to the defined variable in the parent class. The column attribute is used to set the column name in the parent table which is set to unique so that only one object can be associated with an other object.

#### **What is one-to-many association?**

In One-to-Many mapping association, an object can be can be associated with multiple objects. For example Employee object relates to many Certificate objects.

A One-to-Many mapping can be implemented using a Set java collection that does not contain any duplicate element.

<one-to-many> element of set element indicates that one object relates to many other objects.

#### **What is many-to-many association?**

A Many-to-Many mapping can be implemented using a Set java collection that does not contain any duplicate element.

<many-to-many> element indicates that one object relates to many other objects and column attributes are used to link intermediate column.

#### **Is SessionFactory a thread-safe object?**

Yes, SessionFactory is a thread-safe and can be accessed by multiple threads simultaneously.

#### **Is Session a thread-safe object?**

No, Session is not thread-safe.

#### **What is the difference between save() and persist() methods of session object?**

session.save saves the object and returns the id of the instance whereas persist do not return anything after saving the instance.

#### **What is the difference between get() and load() methods of session object?**

There are following differences between get() and load() methods.

- get() returns null if no data is present whereas load throws ObjectNotFoundException exception in such case.

- `get()` always hits the database whereas `load()` method doesn't hit the database.
- `get()` returns actual object whereas `load()` returns proxy object.
- A central feature of Hibernate, proxies, depends upon the persistent class being either non-final, or the implementation of an interface that declares all public methods.
- All classes that do not extend or implement some specialized classes and interfaces required by the EJB framework.

### **What is lazy loading?**

Lazy loading is a technique in which objects are loaded on demand basis. Since Hibernate 3, lazy loading is by default, enabled so that child objects are not loaded when parent is loaded.

### **What is HQL?**

HQL stands for Hibernate Query Language. It takes java objects in the same way as SQL takes tables. HQL is a Object Oriented Query language and is database independent.

### **What is first level cache in hibernate?**

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass. The Session object keeps an object under its own power before committing it to the database.

### **What is second level cache in hibernate?**

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second-level cache can be configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions.

### **What is Query level cache in hibernate?**

Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache.

This is an optional feature and requires two additional physical cache regions that hold the cached query results and the timestamps when a table was last updated. This is only useful for queries that are run frequently with the same parameters.

## **Hibernate 4.x Features**

### I. Provide support for initial multi-tenancy

Multi-tenancy is the ability to virtually partition clients/customers (also called tenants) in a big enterprise application instead of having all data in a shared space. This principle allows for improvements in management, monitoring and even security and is very useful for big service providers. Companies that offer cloud infrastructures can benefit from multi-tenancy as well. There are several ways to implement this principle which include:

1. A different database and/or schema for each client/tenant
2. The same database/schema for all clients but with an extra column (e.g. `tenant_id`) in all tables that can be used to filter data

Hibernate supports the first method in the 4.0 release. Support for the second method (i.e. discriminator multi-tenancy) is planned in the next version.(May be Hibernate 5.0 or 5.1)

## II. Working with Service Registry

Initial move to ServiceRegistry. Not all "services" have been migrated to this model yet. The main ones (JDBC and transaction stuff) as well as lower level one (classloading and such) have been migrated. The rest will be moved during Alpha2 development.

## III. Improvised ByteCode

Continued work on new bytecode enhancement support within Hibernate, adding support for inline dirty checking

## IV. Package Re-Structure

Some packages are restructured in order to identify the classes which are intended as

1. public API, which are fully expected to be used in application code.
2. internal implementation details, which are only intended for Hibernate use.
3. SPI contracts, which define
  - extension contracts
  - contracts with Hibernate internals exposed to these extensions

**Example :** org.hibernate.dialect.resolver.DialectResolver->  
org.hibernate.service.jdbc.dialect.spi.DialectResolver

## V. Deprecated Classes and Interfaces are Removed

1. org.hibernate.classic.Session
2. org.hibernate.classic.Validatable
3. org.hibernate.classic.ValidationException

## VI. Deprecated Methods have been removed

1. Deprecated methods that have been removed:
  1. References to org.hibernate.type.AbstractSingleColumnStandardBasicType and org.hibernate.type.SingleColumnType methods should be changed as indicated:
    1. nullSafeGet(ResultSet rs, String name) should be changed to  
nullSafeGet(ResultSet rs, String name, SessionImplementor session)
    2. get(ResultSet rs, String name) should be changed to  
get(ResultSet rs, String name, SessionImplementor session)
    3. nullSafeSet(PreparedStatement st, T value, int index) should be changed to  
nullSafeSet(PreparedStatement st, Object value, int index,  
SessionImplementor session)
    4. set(PreparedStatement st, T value, int index) should be changed to  
set(PreparedStatement st, T value, int index, SessionImplementor session)
  2. References to org.hibernate.usertype.UserType methods should be changed as indicated:
    1. nullSafeGet(ResultSet rs, String[] names, Object owner) should be changed

- to  
`nullSafeGet(ResultSet rs, String[] names, SessionImplementor session, Object owner)`
2. `nullSafeSet(PreparedStatement st, Object value, int index)` should be changed to `nullSafeSet(PreparedStatement st, Object value, int index, SessionImplementor session)`
  3. `Session.reconnect()` - manual disconnection and reconnection is now only supported for user-supplied-connection scenarios (JDBC Connection passed in while opening the Session)
  4. `Session.connection()` - use `Session.doWork()`, `Session.doReturningWork()` or `Session.sessionWithOptions()...openSession()` as replacement depending on need
  5. Most of the overloaded `SessionFactory.openSession` methods. Use `SessionFactory.withOptions()...openSession()` instead

## VII. Change in Configuring Listener

`hibernate.cfg.xml` no longer supported as means of specifying listeners. New approach involves using an `org.hibernate.integrator.spi.Integrator` which works based on "service discovery".

## VIII. Logging Framework changed to JBoss Logging

Developers wanted to have I18N support in the logging. Presently, JBoss Logging is the only logging library with full i18n support including parameterization. JBoss Logging also supports i18n in exception messages through a common approach.

## VIII. Added JPA 2.1 specification support which includes

- 1) Support for stored procedures. Now you can call stored procedure in two ways
  - A) Hibernate way
  - B) JPA way in your hibernate App

```
@PersistenceContext
EntityManager entityManager;
```

```
StoredProcedureQuery query = entityManager.createStoredProcedureQuery(
  "getEmployeeDetails", Employee.class );
query.registerStoredProcedureParameter( "salary", String.class, ParameterMode.IN );
query.setParameter( "salary", "9000" );
List salGT10000 = query.getResultList();
```

- 2) `CriteriaUpdate` and `CriteriaDelete` allow definition and execution of UPDATE and DELETE queries in Criterion Style
- 3) Entity listeners can now take advantage of dependency injection through CDI.(Context and Dependency Injection)
- 4) `AttributeConverters`, to apply conversions on beans
- 5) sic values between their database representation and their representation in your domain model. similar to Hibernate's Type

**Some Other Changes**

1. package org.hibernate.connection was removed
2. Hibernate.INTEGER and other similar instances were removed and now IntegerType.INSTANCE should be used; another example is Hibernate.BINARY -> BitaryType.INSTANCE and etc.)
3. org.hibernate.util.EqualsHelper moved to internal package org.hibernate.internal.util.compare.EqualsHelper
4. SessionImplementor have lost getEntityMode() and to get it I use SessionImplementor.getEntityPersister().getEntityMode()
5. org.hibernate.persister.entity.EntityPersister got new methods to implement
6. EntityMode.MAP was removed
7. Hibernate.entity(Class) was removed (didn't find really good way to migrate it to new version)

org.hibernate.jdbc.BatcherFactory, Batcher, and their implementations have been replaced by org.hibernate.engine.jdbc.batch.spi.BatchBuilder and Batch, with default implementations in org.hibernate.engine.jdbc.batch.internal. You can override the default BatchBuilder by defining the "hibernate.jdbc.batch.builder" property as the name of a BatchBuilder implementation, or by providing a BatchBuilder in a custom ServiceRegistry.

## Hibernate 5 ORM Features

Hibernate 5 is the major release of Hibernate framework, which provides the support for JDK 8 and Special database. In this article we are going to discuss all these features of Hibernate ORM 5.

In this tutorial series we will teach you Hibernate 5 with articles, tips and example codes. We have also provides you the necessary video tutorials of Hibernate 5 to explains the source code and examples projects.

RedHat is the company developing the Hibernate framework, Hibernate 5 was released on 20th August, 2015 with many new features and updated API. This release of Hibernate 5 perhaps a major release.

Let's see the features of Hibernate ORM 5:

### **Bootstrap**

New Bootstrap comes with release of Hibernate ORM. New bootstrap mechanism of the Hibernate SessionFactory has been introduced with many features. Although method of creating SessionFactory used in Hibernate 4 is still compatible.

### **Modularity**

Hibernate 5 supports following:

- Wildfly
- OSGi - The OSGi stands for Open Services Gateway initiative, which is an specification for dynamic, modularized system. Hibernate 5 supports OSGi which provides the capability of installing, activating, deactivating, and uninstalling during runtime, without requiring a system restart.

### **Why modularize?**

- Extensions
- ORM Integration

### **Java 8 Support**

Support of Java 8 in Hibernate 5 is one of the most important changes and it provides the support for Java 8 Date and Time API. To include this feature in your project you have to add hibernate-java8.jar file into your project.

If you are using maven then include following dependency:

```
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-java8</artifactId>
    <version>5.1.0.Final</version>
</dependency>
```

## New Types

- Date and Time API

Here are the new classes introduced:

DATE: java.time.LocalDate

TIME: java.time.LocalTime, java.time.OffsetTime

TIMESTAMP: java.time.Instant, java.time.LocalDateTime, java.time.OffsetDateTime and  
java.time.ZoneDateTime

## Java 5, 6 and 7

- Generics
- AutoCloseable

## Bytecode Augmentation

It comes with following features:

- Smarter change detection
- Better Maven and Gradle support
- Enhanced Dirty checking
- Bidirectional relationship management

## AttributeConverter

Hibernate 5 also supports AttributeConverter:

- Enums without @Enumerated
- Invoked on null values
- Hibernate Envers integration
- @ElementCollection
- @MayKey
- @GeneratedValue(strategy=AUTO)  
UUID, custom strategy
- Naming strategy  
physical and implicit

## Hibernate Search

Hibernate 5 comes with support for full-text search through Lucene 5. So, developers can use the new features of Lucene full-text search engine in their applications.

## Full-text Search support:

- Lucene 5
- Item level queries
- Cluster - It supports Master/slave cluster

## New Features of Hibernate 5

### New features of Hibernate 5 are:

- Lucene 5
- Significant performance improvements
- ElasticSearch backend

### Hibernate OGM(Object/Grid Mapper)

This is another major feature of Hibernate 5 as through OGM it provides the ability of persistence to the NoSQL Database. In BigData environment Hibernate 5 can be used to interact with the NoSQL Databases such as Redis, Cassandra and others.

Hibernate provides the JPA(Java Persistence) support for NoSQL databases. It supports:

- MongoDB(Fongo)
- Infinispan
- Neo4J(remote and embedded)
- Redis
- EhCache
- Cassandra
- CouchDB

### Hibernate Envers

The Envers project of Hibernate enables the developer to audit the persistent classes in the application. The aim of the Envers project is to provide the ability to developers for auditing of persistent classes.

Developers will be able to use the Envers feature in application by just adding @Audited annotation in the persistence class. The Envers project will create a table for each entity which is being audited. This table hold the audit history of the changes which is made to the entity. It also allows the developer to query the historical data with ease.