

WWW.tutorial4us.com



College
Projects

```
void main()  
{  
  int no;  
  clrscr();  
  printf("%d\n", no);  
}
```

Basic
Program

C
Tutorial



JAVA
Programming



Tutorial4Us

Tutorials for Beginners



Tutorial4Us

Tutorials for Beginners



Tutorial4Us

Tutorials for Beginners

tutorial4us.com

A Perfect Place for All **Tutorials** Resources

Core Java | Servlet | JSP | JDBC | Struts | Hibernate | Spring

Java Projects | C | C++ | DS | Interview Questions | JavaScript

College Projects | eBooks | Interview Tips | Forums | Java Discussions

For More Tutorials Stuff Visit

www.tutorial4us.com

A Perfect Place for All **Tutorials** Resources

Hibernate

(Natraj Notes)

www.tutorial4us.com

HIBERNATE

15/7/10

The logic that is given to prepare user interface and to display results by applying styles is called presentation logic.

→ The main logic of the application that generates results based on given input values is called business logic.

→ The permanent storage unit that can store data is called persistence store.

Ex:- Files, Database etc.

→ Insert, update, select and delete operations performed on the DB table - persistence store.

to maintain data are called persistence operation.

The logic used for this purpose is called as persistence logic.

→ persistence operations are also called as CRUD (or)

CRUD (or) SCUD operations

C → create (insert)

U → update (modify)

R → Read (select)

D → delete (remove)

S → select

C → create

U → update

D → delete

The technologies given to develop persistence logic is called persistence layer technology.

Ex:- JDBC, Hibernate etc.

→ JDBC, Hibernate code comes as persistence logic.

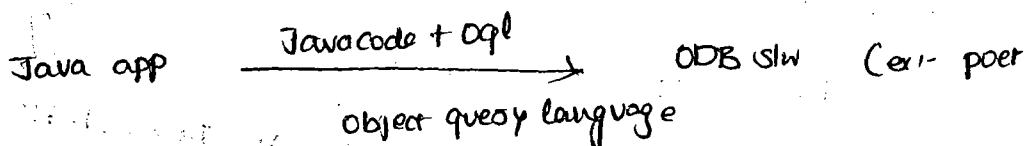
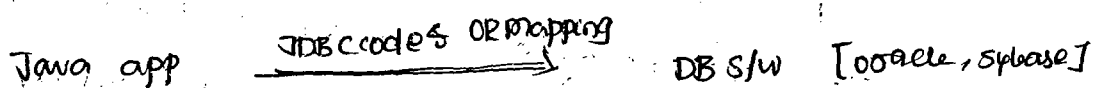
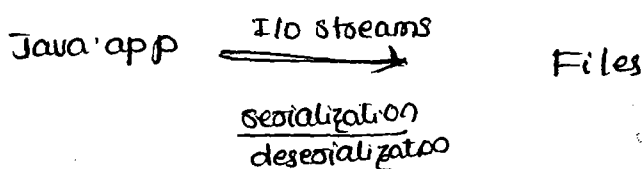
→ persistence logic is always helper logic.

Business logic.

Example applications -

Test APP

- read m_1, m_2, m_3 values from end user (presentation logic)
 - $total = m_1 + m_2 + m_3$; (Business logic)
 - store student details to DB table (persistence logic)
 - display results (presentation logic)
- (Servlet API) (JDBC) (Servlet & JSP)
- * struts - hibernate - spring
 - * RMI - EJB (predefined code & JDBC)
 - * XML - webservice (Servlets & JSP)
- Files are suitable as persistence stores in small scale applications like, desktop & mobile games etc.
 - DB softwares are suitable as persistence stores in medium & large scale application like websites, banking applications etc.
 - DB softwares that is capable of storing objects as table column values is called object DB softwares.



Example

Logic

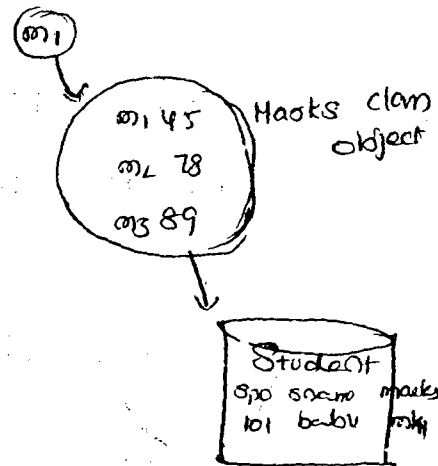
```
class marks
```

```
{  
    int m1, m2, m3;
```

```
    =
```

```
}
```

```
marks m1 = new marks(45, 78, 89);
```



Limitations with ODB slw :-

- ① storing multiple values in a single column of db table column is against of normalization Rule no: ①
- ② Generating reports from table of ODB slw is complex process.

Limitations with files as persistence stores :-

- ① No security
- ② Can't store huge amount of data
- ③ No query language support

Ex:- manipulation of files are very complex.

- Reading & comparison of data is complex.
- To solve problems with files & ODB slw use RDBMS DB slw like oracle, spbase, mysql & etc. as persistence logic stores.
- Java application can use JDBC code as persistence logic to interact with RDBMS database slws, But following limitations are these.

① SQL Queries are DB s/w dependent
since JDBC persistence logic use sql queries.

The JDBC code is DB dependent persistence logic due to this changing DB s/w in the middle of production environment (after release) and in the middle of development environment is complex process.

② JDBC Result Set object is not a serializable object so we can't send object directly over the network.

③ Exception handling, transaction management, facilities in JDBC coding are not that much sufficient for large scale application development.

✶ To solve ~~for~~ all these problems use ORMapping based persistence logic to interact with RDBMS DB s/w.

16/7/10

ORMapping :-

The process of mapping Java class with DB table, member variables with table columns & making that java class objects representing DB table rows by having synchronization b/w them is called OR-Mapping.

→ Synchronization b/w object & table row is nothing but modification done in java object will reflect in ~~the~~ table row & vice versa.

→ ORM softwares are responsible for this synchronization and to develop objects based ORMapping persistence logic.

on table rows (CRUD operations), as well as
 Java objects that are representing table rows, and
 there is no need of SQL queries. This makes ORM Mapping
 persistence logic as DB independent persistence logic.

Due to this, we can change DB software in
 the middle of project development (or) utilization
 without disturbing the persistence logic.

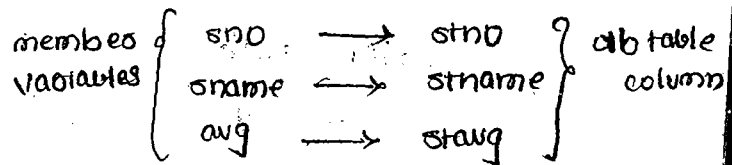
```
class Student
```

```

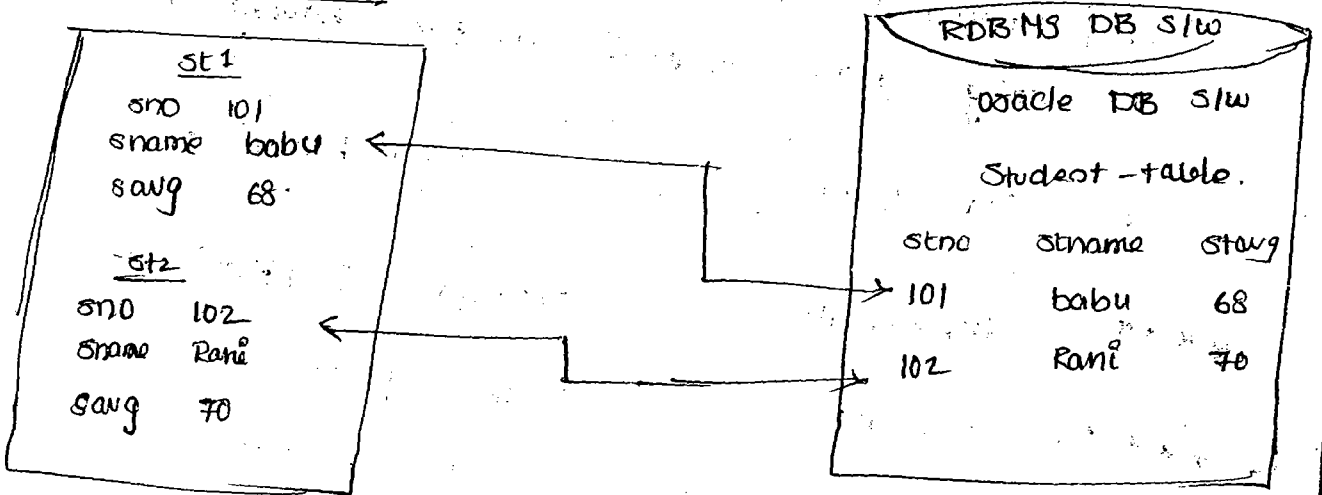
{
  int sno;
  String sname;
  float avg;
}

```

ORM Mapping work



Java Application



st1, st2 are student class objects representing

→ db table, rows.

The special S/W that provides abstraction layer on
 same category core (or) basic technology is called

framework software.

→ All ORM S/Ws internally use JDBC code to perform

persistence operation on DB table rows that are
 represented by manipulating objects.

(hiding - implementation) layer to programmers on JDBC code. so ORM sw's are also called as Framework softwares.

→ struts is a web framework sw providing Abstraction layer on Servlets, JSP Technologies.

→ spring is a Java-J2EE Framework sw providing Abstraction layer on Java-J2EE technologies like JDBC, JNDI, JMS, EJB, Servlet-JSP & etc.

→ hibernate is a O-R mapping based framework sw providing Abstraction layer on JDBC.

→ Framework sw's generate common logics of the application development dynamically by using core-technologies so, Framework sw's improve productivity (doing more work in less time having accuracy).

All O-R Mapping sw's (Java based) :-

**** 1) ✓ hibernate → redhat (Red Hat)

*** 2) ✓ Toplink → oracle corporation

3) EJB entity Beans → Sun Microsystems (oracle corporation)

*** 4) ✓ JPA → sun microsystem (given from J2EE)
(Java Persistence API)

5) OJB → Apache Foundation.
[Object Java Beans]

6) JDO → Adobe.
(Java Data Objects)

*** 7) ✓ iBatis → Apache Foundation.

1000 Objects will be created in java application
How can you justify this heavy weightness on
Java application while working with O-R Mapping
Persistence Logic?

Ans:- while working with O-R mapping persistence
logic if 1000 records are there in the table
and they are not selected at a time, then
1,000 objects will not be created in java application.

If all 1000 records selected at a time then,
1000 objects are created in java application.

So, It is recommended to programmer to
select less amount of records page by page
and to use pagination (displaying huge
amount of records page by page) to display
the records,

Here based on programmer's logic and
the no. of records that is selecting at a time.
The no. of objects created in java application
will be decided. So we must solve above
problem by doing proper programming and
pagination.

→ OR Mapping Slw's are given only to develop
Persistence Logic & they are not suitable to
develop other logics like presentation,
business logic and etc;

→ type :- ORM SW (OO) ORM tool (OO) ORM based framework.

→ vendor :- soft corp (Red Hat)

→ creator :- Ho. Gaving King & team.

→ version :- 3.5 (latest → compatible with J2SE 1.5 and 3.2/3.3 (regularly used version))

compatible with J2SE 1.5

→ To download SW :- Download SW as Zip file

from [www. hibernate .org](http://www.hibernate.org) (OO)

[www. sourceforge .net](http://www.sourceforge.net) website.

→ online tutorial :- [www. roseindia .net](http://www.roseindia.net)

→ good online articles :- [www. onjava .net](http://www.onjava.net),

[www. devzdev .net](http://www.devzdev.net),

[www. purposejava .com](http://www.purposejava.com),

[www. javabear .net](http://www.javabear.net)

→ Technical FAQ's :- [www. forum .hibernate .org](http://www.forum.hibernate.org)

→ For Interview FAQ's :- [www. geekinterview .org](http://www.geekinterview.org)

→ Reference books :- pro hibernate → apress (publisher name)

:- hibernate in Action → manning Series
(publisher name)

k.
zsdkt5
laay
stk 157

This API provides (Application programme Interface) base for the programmers to develop that technology based S/W application in java environment. API is nothing but set of classes and interfaces which come in the form of java package.

→ To develop hibernate persistence logic, programmer uses hibernate API & some helper resources like Java classes and XML files.

hibernate - def: - hibernate is an open source, light weight java based ORM software as framework software to develop ORM mapping style DB independent persistence logic in all kinds of java applications like stand alone, applets, webapplication and distributed enterprise applications.

→ the application that deals with complex & heavy weight business logic and contains additional services like security, transaction management and etc is called an enterprise application.

Ex: - banking application, credit/debit card processing application etc.

since hibernate S/W is free S/W and its source code will be supplied to programmers. it called open source S/W.

→ ETB entity bean components are heavy weight components

Reason: - They look for application server S/W.

weight slw s.

- The Resources of Entity Bean components must be developed by using EJB API support.
- Learning & applying EJB entity bean component is always complex process.

~~②~~

Hibernate is Lightweight slw s -

Reasons -

- No servers, containers are required to execute hibernate code.
- the basic J2SE slw is enough for execution.
- certain resources of hibernate application can be developed without using hibernate API.
- Learning & applying hibernate in projects development is always easy to perform.
- hibernate 3.2.5.ga.zip
- use changelog.txt file of hibernate home directory (the directory where hibernate slw is installed).
- too knowing difference b/w various versions of hibernate.
- hibernate-home \ hibernate 3.jar file represents the whole hibernate API.

→ The ORM persistence logic is DB independent even though the DB S/W is changed in the middle of project development. DB utilization in this process the DB table design must not be changed to get this effect.

⇒ MVC architecture is become industry's standard in the development of real world S/W projects.

M - Model → B.O.L + P.O.L → [Ex: Account objects]

V - View → P.O.L → [Ex: Beautician]

C - Controller → Integration / Connectivity logic.

[Ex: Traffic police]

→ logic that controls & monitors all the operations MVC architecture based applications is called "integration logic". This logic contains code to perform the communication b/w view layer resources and model layer resources.

JSP → servlet → java bean → DB S/W.

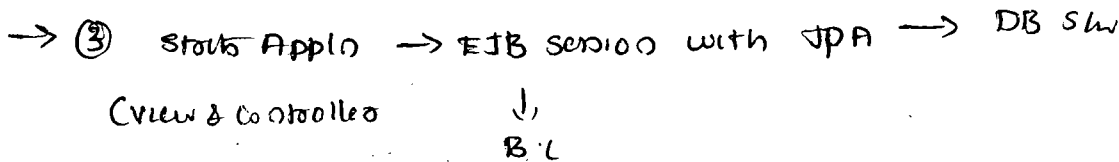
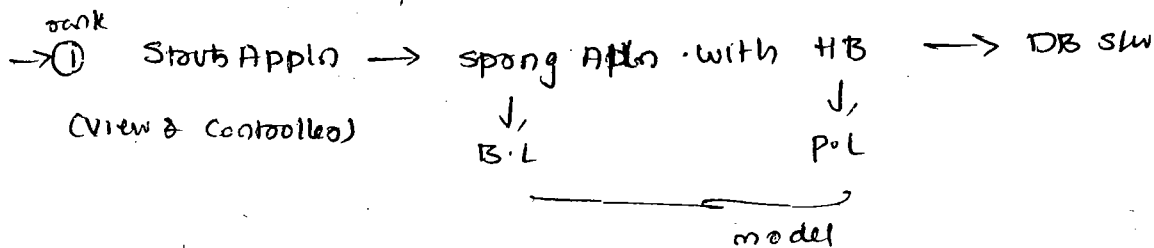
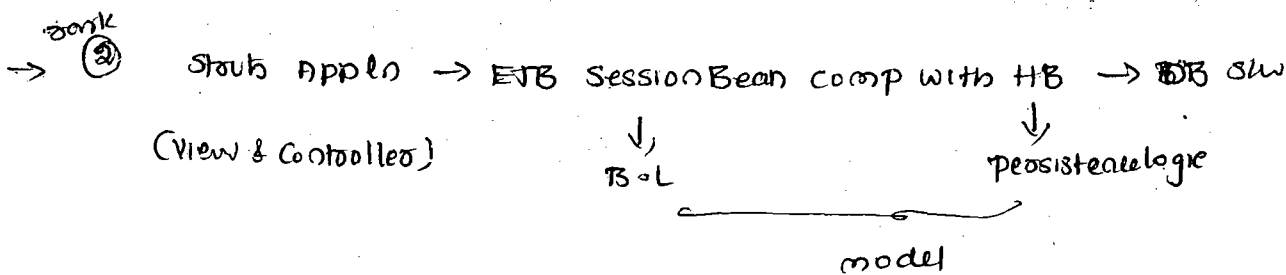
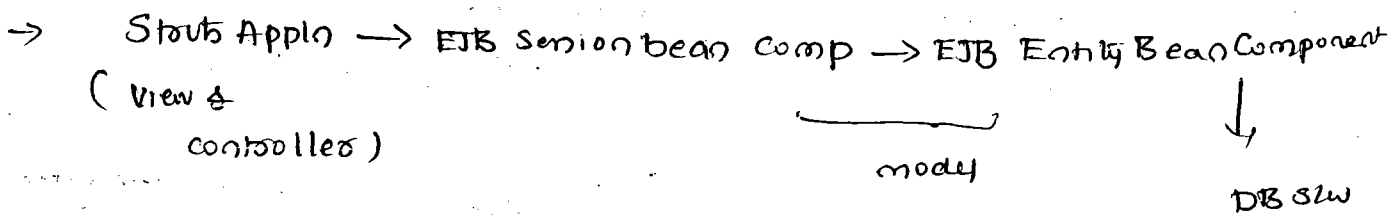
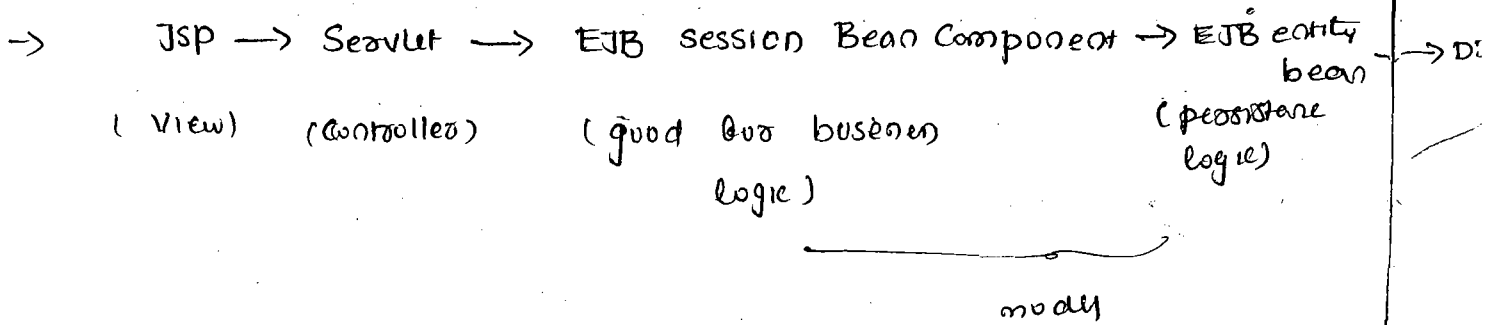
(view) controller (model)

JSP → servlet → java bean with DAO class → DB S/W.
↑
contains purely persistence logic.

model

→ The java class that contains purely P.O.L and separate that P.O.L from other logics of the application is called 'DAO' class.

that means any modification done in P.O.L does not affect other logic of the application.



view layer JSP, html, Velocity, Freemarker, XSLT etc

=> various technologies to develop "integration logic" at controller layer Servlet, Servlet with filters

=> Technologies to develop B.L. at "model" layer EJB session bean, RMI, CORBA (Common Object Request Broker Architecture) Spring and etc.

=> web framework SW to develop view, controller layer logics.

struts

JSF (Alternative for struts from Sun/Micro Systems)

Webwork

Xwork, Spring MVC and etc

=> Technologies to develop persistence logics of model layer JDBC, HB, toptik, iBatis, EJB entity bean, OJB, JPA and etc.

* => If application and its client reside on same JVM then application is called "Local application" and that client is called "Local client" to application.

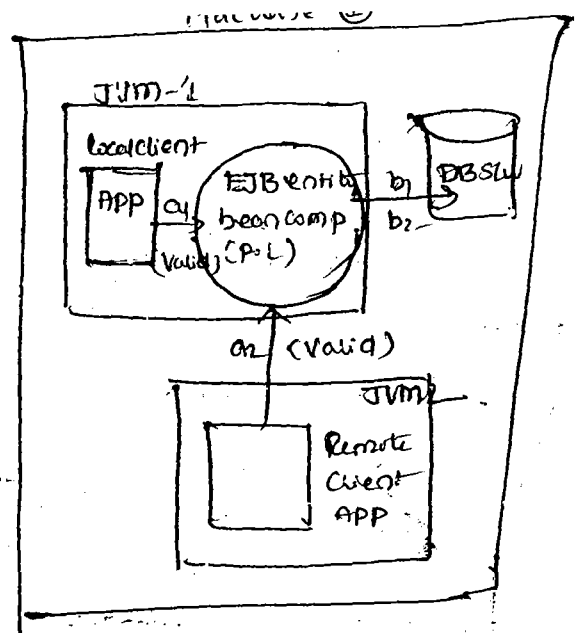
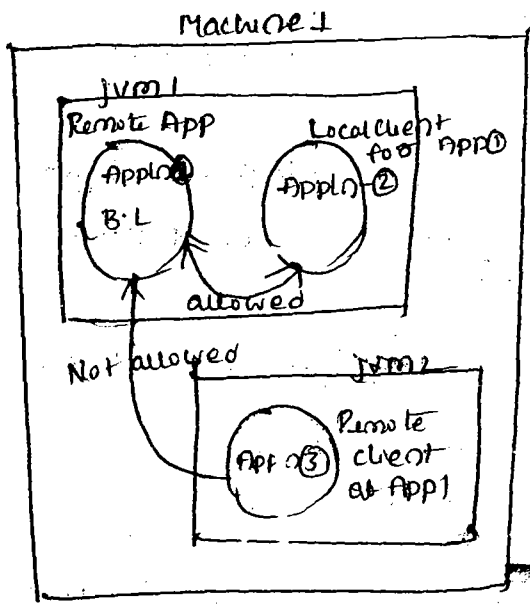
-> If application and its client reside and execute on two different JVM's at same machine (or) different machine then that application is called "Remote application" and its client is called "Remote client".

-> distributed application allows both remote and local clients. EJB components are distributed components so the persistence logic of EJB entity bean component can be accessed from local clients or remote clients.

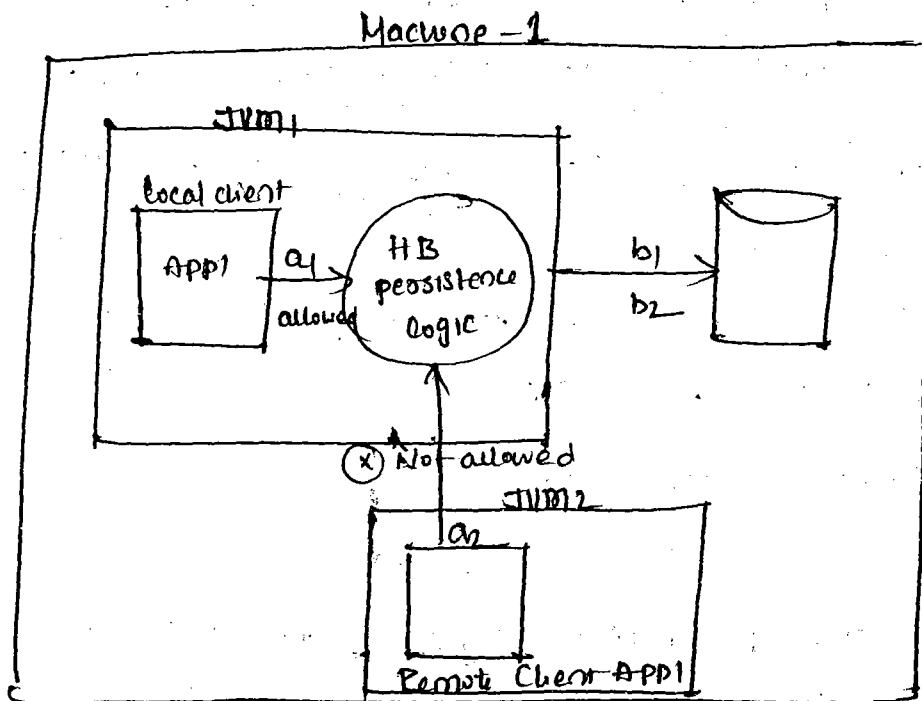
entity bean
ne
-> DB S/W

parent

SW

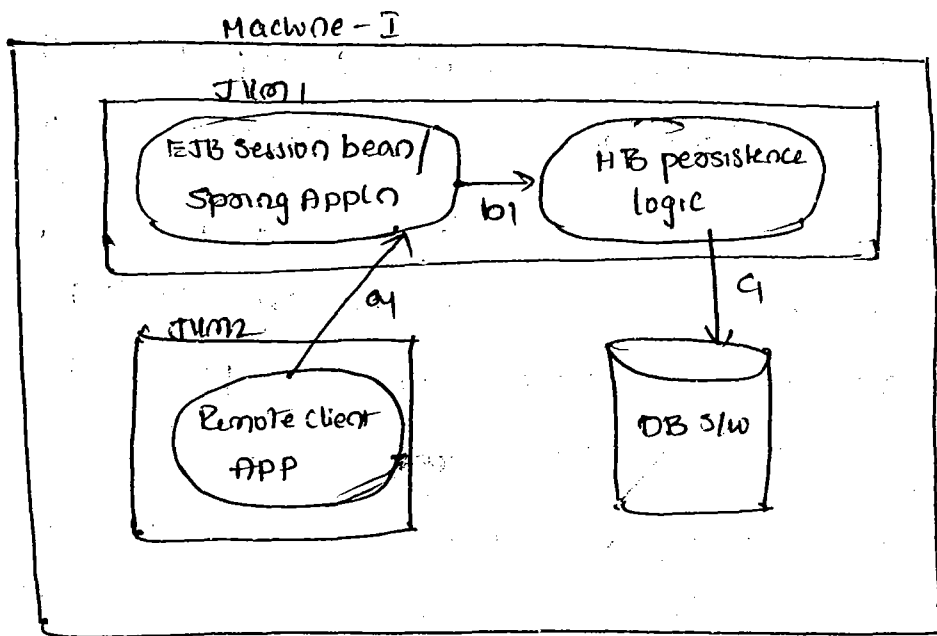


→ **HB** is not a Distributed technology so that applications can't access hibernate persistence logic from Remote places. Due to this the application and its **HB** persistence logic must be reside at execute from the same JVM. This is limitation of **HB** slw. This indicates **HB** persistence Logic must be accessed only from local client apps.



multiple java applications are executing parallelly and simultaneously.

→ we can access hibernate persistence logic from remote client applications by taking the support of EJB session bean (or) Spring distributed application as mediator application as shown below (EJB session bean component are distributed component by default)



$a_1 \rightarrow b_1 \rightarrow c_1 \rightarrow \text{DB S/W (Called)}$

→ since HB is not distributed technology, the HB persistence logic must reside along with application that wants to interact with DB S/W by using this HB P.O.L.

For example if xyz java applo wants to use HB P.O.L to interact with DB S/W then xyz applo and its HB persistence logic must be reside and execute from same JVM but the destination DB S/W can be there in remote (or) local computer to xyz applo.

Traditional client-server applications,

these are location dependent. that means any change in the location of server application, we need to inform to modify code in client application.

- Distributed applications are client-server applications having location transparency (location independency) that means change of server application location will be detected by client applications dynamically without modifying code.
- web application can be developed as traditional client-server applications (or) as distributed applications
- the application that contains complex large scale business logic and handles multiple middleware services (security, connection pooling, transaction management) an enterprise application.
- ⇒ An enterprise application can be standalone (or) a-tier (or) distributed (or) web application.

Ex:-

- 1) Banking application is distributed & enterprise application.
- 2) online shopping website is web application and enterprise & distributed application.
- 3) Hibernate supports POJO & POJI model programming that means we can take simple & regular java classes and java interfaces as resources while developing hibernate - persista. logic.
- 4) EJB 3.x, struts 2.x, spring, JSP technologies also support POJO & POJI model programming.

while developing a java class as resource at certain s/w technology based java application, if that class is not extending from a predefined class of that technology API and if that class is not implementing a predefined interface of that technology API then that class is called POJO class.

Ex:-

- 1)

```
public class Test
{
}
;
```

Here test is POJO class because it is not extending from any other predefined class.

- 2)

```
public class Test extends Demo
{
}
;
```

Demo is predefined class, so Test is POJO class.

- 3)

```
public class Test extends Frame
{
}
;
```

Test is not a POJO class it is awc API dependent.

- 4)

```
public class Test implements ABC
{
}
;
```

ABC is predefined interface so Test is POJO class.

{

≡

}

Test is a POJO class.

6) public class Test implements java.io.Serializable.

{

≡

}

Test is a POJO class because Serializable is not a
technique.

It is only basic concept of Java.

7) public class Test implements java.rmi.Remote.

{
≡
}

Test is RMI API dependent, so Test not POJO class.

8) public class Test extends HttpServlet

{

≡

}

Test is Servlet API dependent, so Test is not
POJO class.

9) public class Test

{

public void bmi()

{

≡

}

}

Test is POJO class.

while developing java interface as resource on certain technology based java application.

If that interface is not extending from predefined interfaces of that technology API then that interface is called POJI.

Ex:-

1) public interface Demo

{

≡

declaration of methods

}

Demo is POJI

2) public interface Demo extends Xyz, Mno

{

≡

}

since ~~Xyz~~ Xyz, Mno interfaces are used

defined interfaces so Demo is called POJI

3) public interface Demo extends java.rmi.Remote

{

≡

}

Demo is rmi API dependent so Demo is not POJI.

Advantages of Hibernate 2 -

- 1) supports POJI - POJO model programming.
- 2) Light weight technology to develop DB slw, independent persistence logic.
- 3) Allows to work with any Java, JEE framework slw s based applications to make them ~~interfa~~ interacting with DB slw.
- 4) use built in transaction management, connection pooling support.
- 5) Allows to work with third party JDBC connection pool slws like C3PO, ... etc.
- 6) supports two levels of ^{cos} caching to reduce network round trips b/w client applications DataBase slw.
- 7) allows to call PL/SQL procedures & functions
- 8) Gives HQL (Hibernate Query Language) as DB slw independent to perform persistence operations.
- 9) Allows to work with DB specific native SQL to perform persistence operations
- 10) allows object level Relationship in development of persistence logic when tables are there in relationship like 1-1, 1-n, n-n etc.
- 11) Given special data structures like Bag, IdBag etc to support object level relationships.

one record at one child table.

12) Easy to Learn and Easy to apply.

22/7/10

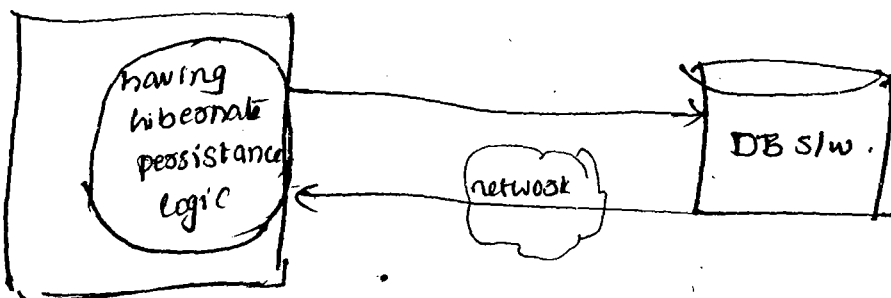
Browser → servlet → EJB component → DB SW.

The above combination browser window is client to servlet. Servlet is client to EJB component and EJB component to browser window.

This indicates there are no fixed client & server applications in SW project. Based on their roles & logics we can call them as client / server applications.

→ When java application uses JDBC code to interact with DB SW, the java application is not client to JDBC code it is client to DB SW.

Similarly, when java application uses Hibernate persistence logic to interact with DB SW then the java application is not client to Hibernate SW. It is client to DB SW.



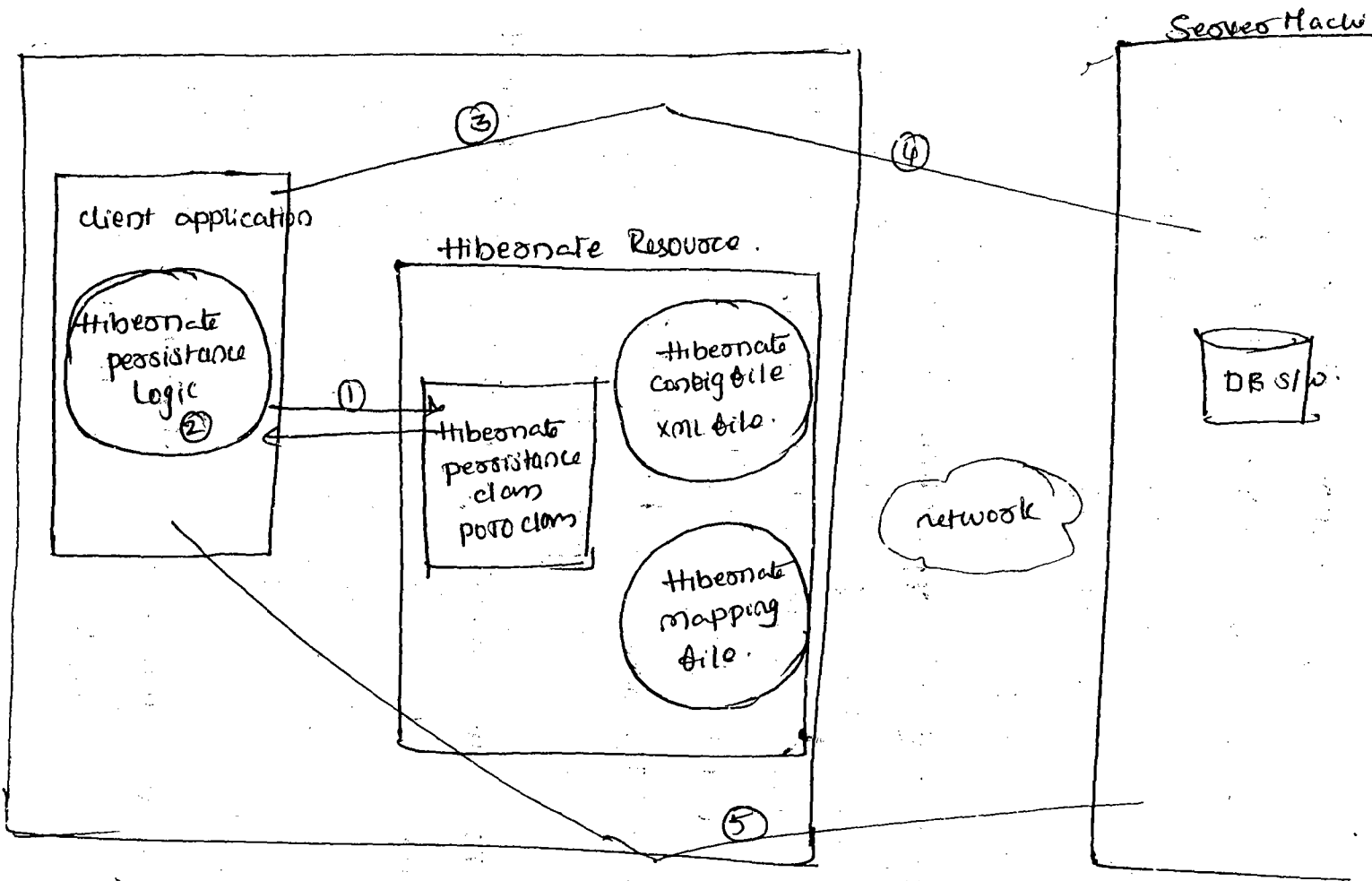
ersistence logic to interact with DB s/w.

Here the DB s/w can be local (or) remote to Java application but the hibernate persistence logic must be there along with java application.

→ different types of java JEE applications need following resources directly & indirectly to develop ORM mapping based persistence logic as hibernate persistence logic.

- 1) JDBC drivers
- 2) hibernate configuration file (xml file)
- 3) hibernate persistence class (Generally POJO class based Java bean)
- 4) hibernate Mapping file (hbm.xml file) (xml file)
- 5) hibernate API (available in hibernate-3.0.jar file)

Hibernate Architecture.



Java application
 Servlet / JSP app
 Server / JSP component
 EJB Comp / RMI app

Servlet application
 JSP
 Spring

(Faint handwritten notes on the left margin)

- ① - client application activates hibernate slw and makes that slw collecting hibernate resources.
- ② programmer uses hibernate API and hibernate resources to develop objects base OR Mapping style persistence logic.
- ③ Java (client) application interacts with DB slw by using this persistence logic (this OR Mapping persistence logic internally generates JDBC code to fulfill this requirement)
- ④ Data / DB slw will be manipulated based on the instructions given through persistence logic.
- ⑤ DB slw sends generated results back to client application.

hibernate configuration file :-

(Any file names .xml can act as hibernate configuration file but hibernate slw looks to take hibernate.cfg.xml as default configuration file name.

If any other file name is taken as hibernate configuration file it must be informed to hibernate slw explicitly.

→ This configuration file contains details to connect to database slw like driver class name Database url, Database ~~url~~ username, password and etc.
 All these details you should pass as the value of hibernate configuration properties.

will be paired based on the version
the following properties are minimum properties
of hibernate configuration file for any DB SW.

hibernate.connection.driver_class

hibernate.connection.url

hibernate.connection.username

hibernate.connection.password

hibernate.dialect

mappingfile name (hbm file name)

→ lots of properties are there in hibernate configuration
file you can collect their names from

hibernate-home/etc/hibernate.properties file

(cos) chapter 3 of pdf file of hibernate

Q) ~~What is the use of hibernate SW supplied
DB SW specific class~~

Imp Q) what is the use of hibernate.dialect property?

Ans:- this property takes hibernate SW supplied
DB SW specific classname as the value.
so, this class name will change based on the
DB SW and its version that be used in
hibernate application.

Ex:-

1) oracle any version.

org.hibernate.dialect.OracleDialect

a) oracle 9i-10g

org.hibernate.dialect.Oracle9Dialect

3) MySQL

org.hibernate.dialect.MYSQLDialect

too more dialect classname refer chapter 3
at pdf file.

→ hibernate.dialect.property value helps hibernate
slw

1) To generate and assign intelligent & sensible
default values for some hibernate configuration
properties (when they are not specified in config-
uration file) based on the DB slw.

2) makes hibernate slw optimized sql queries
smoothly based on the DB slw to fulfil persistence
operation requirement.

23/07/10

→

while developing J2EE persistence logic we can
see 100 more J2EE configuration files.

this depends upon no. of DB's that are involved
in the persistence logic execution.

Hibernate Mapping file :-

<any-filename>.xml can act as J2EE mapping
file. This file should be specified in J2EE configu-
ration file.

There is no default name for this file,
this file contains various types of ORM mapping
configuration like basic ORMapping, collection Mapping,
Association Mapping, interface mapping and etc

- DB table member variables with table column names, this file is base file for HB s/w to understand ORM Mapping configuration related to HB persistence classes.

→ you can use example applications of Hibernate-Home test folder to gather persistence example
HB-Mapping files

HB persistence class :-

A java class (or) java bean class that is taken as POJO class & mapped with DB table is called HB persistence class.

→ This class is base class for the programmer to develop objects based ORM Mapping style persistence operation in the client application.

→ It is recommended to take java bean as HB persistence class but it is not mandatory.

client Application :-

This application is client to DB s/w and interacts ; manipulates DB table data by using ~~HB~~ HB persistence logic.

→ To develop this logic programmer uses hibernate API, hibernate resources.

→ If classes of one jar file uses the classes of other jar files then other jar files are called dependent jar files of that one jar file.

→ If java application uses two party (more than JDK API's) then the 3rd party related main & dependent jar files must be added in the class path. then only our java application will be recognizing and using 3rd party API during compilation & the execution of application.

⇒ example scenario to understand importance of adding main & dependent jar files in the classpath when application uses 3rd party API.

First.jar (Main jar file)

```
public class Test
{
    public void m1()
    {
        Demo d1 = new Demo();
        d1.m2();
    }
}
```

Second.jar (Dependent jar file to First.jar file)

```
public class Demo
{
    public void m2()
    {
    }
}
```

```
public class App1
```

```
{
```

```
    public main ()
```

```
    {
```

```
        Test t = new Test ();
```

```
        t.m1 ();
```

```
    }
```

```
}
```

```
→ javac App1.java ↵
```

error :- can't invoke symbol

sol :- add first.jar file in classpath

```
→ java App1 ↵
```

error :- java.lang.NoClassDefinationError

sol :- add first & second.jar files to classpath where Demo class is used.

→ Hibernate is 3rd party slw so, it's API is called 3rd party API. when java application uses Hibernate API, to develop hibernate persistence logic the following jar files we need to add in classpath as ~~main~~ main and dependent jar files

→ Hibernate 3.jar is main jar file representing HIBAPI & it is having 7 no. of dependent jar files.

1) hibernate.o.jar

2) dom4j-1.6.1.jar

3) cglib-2.0.3.jar

4) commons-collections-2.0.1.jar

5) commons-logging-1.0.4.jar

6) jta.jar

7) asm.jar

8) antlr-2.7.6.jar

1) → Main.jar file available in Hibernate Home
Dialect.

2-8) → hibernate3.jar file available in
hibernate-home\lib folder, these file names you
can collect by seeing required class in
-readme.txt file at hibernate-home\lib
folder.

→ XML parsers are responsible to read &
manipulate data of XML files.

Ex! - SAX ~~DOM~~ (Simple API XML parsing),
DOM (Document Object Model), JDOM (Java DOM),
DOM4J and etc.

→ Hibernate ...

to read & manipulate content of xml files
called lib configuration file, HBMapping files.

*Imp
→ The web resources of web application will
be compiled from command prompt & will be
executed from webserver (or) application server.
So if these web-resources uses 3rd party API
then the 3rd party API related main jar files
should be added in classpath & the 3rd party
API related main and dependent jar files
must be added in WEB-INF/lib folder of
web-application,

→ when java web application based webresources
use Hibernate API to develop persistence logic
then the hibernate API related main jar file
(hib3.jar) must be added in the classpath
and the HBAPI related Main and dependent
jar files (17 jar files) must be added in
the WEB-INF/lib folder of web-application.

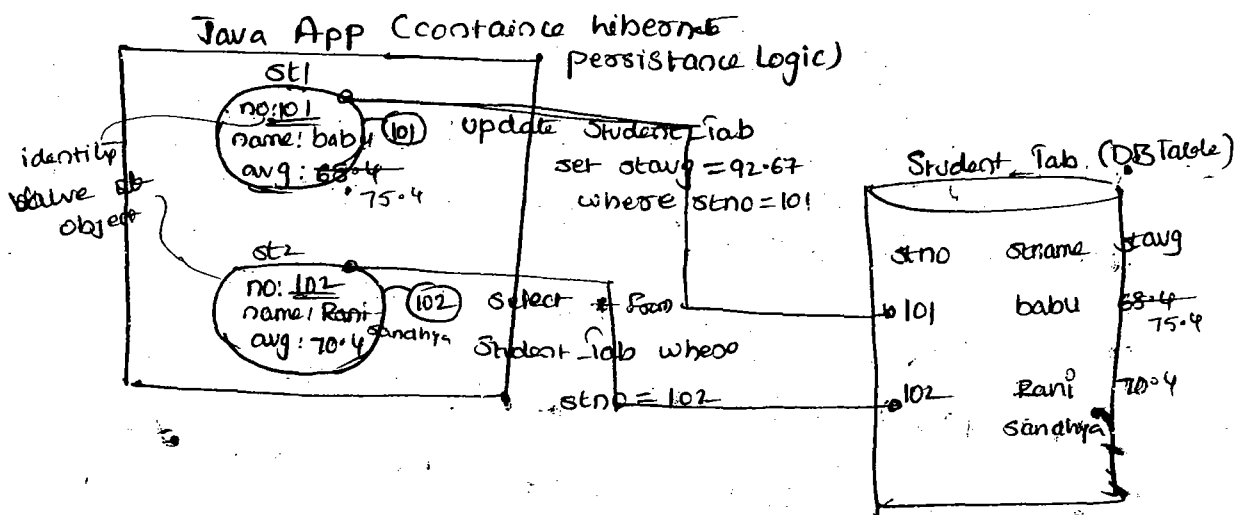
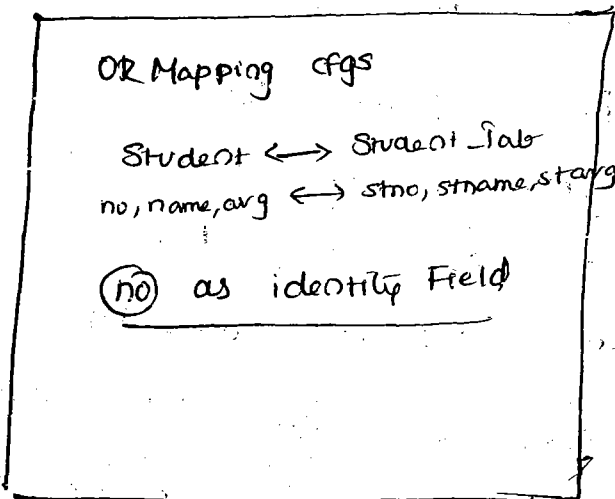
⇒ understanding synchronization btw java objects & table rows while dealing with OR Mapping Persistence Logic.

```
public class Student
```

```

{
    int no;
    String name;
    float avg;
    setX: X()
    {
        ...
    }
    getXXX()
    {
        ...
    }
}

```



→ every hibernate POJO class / persistence class object contains one identity value. hibernate slw generates this identity value based on identity field configuration done by the programmer in hibernate mapping file, programmer can configure one or more hibernate POJO class member variables as identity field.

→ In the above diagram no is contiguous as identity field, so that member variable value of st2(102) have become identity values of st1, st2 objects.

* when modification is done in POJO class object the hibernate slw internally generates update ~~select~~ query by taking identity value as criteria value to select the changes in relevant record of the DB table.
(refer above diagram)

* when modification is done in Table row (directly) then hibernate slw uses Select Query by taking object POJO class object identity value as criteria value to select that changes to the related java object.
(refer above diagram)

NOTE! - member variable of java class are technically called as fields.

NOTE! - hibernate slw identify each hibernate POJO class object by using its identity value.

→ when the programmer modifies identity field member variable value of the object (os) if the programmer modifies identity field ~~real~~ related table column value directly from sql prompt hibernate slw perform Synchronization in a regular manner, in this process if necessary the hibernate slw becomes ready to create new hibernate POJO class objects.

identity field from the member variables of
Hibernate POJO class?

ans:-

sc(1):- If POJO class related DB table is having
one unique key constraint column or one
singular primary key constraint column then take
that column related member variable in
HB POJO class and configure that one as
singular identity field of POJO class.

sc(2):- If POJO class related DB table is having
composite primary key constraint based on
multiple columns then take those multiple columns
related member variables of POJO class and
configure them as composite identity field of
POJO class.

sc(3):- If POJO class related DB table contains
no constraints programmer should ~~not~~ analyse
and given multiple columns in which
duplicate values will not be there, then programmer
has to take these multiple columns member
variables of POJO class to configure them as
composite identity field.

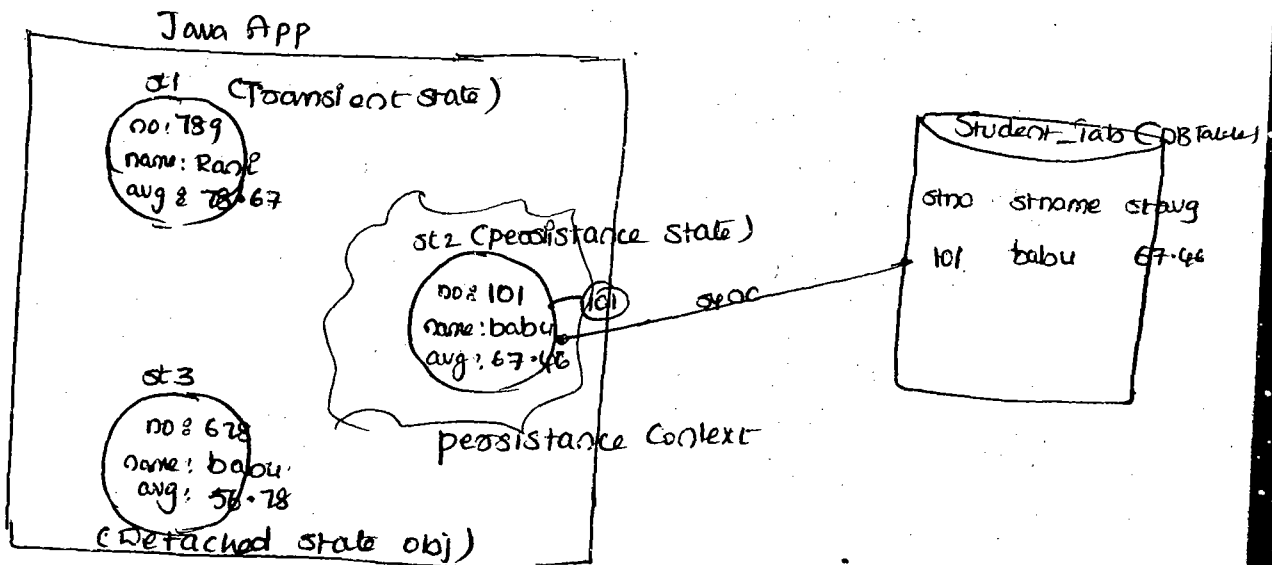
NOTE(1):- If only one member variable of HB POJO
class is configured as identity field then it is
called "singular identity field"

If multiple member variables are configured
then it is called composite identity field.

table columns are having constraints are not identity field configuration must be performed on hibernate. POJO class member variables =

⇒ we can see HJB POJO class object in ③ states

- ① transient state
- ② persistent state
- ③ detached state



Transient State :-

In this state POJO class object does not represents record in the table and object does not contain identity value, it is just ordinary java class object of POJO class.

Persistent state :-

this state object represents table row and maintain synchronization with table row, this state object contains identity value. programmer uses this state object to perform persistence operation on table rows.

maintain persistence state object is called
Persistent context.

Detached state, Transient State objects
resides outside persistence context.

Detached State:-

this state object contains identity value
but doesn't represent a record bco either
that record is deleted or the identity field
column value of that record is modified.

Detached object doesn't perform synchronization
with record.

Detached state object has represent the record
~~entity~~ earlier, but it is not currently representing
the record. that is the reason to have identity
value in Detached state object.

26/7/10

→ while developing hibernate Persistence logic in the
client application, the programmer needs to deal with

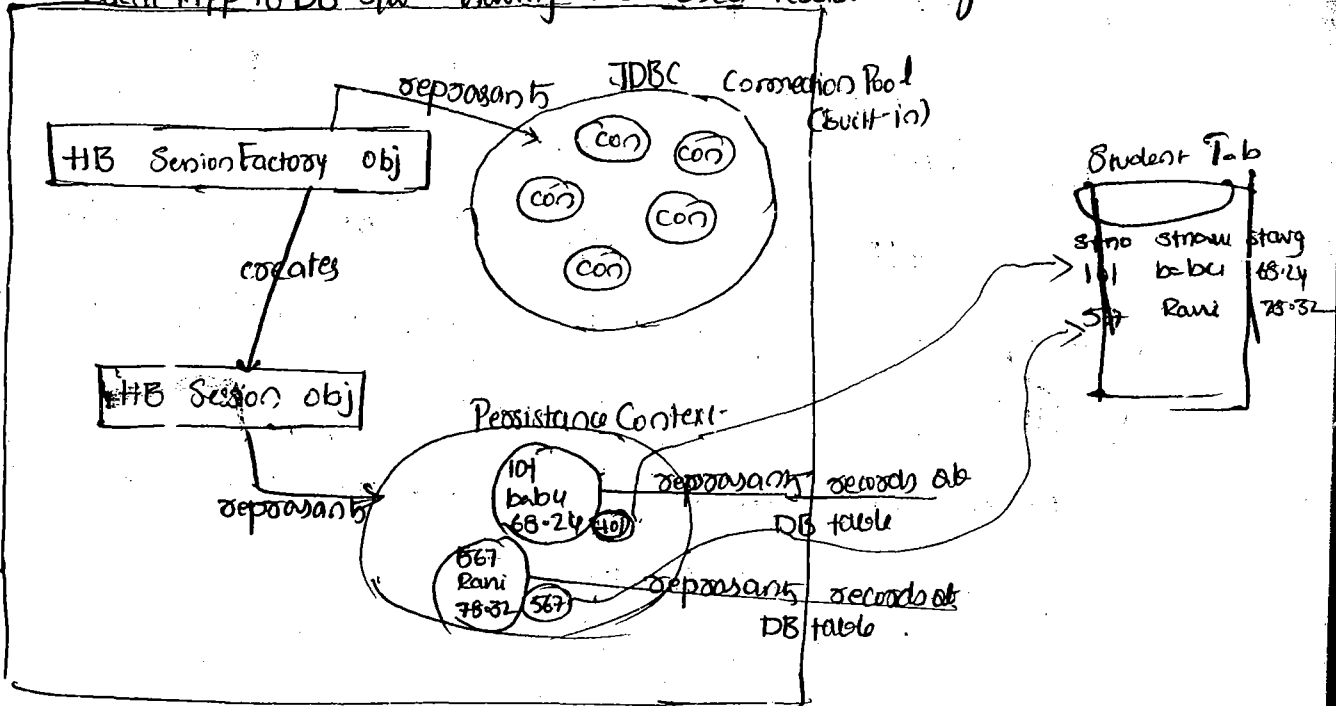
② important object-

① hibernate SessionFactory object

② hibernate Session object

Object ?

Client App to DB sw having H/B based Persistence Logic



Hibernate SessionFactory Object :-

- this object creates and represents JDBC connection pool for DB, based on the details placed in H/B configuration file.

This connection pool contains set of readily available JDBC connection objects. H/B SessionFactory

object means, it is an object of a class that implements org.hibernate.SessionFactory interface

this object is capable of representing creating one or more H/B Session objects.

Hibernate Session object :-

SessionFactory object uses one JDBC connection object of connection pool and this connection object based

Statement object to create H/B Session object.

- this Session object represents Persistence Context where Persistence state H/B pojo class objects reside representing

programmed to provide instruction to HB slw to perform persistence operations on the table rows based on the operations performed on the POJO class object.

HB Session object is no way related with HttpSession object at Server programming.
HB Session object means it is the object of a class that implements org.hibernate.Session interface.

⇒ Sample Code of Client App (that contains hibernate persistence logic) :-

```
// read cfg properties from hibernate.cfg.xml
// by activating HB slw.
→ Configuration cfg = new Configuration();
   cfg = cfg.configure();

// create HB SessionFactory object
→ SessionFactory factory = cfg.buildSessionFactory();

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

// write HB persistence logic.
// close HB Session object.
→ ses.close();
```

slw
isle
oo

cfg ① → it just ~~can~~ call HB slw to perform Persistence Logic
and ~~this~~ this object empty

cfg ② → it reads the XML file and contains details of
cfg file.

with
go
ls
L.

① Configuration class object creation activates
HB slw.

② Configure() is the factory method of
hibernate.cfg.Configuration class, which reads
configuration properties from hibernate.cfg.xml file.

③ buildSessionFactory(), uses configuration
properties of cfg object

(a) creates JDBC connection pool

(b) creates and returns HBSessionFactory object
representing that connection pool.

④ openSession(), makes HBSessionFactory object
to create one HibernateSession object

⑤ ~~Session~~ session.close(), closes connection with
DB slw represented by Session object and also
closes Persistence Context.

③ → factory.close(), closes SessionFactory object
by releasing or cleaning JDBC connection pool.

Deactivated in client apps ?

Ans) when Configuration class object is created,
+HB slw will be activated,
when factory.close() is called +HB slw
will be deactivated.

⇒

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

+HB slw looks, take hibernate.cfg.xml as Hibernate
Configuration file,

Configuration cfg = new Configuration();
cfg = cfg.configure("/myfile.xml");

+HB slw looks, take myfile.xml as +HB Configuration
file.

⇒ In the client application, we can see one
or more SessionFactory object and
one or more +HB Session object based
on the application requirement (depends on
the no. of DB slw involvement).

related

↓

Single Row Operations

call following on #B Session obj
save (pojo obj) → to insert a record
persist (pojo obj) → to insert a record

merge (pojo obj) → to update a record
update (pojo obj) → to update a record

save (or) update (pojo obj) → to insert or update a record

load (pojo obj, Identity value) → to select a record.

get (pojo obj, Identity value) → to select a record.

delete (pojo obj) → to delete record

↓

Batch operations

HQL
Native SQL
Criteria API

*** Most recommended technique to performed persistence operation, from #B environment will be HQL

7/07/2010: Develop the application to insert record into database

table by using hibernate persistence logic.

→ slw setup required

J2SDK 1.4 + slw, oraclegi + slw, Hibernate 3.x slw

→ Resource that are required.

1) DataBase table

Employee (table).

Eid number primarykey

FIRSTNAME varchar(20)

LASTNAME varchar(20)

EMAIL varchar(20)

→ create table employee (EID number primarykey,

FIRSTNAME varchar(20),

LASTNAME varchar(20),

EMAIL varchar(20));

2) Hibernate Resource

EmpBeam.java → Hibernate persistence class

hibernate.cfg.xml → Hibernate configuration file

Employee.hbm.xml → Hibernate mapping file.

3) Client Application

TestClient.java → Client Application (Java Application)

→ Procedure to develop the above application.

Step 1 - Develop the above given java, xml, Resource and

save them in a directory

F:\apps\hibernate\app1

→ TestClient.java

→ EmpBeam.java

→ Hibernate.cfg.xml

→ Employee.hbm.xml

hibernate.cfg.xml

<DOCTYPE -----

hibernate-configuration 3.0.dtd >

</hibernate>

</session-factory>

7 <property name="hibernate.connection.driver_class">~~hibernate~~
oracle.jdbc.driver.OracleDriver </property>

<property name="hibernate.connection.url">jdbc:oracle:thin@
localhost:1521:scotya </property>

<property name="hibernate.connection.username">scott </property>

<property name="hibernate.connection.password">tiger </property>

<property name="hibernate.dialect">org.hibernate.dialect.oracle9
dialect </property>

<property resource="Employee.hbm.xml"/>

</session-factory>

</hibernate-configuration>

Note 1: hibernate property file, hibernate.cfg.xml file of hibernate-
-home\etc folder as reference example files.

Note 2: The above xml is developed against the dtd rules
available in hibernate-configuration-3.0.dtd file.

// EmpBean.java (hibernate pojo class / hibernate persistence class).

```
public class EmpBean
{
    int no;
    String fname, lname, mail;

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

    public int getNO()
    {
        return no;
    }

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

```

    &
    return fname;
}

public void set setName (String lname)
    &
    this.lname = lname;
}

public String getLname ()
    &
    return lname;
}

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

public String getMail ()
    &
    return mail;
}
}

```

Note: The above class is java bean class that is taken as hibernate pojo class.

Note: hibernate pojo class must contain a zero argument constructor (public) directly or indirectly
 program or generated one ← the java compiler generated one.

Employee.hbm.xml

< DOCTYPE -----

----- hibernate-mapping 3.0.dtd ----->

```

<id name="no" column="EID"/> ⇒ singular identity field configuration.
<property name="fname" column="FIRSTNAME"/>
<property name="lname" column="LASTNAME"/>
<property name="mail" column="EMAIL"/>
</class>
</hibernate-mapping>
O-R-mapping configurations.

```

Note: use hibernate home/test folder based example applications to develop above mapping file.

The above mapping is developed against the dtd rules of hibernate-mapping-3.0 dtd file

→ <property> tag is given to map member variable with database table column

→ <id> tag can also be used for this purpose but this tag also configure that member variable as singular identity field.

TestClient.java (Client Application to database slw that is using Hibernate persistence logic)

```
import org.hibernate.cfg.*; // For configuration class
```

```
import org.hibernate.*; // For session(i), sessionFactory(i), Transaction(i).
```

```
public class TestClient {
```

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

```
// Activate HB slw and make the slw reading HB cfg
```


comfig...

```
efg = efg.configure();
```

```
// create sessionFactory object
```

```
SessionFactory factory = efg.buildSessionFactory();
```

```
// create session object
```

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

```
// write HB persistence logic
```

```
// create HB pojo class object having data.
```

```
EmpBean eb = new EmpBean();  
eb.setId(10);  
eb.setName("raja");  
eb.setLname("rao");  
eb.setEmail("rao@gmail.com");
```

'eb' object state is transient state here.

```
// now insert record.
```

```
Transaction tx = ses.beginTransaction();
```

```
ses.save(eb); // insert the record.  
tx.commit();
```

'eb' object state is persistent state here.

```
// close session object
```

```
ses.close();  
// close sessionFactory obj  
factory.close();
```

'eb' object state is detached state here.

```
// close of main
```

```
// close of class
```

Now:

All non select operations, ...
HB persistent logic must be executed as transactional
stmts as shown above.

non select operations means insert, update, delete, -
operations.

Steps:

Add following jar files to the class path

1) hibernate3.jar // available in HBhome directory representing -
- HB API.

2) dom4j-1.6.1.jar

3) cglib-2.1.3.jar

4) commons collection-2.1.1.jar

5) commons-logging-2.1.1.jar

6) jta.jar

7) asm.jar

8) antlr-2.7.6.jar

9) ojdbc14.jar file.

2-8 ⇒ dependent jar file to hibernate3.jar available in
HBhome/lib folder.

9 ⇒ Represent Oracle thin driver, available in oracle/ora^{9i, 10g}
-jdbc14lib/folder

Note: Always use my computer Environment variable to add jar-
files to class path.

";" symbol should be taken as separator b/w multiple
values added to my computer Environment variable, should not
be taken as begginner or terminator.

Note:

After adding new jar files to my computer Environment
variable class path plz open new command prompt in order to

step 3: compile java Resources of the application

E:\App\hibernate\APP1 > javac *.java ↵

step 4: Execute the client application

E:\App\hibernate\APP1 > java TestClient ↵

⇒ To make HB s/w showing internally generated sql queries to fulfil our requirement use "show-sql" property of HB configuration file as shown below

```
<property name="show-sql" true</property>
```

⇒ when session.save() method is called by passing transient

- state HB pojo class object

a) That pojo class object will be made as persistent state object of persistence context by generating identity value to that object.

b) when application commit its transaction save() method related record insertion take place in database table. Then on words that object and inserted record will be in synchronisation.

Q) What is the difference btw inserting record by calling session.save() and by calling session.persist() method?

Ans: Both methods are there to insert the record. Both methods are capable of generating identity value for HB pojo class object.

generated identity value as the return value of + method.
This identity value comes as Serializable Object used on this
value the programmer can evaluate whether a d is inserted
- or not.

to
ration
with session.persist() we can't catch case the
generated identity value, because the return type of this method
is void.

→ Example code of session.save() method.

cut-
Transaction tx = ses.beginTransaction();
Integer idval = (Integer) ses.save(Obj);
s.o.p("Identity value" + idval);
tx.commit();

ity
→ Example code of session.persist()

tabb
Transaction tx = ses.beginTransaction();
session.persist(Obj); // insert the record.
tx.commit();

Note: within the transaction, if HB pojo class object is modified
with multiple no. of times instead of generating separate sql query
each time the HB slw keep track of all the change done
to the object from begin of transaction to end of transaction.
Then generate final sql update query reflecting all the changes
when transaction is committed. This saves slw and reduces n/w
round trips b/w application and database slw

0
specification is a document containing rules and
guidelines in the form of APIs to develop n. slw's

new s/w's, Based on proprietary specification only certain company can develop the s/w's.

JDBC, Servlets, EJB, JPA, etc... are open specification.
Oracle is the proprietary specification

Based on JPA specification supplied by SUN MICRO SYSTEM all ORMs s/w's like Hibernate, TopLink, JBoss and etc... are developed

→ Hibernate 3.x version s/w's are given based on JPA specification.

29/07/2010

* Code to "modify" the record by using "session.merge()"
method :-

```
EmpBean eb = new EmpBean();
```

```
eb.setNo(1080); // persistence identity value
```

```
eb.setFname("new raja");
```

```
eb.setLname("new oao");
```

```
eb.setMail("newoao@gmail.com");
```

```
Transaction tx = ses.beginTransaction();
```

```
persistence  
state object eb = (EmpBean) ses.merge(eb); // update the record  
Transaction state object
```

```
s.o.p (eb.getNo() + " " + eb.getFname() + " " +  
eb.getLname() + " " + eb.getMail());
```

Transaction tx = ses.beginTransaction();

ses.update(eb);

tx.commit();

→ "session.merge()" method updates the record and returns persistence state object representing the update record.

where as "session.update()" method update the record but can't return persistence state object of that record (the return type of session.update() method is void).

* ⇒ while performing single row operations from hibernate persistence logic by using single row operation methods of hibernate session object, they can take only "identity value" of given POJO class object as criteria value. In order to perform these operations based on other criteria values we need to work with HQL queries and other techniques.

* code to "delete" the records -

EmpBean eb = new EmpBean();

eb.setNo(1080);

Existing identity value.

// Now delete record

Transaction tx = ses.beginTransaction();

ses.delete(eb);

Transaction state object

tx.commit();

→ here "eb" object state will be "detached" state.

persistance logic can be executed as non-transactional operations -

* Selecting a record by using session.load() :-

```
EmpBean eb = (EmpBean) ses.load(EmpBean.class,  
                                new Integer(1010));
```

↑ persistence state pojo object
representing data of the selected record.

pojo class name
↑ identity value as a criteria value.

```
s.o.p (eb.getNoC() + " " + eb.getFname() + " " + eb.getLname() + " " + eb.getMail());
```

Note :- Comment s.o.p statement to observe the

'lazy loading' with session.load() method
(it can't apply select queries)

* Select a record by using session.get() :-

```
EmpBean eb = (EmpBean) ses.get(EmpBean.class,  
                                new Integer(1010));
```

```
s.o.p (eb.getNoC() + " " + eb.getFname() + " " + eb.getLname + " " + eb.getMail());
```

⇒ Object of java.lang.Class can represent a java class or an interface or an abstract class in a running java application.

"EmpBean.class" statement kept in application generates object of java.lang.class representing

Given above all called they use given identity value
a criteria value, they select record from the table,
They store selected record into given pojo class object
(for this, it dynamically creates object ~~to~~ from given
pojo class) and return that object.

⇒ "session.load()" methods performs "lazy loading"
to select the record that means until "getXXX()"
are called on the return object (obj) record
"will not" be selected from the table.

→ session.get() method doesnot performs "lazy loading"
in any situation.

⇒ Q) How can we avoid "lazy loading" even though
you are working with "session.load()" method?

ans:- ⇒ public Object load (Class name, Serializable id)
This performs "lazy loading"

ex:- EmpBean eb = (EmpBean) ses.load (EmpBean.class,
new Integer (1010));

⇒ public void load (Object obj, Serializable id)
It doesnot performs "lazy loading"

ex:- EmpBean eb = new EmpBean ();
ses.load (eb, new Integer (1010));

⇒ public Object get (Class name, Serializable id)
it doesnot performs "lazy loading"

class object identity field member variable value as
criteria value to check whether that value based
record is already available in the table or not.
If available This method performs "update" operation
otherwise This method performs "insert" operation.

Ex1- EmpBean eb = new EmpBean();

eb.setNo(1009);

eb.setName("raja");

eb.setLname("rao");

eb.setEmail("rao@gmail.com");

Transaction tx = ses.beginTransaction();

ses.saveOrUpdate(eb);

tx.commit();

⇒ In server side components like servlets, JSP's,

Servlet components and etc it is recommended to

use "session.get()" method to select a record.

In other situations it is upto the choice

of programmer to use "session.load()" (or)

"session.get()" method to select a record

MySQL

type: DB SW

version: 4.0

Vendor: MySQL

Open source DB SW

Default port no: 3306

default logical db names: test, mysql

type: 4 mechanism based connection :- connecto;

default admin username and password:

root (username)

root (password)

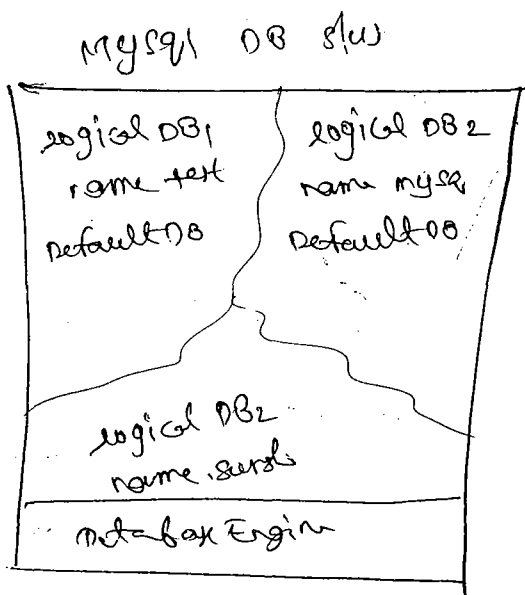
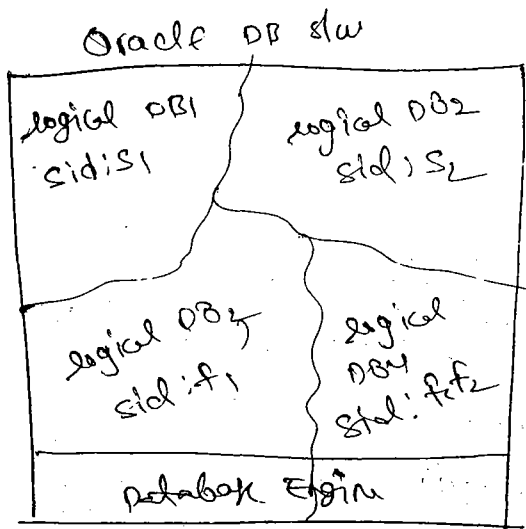
→ After installing mysql 4.x SW we need to activate that SW separately by using winmysqladmin tool available in mysql/home/bin directory (look for traffic signal symbol in the system tray)

→ my sql front is a third party supplied GUI tool to perform operations by connecting to mysql database SW it is like sql*plus tool of oracle DB SW

→ logical databases are the logical partitions created in physical database SW installed in computer. Each logical db will be identified with one logical name. In oracle DB SW this name is also called as sid (service id). In MS Access DB SW each MDB acts as one logical database and that MDB file name is logical database name.

In a company if multiple projects are using same DB SW then the DB SW will be installed

database will be created in that database s/w on one per project base.



⇒ Procedure to create logical database and DB table in that by using mysql front tool.

connector/jdbc driver class

step 1 launch mysql front tool

[New] → [my con name] → OK → host; name: localhost,
 user: root, password: ~~root~~ ^{no password} → connect, port: 3306
 connect → Right click on root of localhost →
 create DB → Database name: sunsh HBDB → OK → select
 sunsh HBDB → Database tab → RC → create table → Table name:
 Employee → Add → ID - INT: 5 primary key, unique, not null
 first name varchar 20 last name varchar 20 Email
 varchar 20 → create → select Employee table RC →
 insert record 101 RAJE RAJ RAJ@gmail.com
 102 RAMESH CHARY CHARY@gmail.com

connector/jdbc driver:

type: type 4 mechanism based jdbc driver

website: WWW.devA.com.

driver class: org.gjt.mm.^{my}sql.Driver

Search Engine

url: jdbc:mysql://<logical db name>

jar file: mysql-connector-java-3.0.8-stable-bin.jar

→ The above jar file is available in mysql-connector-java-3.0.8-stable-bin.jar
→ Zip Extraction

Note: MySQL slow setup, MySQL front slow setup connector/j driver connect sty folder of DVD.

→ Procedure to make our first HB application to interact with MySQL database slow.

step 1: Keep all the resource as it is, but do following modification in HB configuration file.

hibernate.cfg.xml

<DOCTYPE----->

<hibernate-configuration>

<session-factory>

<property name =

>org.gjt.mm.

Driver </property >

< " "

>jdbc:mysql://<sursh

HOBO </property >

<property name = "connection, username" >root </property >

< " " = " password" > </property >

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

</session-factory>

</hibernate-configuration>

steps) Add mssqlconnector.jar (3.0.8/et-ble/-bin/) jar file to the class path along with already existing jar files

Note: it is always recommended to add new jar file to class path at the beginning of existing jar file

steps) Run the client application.

We can't interact with MS-Access DB s/w by using hibernate persistence logic becoz, the HB s/w not providing dialect class for MS-Access.

The word ~~word~~ HO is optional in HO configuration property name

31-07-2010

POJO object to DB table :-

session.flush() method, forces the HB s/w to synchronize with underlying DB s/w records by using the data as persistent state POJO class object as persistence context

Write following code in client application, to see the modification done in object in associated record of the DB Table.

11/11/10

```

// select record from DB table into POJO object
EmpBean eb = (EmpBean) ses.get(EmpBean.class, new
Integer

```

```

s.op ("eb.getNo() + " " + eb.getName() + " " +
eb.getLname() + " " + eb.getEmail());

```

```

// modify "eb" (persistent state obj) object data

```

```

eb.setEmail("xyz1@xyz1.com");

```

```

ses.flush("xyz1@

```

```

ses.flush(); // this method selects changes done

```

```

// in "eb"

```

```

// to DB table row through synchronization.

```

```

// close session obj.

```

```

ses.close();

```

```

} } factory.close(); // close session factory

```

The above code demonstrates POJO object to DB table row synchronization.

DB table to POJO Object ? — (Synchronization)

code in the client application to demonstrate
 @ DB table row to java object based to DB Synchronization
 [Modification done in DB table row from DB side will
 reflect in the associated persistent state object]

```

// select record from db table into POJO obj.

```

```

EmpBean eb = (EmpBean) ses.get(EmpBean.class,
new Integer(1016));

```

```

s.op ("eb.getNo() + " " + eb.getName() + " " + eb.getLname() +
" " + eb.getEmail());

```

// record "1016" in db table from sql prompt.

```
ses = refresh(eb);
```

// read or reads data from db table row into eb object for synchronization.

```
SOP ( eb.getId() + " " + eb.getFname() + " " +
```

```
eb.getLname() + " " + eb.getMail());
```

Note:- Can you develop HJB application (or) HJB Persistence logic without HJB Configuration file?

ans:- yes, it is possible and there are two approaches for this,

Approach (1):- (By using setProperty() at org.hibernate.cfg.Configuration class.

code in client application:-

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

```
cfg.setProperty("hibernate.connection.driverClass",  
"oracle.jdbc.OracleDriver");
```

```
cfg.setProperty("hibernate.connection.url",  
"jdbc:oracle:thin@192.168.1.152:1521");
```

username
password cfg.setProperty

```
cfg.setProperty("hibernate.dialect", "org.hibernate.dialect.  
Oracle9Dialect");
```

```
cfg.addFile("Employee.hbm.xml");
```

// create sessionFactory obj

```
sf = cfg.buildSessionFactory();
```

→ since configuration property values are hardcoded in the application the application will lose its flexibility

→ we can't work with certain miscellaneous DB configuration properties like 'show_sql'
cfg.setProperty("show_sql", "true"); // it can't show query

to solve this problem no ① use Approach ②

Approach ②

→ the text file that maintains the entries in the form of <key, value> pairs.

→ java.util.properties is a Map Data Structure each element of this Data Structure should have string values as <key and value> pairs.

this class is a subclass of Hashtable class

this Hashtable Map Data Structure elements can take any objects as keys and any objects as values

the element values of java.util.properties Data Structure can be collected from the Text Properties file.

Note:-

Example code based on Approach No ②

props.txt:-

this is properties file

hibernate.connection.driver_class = oracle.jdbc.driver.OracleDriver

hibernate.connection.url = jdbc:oracle:thin:@152.1.3.1:1521:scott

hibernate.connection.username = scott

show - sql = true .

=

Code in client application to create HJB Session object - 2-

// read value from props.txt to java.util.Properties obj

```
FileInputStream fis = new FileInputStream("props.txt");
```

```
Properties p = new Properties();
```

```
p.load(fis); // load data from props.txt to  
"p" object.
```

```
s.o.PC p.tostring());
```

// activate HJB slw and make the slw reading HJB cfg file.

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

```
- cfg.setProperties(p);
```

```
cfg.addFile("Employee.hbm.xml");
```

== // create Session Factory object -

Note: - these approaches ① & ② practices are not industry standard practices always work with xml file as HJB Configuration file in real world applications.

=

object in the form of single line statements.

```
Session ses = new Configuration().configure("/mycfg.xml").  
    buildSessionFactory().openSession();
```

the above statement demonstrate method chaining process.

2/08/10

My Eclipse: -

type: IDE sw to develop java, j2ee and other java
flw sw's based application.

Version: 6-x. (compatible with j2sdk 1.5)

Vendor: - Eclipse

Commercial sw: -

(cheat-codes)

does not give built in servers, but allows

to work with configure all external servers.

to download sw: - www.myeclipseide.com

for documentation: - www.myeclipseide.com.

→ a plugin is a patch sw (or) sw application
to enhance the functionalities of existing sw (or)
sw application.

plug-in's come as jar files.

MyEclipse IDE = Eclipse IDE + builtin plugins.

Eclipse IDE

MyEclipse IDE

- 1) Design to develop basic J2sdk application.
- 2) extend plug-ins are required to develop advanced technologies based applications.
- 3) opensource slw

- 1) Design to develop all kinds of java, J2EE apps and other Framework slw based applications
- 2) built in plug-in's are there to ~~work~~ develop advanced technologies based java applications.
- 3) commercial slw.

Q) How to add plug-in's to the projects of Eclipse IDE?

Ans:-

① download jar files that represent certain technologies related plug-in's like hibernate plugin, spring plugin etc.

Note:- these plug-in provide the necessary environment required for the programmer to develop certain technology based application from Eclipse IDE.

- ② create project in Eclipse IDE and observe that there will be plug-in's folder in the project.
- ③ Add the downloaded plug-in related jar files to plug-in folder of project.

activates plug-in and programmer can use the plug-ins applied environment to develop that technology based application.

Procedure to develop 1st HB application by using

MyEclipse 6.0x IDE :-

Step 1 :- Launch MyEclipse IDE and choose work space.

Note :- work space is a folder where all projects created in MyEclipse IDE will be saved.

** Step 2 :- update the subscription

MyEclipse Menu

↳ update subscription

↳ subscribes :- administrator

subscriptioncode :- "XXXX" (check code)

Step 3 :- use MyEclipse IDE, to create DB profiles :-

⇒ connected with ORACLE DB s/w . (DBNode)

Windows

↳ open perspective

↳ other

↳ MyEclipse DB Explorer

OK

goto 'DB Browser window

Right click

New

Oracle thin driver

Driver name :- ocap (logical name of profile)

Connection url :- jdbc:oracle:thin:@1521:1521:1521

username :- scott

password :- tiger

Unives word:-

Address:-

save password.

Next

Next

finish

⇒ Right click on ORAP in DB Browser window.

open connection.

⇒ step ①:- Make sure that Employee table is available in Oracle DB slw having at least one primary key constraint column.

Create java project in MyEclipse IDE

• File
↳ New
↳ project
↳ java project
↳ Next

project name:- MyHIS project

Next

↳ finish

⇒ Add hiberate capabilities to the project

Right click on project :-

↳ MyEclipse

↳ Add hiberate capabilities.

select hibernate 3-1 core libraries

~~jav~~ " " Advanced support librar



Next



Next

DataSource :- use JDBC Drivers

DB Drivers - oasp (DB pool created above)



Next

create SessionFactory class?

⇒ src

⇒ java package ? New → p1

class name : MyHelper



finish



goto hibernate.cfg.xml file

↳ add miscellaneous properties

→ show-sql

→

Note:- the above step gives following things.

- adds hibernate API related jar files to -
build path (or) classpath of the project
- hibernate.cfg.xml as hibernate configuration file
- use myhelper.java file having code to create
hibernate ~~or~~ SF object and Session object.

~~Step 1~~
=> Postoom . Hibernate Reverse Engineering on
Employee table to generate HBM POJO class
(Employee), HBM Mapping file (Employee.hbm.xml)
dynamically :-

Window
↳ open perspective
↳ other
↳ MyEclipse DB Explorer
↳ DB Browser Window

Expand ORAP ←

Expand Connected to ORAP ←

Expand Scott ←

↳ Expand table

↳ Right click on Employee
↓

HBM Reverse Engineering

Java src folder :- /MyEclipse/src

HBM Mapping file

~~hbm~~ update HBM configuration with mapping file location

Java Date Object

Next

Type Mapping :-

select HBM types

↳ Next

↳ Finish

↳ OK

① Employee.java as 3 POJO class

② Employee.hbm.xml as hibernate mapping file

⇒ Develop client app in the project :-

Right click project

→ New
↳ class

Name : Test

Main arg[])

Finish

ctrl + shift + O

// import statements

class TestClient

```
Transaction tx = session.beginTransaction();
```

```
Employee e1 = new Employee();
```

```
e1.setEid(016);
```

```
e1.setEname("babu");
```

```
e1.setLname("kashin");
```

```
e1.setEmail("a@b.com");
```

```
// e1.setEmail(args[0]);
```



```

        tx = commit();
    MyHelper.closeSession();
}
catch (HibernateException he) {
}
catch (Exception e) {
}
}
}

```

⇒ Execute the client application.

Right click on source code of TestCode obj java
 ↳ Run as
 ↳ java application.

⇒ What is the diff b/w session.update & session.merge?

both methods take POJO class object as argument
 value

session.update() :- update the record, only
 when given object related record is available
 in the table, otherwise update method fails
 in record updation. [if record is not available]

session.merge() :- merge method tries to
 update the record if given POJO object record
 is available in DB table, or, (if not available)
 this method inserts the record by using given
 POJO class object data.

application being from my Eclipse IDE :-

Right click on source code of application

↓
Run As

↓
open run dialog

↓
chose app name (TestCueor)

↓
Arguments tab

program arguments:

Val1 <space> Val2 <space> Val3
args[0] args[1] args[2]

↓
Apply
↳ Run

MYSQL
3/8/10

procedure to create DB profile in MyEclipse IDE pointing

to MYSQL DB slw :-

Window menu

↳ open perspective

↳ Other

↳ MyEclipse DB Explorer

↓
DB Browser window

Right click

new

Driver template : MYSQL Connector

Drivername : mysql

Connectionurl : jdbc:mysql://localhost:3306/

username : root

password : - (NO pwd)

jar file :-

mysql

mysql-connector-java-5.0.8-stable-bin.jar

↓
 next
 ↳ finish
 ↓ =

Right click on mysqlp
 ↳ DB Browser window
 ↳ open connection
 ↳ ok

 → Example Application to communicate with multiple DB slvs
by using hibernate persistence logic & -

*** while working with Oracle 10g DB slvs [XE] to commu-
 nicate from HIB Persistence logic also add ehcache-1.2.3
 jar file in class path along with other regular
 jar files.

→ Dialect class name for this environment
 org.hibernate.dialect.Oracle10gDialect
 (or)
 org.hibernate.dialect.Oracle9Dialect.

Resources Required :-

(1) in Oracle DB slvs -

Employee (table)

Field	PK	Number
FIRSTNAME		VARCHAR2 (20)
LAST NAME		(20)
EMAIL		(20)

make sure that some records are available in
 this table.

Employee (table)

EID PK ~~varchar~~ int(5)

~~varchar~~

FIRSTNAME

varchar(20)

LASTNAME

''

EMAIL

''

(3) files :-

mycfg-ora.xml (oracle DB s/w related cfg file)

mycfg-mysql.xml (mysql DB s/w related cfg file)

Employee-hbm.xml (HBM file)

EmpBean.java (HIS persistence class)

TestClient.java (Client app having persistence logic)

Aim:- AIM OF THE ABOVE APPLICATION IS SELECT A RECORD FROM ~~DB~~ DB TABLE (Employee) OF ORACLE S/W AND INSERT THAT RECORD INTO MYSQL DB TABLE (EMPLOYEE)

mycfg-ora.xml :-

write minimum 6 properties ab point into oracle DB s/w and take Employee-hbm.xml as mapping file.

mycfg-mysql.xml :-

write minimum 6 properties ab point into mysql DB s/w and take Employee-hbm.xml as mapping file.

(4) Employee-hbm.xml :-

← same as first application →

100
5 s/w s
tion
Common
-1-2-3
gular

← same as 1st application →

TestClient.java :-

```
import org.hibernate.cfg.*;
```

```
import org.hibernate.*;
```

```
public class TestClient
```

```
{
```

```
    main ( ) throws Exception.
```

```
{
```

```
    // get HB session object connected to oracle DB SW.
```

```
    Configuration oacfg = new Configuration();
```

```
    oacfg = oacfg.configure( "hib.cfg-ora.xml" );
```

```
    SessionFactory oafactory = oacfg.buildSessionFactory();
```

```
    Session oasess = oafactory.openSession();
```

```
    // get HB session object connected to mysql DB SW
```

```
    Configuration mysqlcfg = new Configuration();
```

```
    mysqlcfg = mysqlcfg.configure( "hib.cfg-mysql.xml" );
```

```
    SessionFactory mysqlfactory = mysqlcfg.buildSessionFactory();
```

```
    Session mysqlsess = mysqlfactory.openSession();
```

```
    // write HB persistence logic interacting with multiple DB SW
```

```
    // select a record from oracle DB SW
```

```
    EmpBean eb = (EmpBean) oasess.get( EmpBean.class, new Integer(1016) );
```

```
    // insert the record into mysql DB SW
```

```
    Transaction tx = mysqlsess.beginTransaction();
```

```
    mysqlsess.save( eb );
```

```
    tx.commit();
```

```
    oasess.close();
```

```
    mysqlsess.close();
```

```
    oafactory.close();
```

```
    mysqlfactory.close();
```

```
} // closing session object
```

```
}
```

```
} // closing Session object
```

```
}
```

Add Regular hibernate jar

add ~~edge~~ ojdbc14.jar, m

-connection-java-3.0.8-stable

-bin.jar

⇒ compile all java resource and execute the client application.

SessionFactory object is not a singleton object
but it is immutable (not a singleton object)

NOTE! - saying the implementation class of
e.g. hibernate SessionFactory interface is singleton java
class is a wrong statement - because we can create
multiple SF objects in a single java application
as shown above.

NOTE@! - the java class that allows to create only
one object for jvm is called singleton java class

→ SessionFactory object is immutable object, that means
after creating SF object if you modify its configuration
properties dynamically at runtime they will not be
reflected to the SessionFactory object.

→ String class object is immutable object, StringBuffer
class object is mutable object.

→ immutable object means when modification is done
in the object it will not reflect in the same object,
modification will be done by creating new object

→ mutable object means, modifications done in the object
the object will reflect in the same object.

Test = java (mutable class)

```
class Test
{
    int a;
    String b;

    Test (int a, String b)
    {
        this.a = a;
        this.b = b;
    }

    public void setDate1 (int a)
    {
        this.a = a;
    }

    public void setDate2 (String b)
    {
        this.b = b;
    }

    public String toString ()
    {
        return "a = " + a + " b = " + b;
    }

    main (String [] args)
    {
        Test t = new Test (10, "babu");
        S.o.p ("t obj date " + t.toString());
        t.setDate1 (20);
        t.setDate2 ("hello");
        S.o.p ("t obj date " + t.toString());
    }
}
```

Comment:- when setDate1(),
setDate2() methods are
called on "t" object
they directly modify
't' object data
so, Test class
here is mutable class

Com
wh
set
on
th
to
m
de
set
to
set
here
Test
is c
inmut
cla

```

class Test
{
    int a;
    String b;

    Test (int a, String b)
    {
        this.a = a;
        this.b = b;
    }

    Test ()
    {
    }

    public Test setDate1 (int a)
    {
        Test temp = new Test();
        temp.a = a;
        return temp;
    }

    public Test setDate (String b)
    {
        Test temp = new Test();
        temp.b = b;
        return temp;
    }

    public String toString ()
    {
        return "a=" + a + " b=" + b;
    }
}

```

Concl:-
 when setDate1(),
 setDate2() are called
 on 't' object,
 they are not modifying
 't' obj date,
 moreover they
 are creating and
 returning new objects
 having modification
 related date, so
 here

main (String[] args)

```

{
    Test t = new Test (10, "babu");
    s.o.p ("t obj date " + t.toString());
    Test t1 = t.setDate1 (20);
    Test t2 = t.setDate ("hello");
    s.o.p ("t obj date " + t.toString());
    s.o.p ("t1 obj date " + t1.toString()); // mutable
    s.o.p ("t2 obj date " + t2.toString()); // mutable
}

```

immutable

mutable

No java

Test ⇒ while developing use define immutable class. No java
 class is called method of that class should have logic of modifying
 immutable class. current invoking object date. should
 if modification is required these method in that
 construct new object and should keep modified date
 new object.

4/8/10 IDENTIFIERS - (ALGORITHMS)

⇒ Identity value of this POJO class object is the criteria value for this sql to perform synchronization with this POJO class object and table row.

→ Hibernate supply lot of predefined algorithm as identity value generator for POJO class object.

Most of these algorithms generate dynamic and unique value as identity values of this POJO class objects.

→ These algorithms are predefined classes supplied by HBAPI, implementing org.hibernate.id.IdentifierGenerator

all these classes having nick names (or) short names to utilise.

Nick Name

Algorithm class name

1) assigned \longrightarrow org.hibernate.id.Assigned
default algorithm.

2) increment \longrightarrow org.hibernate.id.IncrementGenerator

3) identity \longrightarrow org.hibernate.id.IdentityGenerator

4) sequence \longrightarrow org.hibernate.id.SequenceGenerator

5) hilo (High and low) \longrightarrow org.hibernate.id.TableHiloGenerator

6) seqhilo \longrightarrow org.hibernate.id.SequenceHiloGenerator

8) guid \rightarrow org.hibernate.id.UUIDGenerator

9) native \rightarrow —

10) select \rightarrow org.hibernate.id.SelectGenerator

11) foreign \rightarrow org.hibernate.id.ForeignGenerator

\rightarrow To specify these algorithms use `<generator>` tag (sub tag of `<id>`) in hibernate mapping file.

If no algorithm is explicitly specified in mapping file, H/S shw ~~looks to~~ take assigned as default algorithm.

\rightarrow while working with any identity value generator algorithm considers the following two rules

① make sure that the type of the identity value generated by algorithm is comparable with datatype of identity field member variable of POJO class.

② make sure that underlying DB shw supports the chosen algorithm.

ASSIGNED ALGORITHM -

This algorithm lets the application developer to assign identity value to HB pojo class object manually before calling session.save() (or) other method to insert the record.

code in mapping file.

employee.hbm.xml :-

<hibernate-mapping>

<class name="EmpBean" table="Employee">

<id name="no" column="EID"> ! - singular identity field cfg ->

<generator class="org.hibernate.id.Assigned" />

</id>

<property name="fname" column="FIRSTNAME">

≡

</class>

</hibernate-mapping>

NOTE :- while testing all the algorithms, make sure that the client application is inserting record by calling session.save(), as shown below.

≡

EmpBean eb = new EmpBean();

eb.setNo(235);

eb.setFname("x1");

eb.setLname("y1");

eb.setMail("x1@y1.com");

Transaction tx = ~~session~~ session.beginTransaction();

Integer idval = (Integer) session.save(eb);

System.out.println("id value is " + idval.toString());

tx.commit();

session.close();

factory.close();

Assigned algorithm can generate any type of identity values and assigned algorithm is compatible with all DB S/W.

⇒ Increment algorithm - (existing value + 1)

This algorithm generates identity value of type long, short or int, this algorithm works with all DB S/Ws., this algorithm uses max value + 1 formula on identity field related table column values to generate new identity values.

Example code -

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

⇒ Identity Algorithm - X ORACLE (Not!)

supports identity columns in DB2, MySQL, SQL SERVER, SYBASE, HYPersonic SQL DB S/W.

This algorithm returns identity value of type long (or) short (or) int, this algorithm does not work with Oracle DB S/W.

MySQL DB S/W table column value can be made as identity column by applying auto increment constraint in the column (To apply this constraint the column already should contain primary key constraint)

identity field related table column values to generate next identity value irrespective of whether records in the table deleted or not that means it also considers deleted record values while picking-up ~~new~~ maximum value from the table column.

ex-

```
<id name="no" column="EID">
```

```
<generator class="identity"/>
```

```
</id>
```

Note:- while testing above code use my SQL DB SW and apply following constraints on the EID column of Employee table.

UNIQUE, NOT NULL, primary key, auto increment

*** Q) what is the diff b/w increment algorithm and identity algorithm?

Increment

Identity

1) Identity field member variable related column in DB, need not to have any constraint to work with this algorithm

1) Identity field related column in DB table should be taken as identity column by applying auto-increment constraint to work with this algorithm.

2) When all records of the table are deleted increment

2) If all records of table are deleted after applying

mySQL generates 1 as the identity value.

and working with identity algorithm, This algorithm still considers deleted record values to generate new identity value as discussed above.

③ ~~HIS~~ HIS slw generates identity value for HIS POJO class object, when increment algorithm is used.

③ DB slw generates identity value based on identity column behaviour and gives that value to HIS slw then HIS slw assigns that value as identity value for POJO class object.

④ This algorithm works with all DB slws.

④ this algorithm works with only that DB slws which support identity columns.

5/8/10

SEQUENCE ALGORITHM:-

This algorithm uses sequence, created in DB, PostgreSQL, ORACLE, SAP, HCKOI, Intabase DB slw to generate identities value of type long, short (int)

NOTE! - this algorithm doesn't work with MySQL DB slw because the MySQL DB slw doesn't support sequences.

NOTE! - To pass input value to any algorithm configured in HIS mapping file use <param> tag as the subtag or <generator> tag

→ create sequence my001-seq1 increment by 5; ORACLE
select * from used_seqsequence;

step 1:- create sequence in ORACLE DB slw as shown below

only create sequence myora_seq1 increment by 5;

step 2:- write following code in HB mapping file.

```
<id name="no" column="EID">  
<generator class="nosequence"> Algorithm name  
<param name="sequence">myora_seq1</param>  
</generator>  
</id>
```

parameter name
sequence-name
Created in oracle slw

NOTE:- we can work with sequence algorithm only with those DB slws which support sequences creation.

when sequence algorithm is used the HB slw gets identity value from DB slw by using the specified sequence name.

hilo:-

This algorithm uses hilo algorithm to efficiently generate identity values of type long, short or int by using helper table column value and given parameter values (Max-lo) as source of values.

This algorithm works with all DB slws; this algorithm expects the following 3 parameter values

- they are table (expects helper table name),
column (expects helper column name in the table),

how

Helper Table column value always increments a number incrementation indicating how-many records are inserted in the DB table by using hibo algorithm based identity values.

5;

example code:-

name

step 1:-

create helper table having helper column. Make sure that this helper column having 1 numeric value ~~SQL create table~~ as initial value.

SQL) create table mytable(mycol number);

SQL) insert into mytable values (1);

step 2:- configure hibo algorithm in HB mapping file as shown below.

gets

brief

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

<generator class="hibo">

<param name="table">mytable </param>

<param name="column">mycol </param>

<param name="max_lo">10 </param>

</generator>

</id>

This algorithm uses following formulas to generate initial identity value and next identity values.

formula to generate initial value:-

$(max_lo + 1) * helper_col_val$
(00)

$(max_lo_val) * helper_col_val + helper_col_val$

formula to generate next values:-

$(max_lo_val + 1)$


```

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

```

if the underlying DB sw doesnot support both identity and sequence algorithms then the native algorithm internally uses 'hilo-algorithm'

seqhilo algorithm -

This algorithm is enhancement of hilo-algorithm which internally uses given sequence name and parameter values to generate identity values of type long, short or int

this algorithm doesnot expect special table name from the programmer, this algorithm works with only that DB sw. which supports DB sequences.

(works with oracle, doesn't works with mysql)

Parameter names for this algorithm are seq, max-lo

⊙
example code:-

step 1:- make sure that my000_seq1 is available in oracle DB sw.

step 2:- configure seqhilo algorithm in hibernate mapping file as shown below.

```

<id name="no" column="EID">
  <generator class="seqhilo">
    <param name="sequence"> my000_seq1 </param>
    <param name="max-lo"> 10 </param>
  </generator>
</id>

```

next val in sequence + increment by val of seq + 1)
* (max - lo val) + 1

Formula to generate next identity value:-

previous identity val + (max - lo val + 1) * increment by
val of sequence)

06/08/00

Uuid algorithm:- universal uniqueid

This algorithm uses, network IP address as base
value to generate string type, hexadecimal notation
value as identity value for POJO objects.

This value contains 32 digits in hexadecimal notation

→ This algorithm works with all DB S/Ws.

Example codes:-

① take "no" number variable of EmpBean class a
String variable.

② change EID column datatype to varchar2(35) in
Employee table;

SQL → ALTER table Employee modify EID varchar2(35);

③ configure uuid algorithm as shown below in
mapping file.

```
<id name="no" column="EID">
```

```
<generator class="Uuid"/>
```

```
</id>
```

+1)

insert the below.

```
Transaction tx = session.beginTransaction();
```

```
String idval = (String) session.save(obj);
```

```
System.out.println("id value is" + idval.toString());
```

```
tx.commit();
```

=

guid :- global unique id

this algorithm ^{uses} generate DB generated GUID ^{String} as identity value.

This only work with SQL SERVER, MY SQL DBMS

~~scope~~

select :-

This algorithm generates identity value by executing trigger program in underlying DBMS

Foreign algorithm :-

In order to use, identity value of associated POJO object as the identity value of current POJO object use Foreign algorithm. This algorithm is very useful while working with one-to-one relationship.

conclusion on algorithms :-

JL/PL decides identity value generator algorithm for POJO class object and HIS developers configures them programmatically.

If identity field DATATYPE is numeric datatype it is recommended to use increment (or) native algorithm.

numeric datatype then use assigned algorithm.

only on singular identity field configuration, we can use these algorithms.

while work with composite field identity configuration there is no possibility of working with these algorithms.

use <id> as ~~id~~ HBM file for singular identity

field configuration

use <composite-id> tag for composite identity

field configuration

=

*** Q) How to generate the following type values as identity values for POJO class object?

→ AB001, AB002, AB003

ans - To generate these values write logic manually

by using string manipulation concept and use 'assigned' algorithm to set these values as

identity values for POJO class objects.

=

If ~~table~~ DB Table name is same as POJO class name and if @Column names are same as POJO class no. variable name then there is no need of specifying table name and column names in HBM mapping file while performing OR Mapping configuration.

```
public class EmpBean
{
    int no;
    String fname, lname, mail;
    setters & getters
}
```

DB table:-

EmpBean (table name)
no (numbers)
fname (Vc(20))
lname (Vc(20))
mail (Vc(20))

for the above setup, we can write code in mapping file as shown below.

```
<hibernate-mapping>
<class name="EmpBean">
    <id name="no">
        <generator class="increment">
        </id>
    <property name="fname"/>
    <property name="lname"/>
    <property name="mail"/>
</class>
</hibernate-mapping>
```

⇒ To observe that table name and column names are not specified in HB mapping file.

specify POJO class name as fully qualified class name as shown below.

```
<hibernate-mapping>
```

```
  <class name = "PI.EmpBean" >
```

```
    ≡
```

↳ java package name

```
  </class>
```

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

(or)

```
<hibernate-mapping package = "PI" >
```

```
  <class name = "EmpBean" >
```

```
    ≡
```

```
  </class>
```

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

⇒

HIBERNATE SW supply some built in tools to create or alter DB tables in underlying DB SW. These tools are also useful to generate HIB POJO class dynamically they are.

① Schema Export

② Schema Update

③ Code Generator.

→ Schema Export tool always create new table

→ Schema update tool can create new table or can alter existing table by adding new columns.

These tools are very useful to create tables dynamically in new DB SW based on HIB mapping file.

name

new DB SW, ~~also~~ than these tools are very useful.

-> SchemaExport tool always create new table in DB SW.
if table is already available then it creates new table by dropping the existing table.

we can pass instruction to this tool from
hib mapping file by using length, type, unique,
not-null and etc attributes.

Example :-

step 1:-

prepare hib mapping file by having ORMapping
configuration and instructions

<hibernate-mapping>

<class ~~name~~ name = "EmpBean" table = "Employee" >

<id name = "no" column = "EID" length = "10" type = "int" >

<generator class = "increment" /> ↳ size of the column

</id >

hib datatype

<property name = "fname" column = "FNAME" length = "20"

not-null = "true" unique = "true" />

Applies Not null constant on column. Unique constraint on column.

<property name = "lname" column = "LNAME" length = "25"

type = "string" />
small

<property name = "mail" column = "EMAIL" length = "30"

type = "string" />

</class >

</hibernate-mapping >

step 2:- execute SchemaExport tool command from command prompt
where hib configuration file and mapping file reside.

> java org.hibernate.tool.hbm2ddl.SchemaExport (space)

→ Hibernate Datatypes are the Bridge Datatypes b/w
Java Datatypes and underlying DB SQL type Datatypes
They help the HIB SQL to decide the Datatypes in
DB SQL while creating columns in DB table based
on HIB mapping file by using SchemaExport or
SchemaUpdate Tool as shown above.

The HIB Datatypes are :-

int, long, float, double, byte, string ... etc

→ you can also specify java wrapper Datatypes instead
of HIB Datatypes in mapping file They are

java.lang.Integer

java.lang.String

java.lang.Byte

java.lang.Float

java.sql.Date - - - - - etc

NOTE - specifying both these datatypes (hibernate, java
wrapper datatypes) is optional in hibernate mapping file

in this situation the Datatypes of POJO class member
variables will be utilized as Hibernate Datatypes while
creating columns in dynamic table, while working
with SchemaExport, SchemaUpdate tools.

7/8/10

SchemaExport, SchemaUpdate Tools are just given to
create a new table or to alter existing table structure
in underlying DB SQL.

and can use these tools to copy data

purpose take the support as 3rd party Tools available in the internet.

→ SchemaUpdate tool can create the new table, if table is not already available

SchemaUpdate tool can use existing table or can alter existing table if table is already available, this tool ~~can~~ alter existing table only by adding new columns to table, based on ~~new~~ new property configured in hibernateMapping file.

example:-

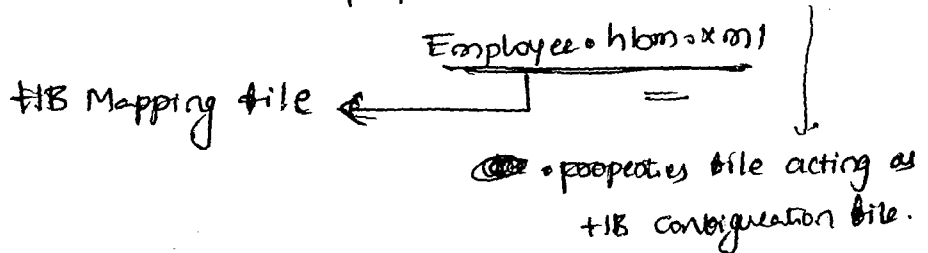
step:-

```
> java org.hibernate.tool.hbm2ddl SchemaUpdate <space>  
--config = mycfg.xml
```

code generator:-

this tool is supplied in HB 2.x, but not given in 3.x s/w. This tool is capable of generating HB persistence class (HB POJO class) based on hibernateMapping file.

```
> java net.sf.hibernate.tool.hbm2java CodeGenerator  
--properties = hibernate.properties <space>
```



to perform hbm2ddl operations.

⇒ CodeGenerator tool design to perform hbm2ddl operations.

⇒ In Hibernate 2.x the HIB configuration file is properties file.

⇒ we can use SchemaUpdate, SchemaExport tool along with HIB application execution by passing special properties in HIB configuration file.

hibernate.hbm2ddl.auto
possible values are.

1. create (uses SchemaUpdate tool internally)

2. update (default) (uses SchemaUpdate tool internally)

3. create-drop (uses SchemaExport tool internally)

Example ①:-

→ In configuration file:-

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

Always create new table in each execution,

Example ②:-

→ configuration file:-

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

can create new table (or) can use existing table (or)

can alter existing table by using SchemaUpdate tool

⇒ (Most Recommended value for hbm2ddl.auto property is update)

<property name="state.hibernate.hbm2ddl.auto">create-drop

</property>

creates tables dynamically in DB slw., when SessionFactory object is created and drops all these tables when SessionFactory object is closed from the application.

→ create-drop ^{of} hibernate.hbm2ddl.auto property

is quite useful in the testing mode of project. Table creation and destruction is required after testing the project with trial (dummy) data.

Connection Pooling

⇒ Connection Pool is a Factory that contains set of readily available JDBC connection objects.

SessionFactory of HB application represents this JDBC connection pool.

while creating each session object based on SF object, connection object of ConnectionPool will be utilized.

HB slw can work with ③ types of

Connection Pools

they are ① HB slw supplied built-in connection pool. (default) ↑

② Third party supplied Connection Pools (JDBC like c3p0, proxool)

③ web-server (or) application server managed JDBC connection pools.

not suitable for production mode environment
of project because of its poor performance.

⇒ The default max size of this built-in connection-pool is '20'.

This can be controlled by using following property of HIS configuration file.

```
<property name="hibernate.connection.pool_size" value="30"/>
```

Q) What is the JDBC connection pool, that have used in your project where HIS persistence logic is placed?

⇒ If the project type is stand-alone application, which can run outside the web servers and application servers use 3rd party connection pool SWs like C3PO (or) Poolool.

If project is ServerManaged application like web-application or EJB component use that ServerManaged JDBC connection pool.

~~***~~
NOTE:- Don't use HIS supplied built in connection pool any project (performance affected)

Our Hibernate Application:

NOTE: - (1) Based on the value passed to ~~the~~

hibernate.connection.provider-class property in the

Configuration file. The HBS SLW resides.

(2) The connection SLW that it is used to create JDBC connection pool represented by Session Factory object.

⇒ This property takes a built-in class name as value. The possible values are.

(1) org.hibernate.connection.DriverManagedConnectionProvider
(default value uses hibernate SLW supplied built-in connection pool.)

** (2) org.hibernate.connection.DatabaseConnectionProvider
(uses web server (OS) Application Servers managed JDBC connection pool.)

(3) org.hibernate.connection.C3POConnectionProvider
(uses C3PO SLW managed connection pool.)

(4) org.hibernate.connection.PoolConnectionProvider
(uses the pool SLW managed JDBC connection pool.)

→ Step 1 :- Add C3PO Pool related Connection Provider classname in HBS configuration file of the application.

→ `<property name = "hibernate.connection.provider-class" value = "org.hibernate.connection.C3POConnectionProvider" />`

Step 1:- Run ...

HB configuration file.

① `<property name="hibernate.c3po.max_size">30</property>`

`<property name="hibernate.c3po.min_size">5</property>`

`<property name="hibernate.c3po.timeout">5000</property>`

`<property name="hibernate.c3po.acquire_increment">2</property>`

→ Add `hibernate-home\lib\c3po-0.9.1.jar` file to classpath.

(This jar file represents c3po ConnectionPool class).

Step 2:- Run the client application in the regular model.

Procedure to work with Pool JDBC ConnectionPool in our HB Application:-

Step 1:- Add Pool pool related connection provider class in HB configuration file.

`<property name="hibernate.connection.provider_class" org.hibernate.connection.PoolConnectionProvider</property>`

Step 2:- Add Pool Pool related property is in

HB Configuration File.

<property name = "hibernate.pool.xml" > myfile.xml </property>

</properties>

<properties>

Note - here we configure the another xml file myfile.xml helps xml file for hibernate configuration.

<properties>

Step 1 - develop above specified myfile.xml containing

</properties>

properties of Pool s/w related connection Pool.

</properties>

myfile.xml

file

<pool-config>

<pool>

<alias> mypool </alias>

Pool

<driver-url> jdbc:oracle:thin:@localhost:1521:scott </driver-url>

</driver-url>

<driver-class> oracle.jdbc.driver.OracleDriver </driver-class>

</driver-class>

</pool>

<driver-properties>

<property name = "user" value = "scott" >

<property name = "password" value = "tiger" >

</driver-properties>

nonPool

<minimum-connection-count> 10 </minimum-connection-count>

<maximum-connection-count> 20 </maximum-connection-count>

</pool>

less

</pool-config>

>

overrides

</properties>

Note 1 - place this file in the directory where the same

hibernate configuration file resides.

Note 2 - since myfile.xml itself is containing user, driver class, and user name and password details so there is no need of specifying same details.

Step 1:- Add `libeonate-home\lib\pool-0.8.3.jar`
to the class path (This jar file represents
pool slw)

Step 2:- execute the client application.

Understanding Server Managed Connection Pools:-

Server Managed Connection Pool will be managed in web-server or Application Server SW.
JDBC DataSource object represents one JDBC
Connection Pool. In order to get access to each
connection object of JDBC connection pool we need
to use JDBC DataSource object.

JDBC DataSource object means it is the
object of a class that implements javax.sql
DataSource interface. To provide global visibility
to DataSource object it will be registered in
registry slw having nickname.

Every webserver / Application Server ^{slw} ~~object~~ supplies
one built-in registry slw.

→ Examples of registry slws :-

- ① RMI - Registry :- ① COS (Common Object Service)
- ② LDAP Registry (Lightweight Directory Access Protocol)
- ③ DNS (Domain Naming Service)
- ④ weblogic Directory Registry and etc. ...

with Registry slw's:

* Java Application can perform insert, update, delete and select operation on DB table by using JDBC API

* Similarly Java Application can perform Bind, ReBind, unBind, lookup and List operation on Registry slw by using JNDI API.

Bind → Adding object to registry (insert)

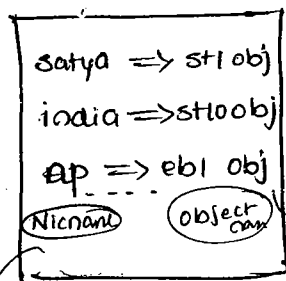
ReBind → Replacing existing object with new object (update)

unBind → Removing object from Registry (delete)

List operation → Retrieving all objects that are bound to Registry (select)

Look up operation → Searching and getting object based on nickname/JNDI name (select)

Registry slw

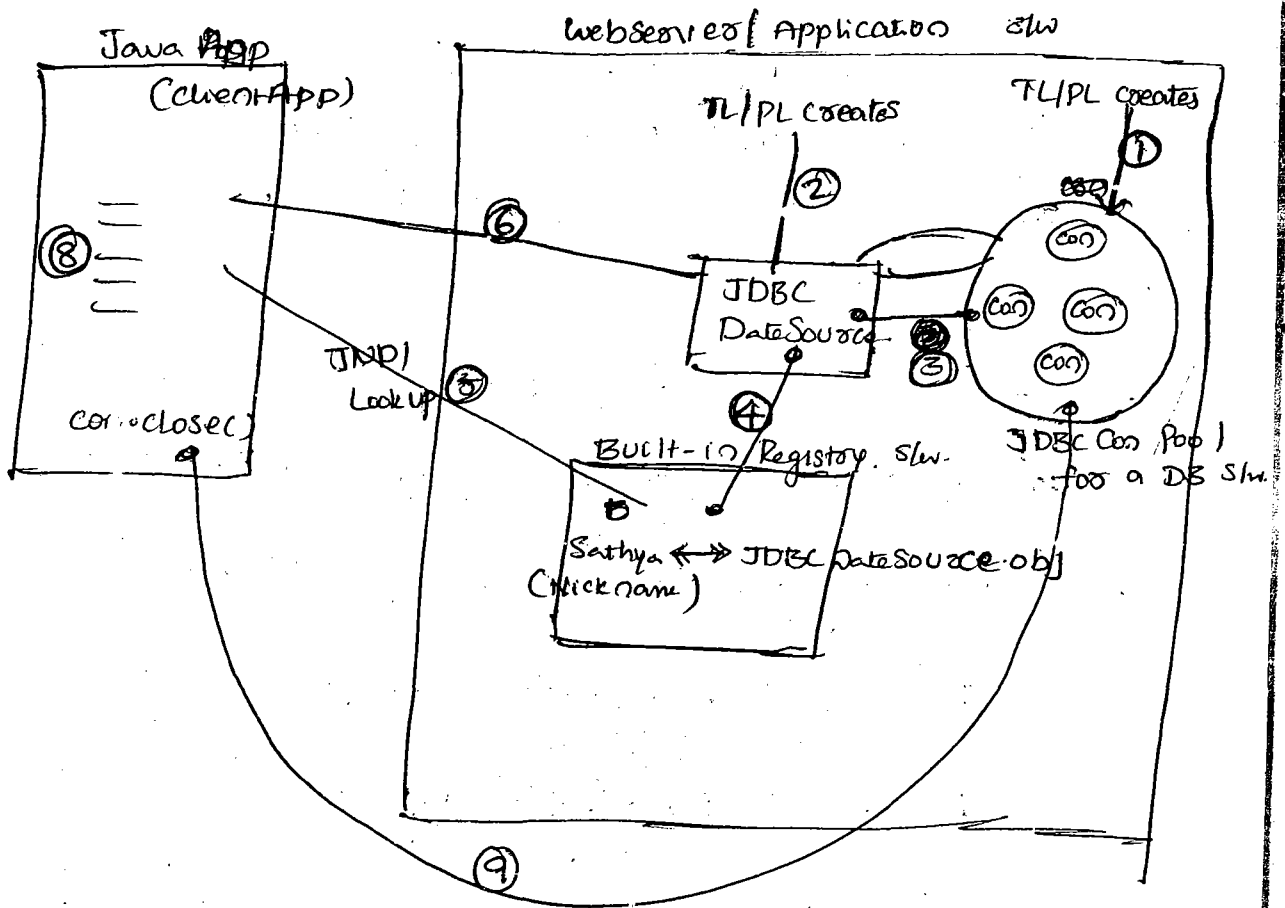


Nicknames (or) JNDI Names

objects that are Registered

→ To get access to JDBC connection objects or connection Pool (Server Manager) programmer must get access to JDBC DataSource object which is representing that connection

Protocol



① TL/PL creates JDBC ConnectionPool for DB SW

②, ③, ④ TL/PL creates DataSource object pointing to ConnectionPool and registers the DataSource object with registry having nickname for global visibility.

⑤ Client app uses JNDI and gets DataSource object from Registry through Lookup operation

⑥, ⑦ Client application uses this DataSource object to get JDBC Connection object from ConnectionPool.

⑧ Client app ~~then~~ creates other object based on this connection object and writes the persistence data.

object back to connection pool so this connection
becomes free to give service to client application

10/8/10

WEBLOGIC :- (9/10)

- Application Server = webContainer / servlet container + EJB container
+ Additional services
- webserver = webContainer / servlet container + middleware
services
- Application server SW is enhancement of webserver
having enhanced facilities and features.

weblogic 11.3.6 -

type :- Application server SW

Version :- 9.0.x (Compatible with J2SE 1.5)

Vendor :- BEA Systems (Oracle SW).

port no :- 7001 (default)

Commercial SW :-

to download SW :- www.commerce.bea.com

for document :- www.edocs.bea.com

allows to create domains.

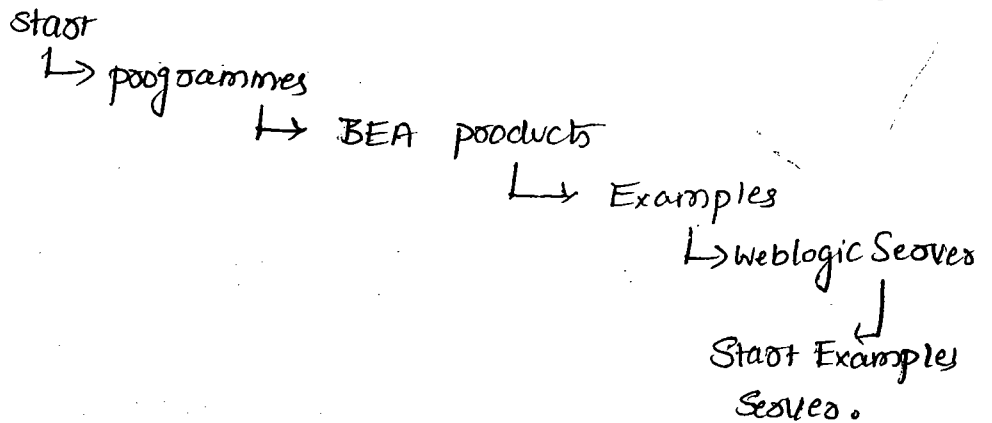
default domain name is :- exampleserver.

if multiple projects of a companies are using
same weblogic SW, then weblogic SW will be installed
only once on a common computer having multiple
logical domains for multiple projects on one per
project basis. Each logical domain acts as one application

Oracle, JDBC DataSource in Example Server Domain

ab weblogic 9.0.x :-

Step 1 :- Start example server domain ab web-logic.



Step 2 :- open administration console ab
example server domain.

⊗ open Browser Window

↓
type "http://localhost:7001/console" url

usr :- weblogic

pwd :- weblogic

↓
login.

Step 3 :- create JDBC DataSource from Administration console

pointing to JDBC Connection Pool for Oracle

Admin Console

↳ Lock & Edit

↳ services

↳ JDBC

↳ Data Sources

↳ New

↳ Name

Name :- myorads

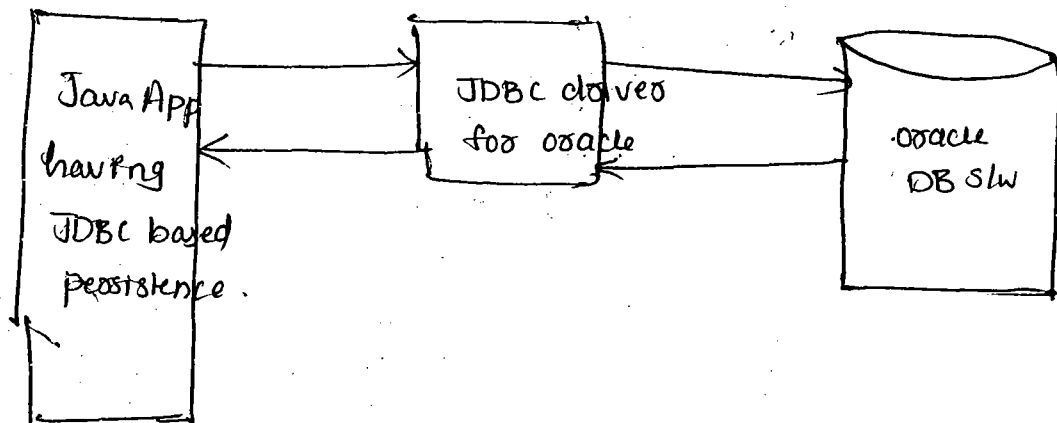
logical name

Activate changes

NOTE:- The Moment you create JDBC DataSource. The JDBC DataSource object will be automatically registered with the weblogic supplied built-in weblogic Directory Registry slw having Nickname

SAPJAJNDI.

→ Communication b/w Java App and DB:-



JDBC Connection object in Java App

communicated b/w Java App and Registry slw:-

dependencies connectivity with DB slw. To create this connection object the following details are required

- ① JDBC Driver class name
- ② DB url
- ③ DB username
- ④ DB Password

JDBC Driver is the Bridge b/w Java Application and DB slw, every JDBC Driver is identified with its Driver class name.

and Registry slw

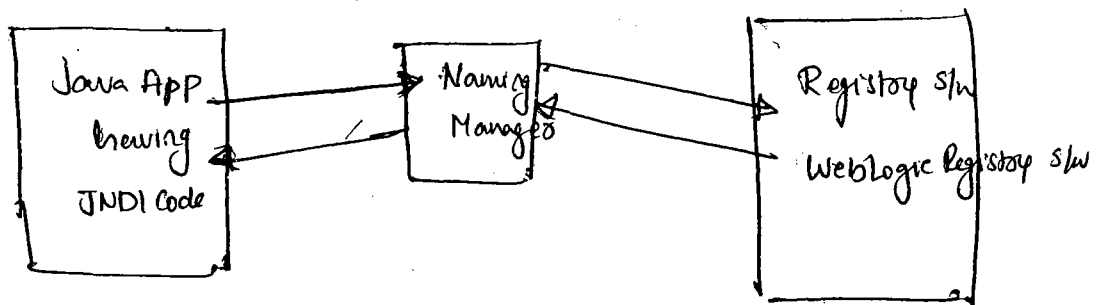
Every Naming Manager will be identified with its InitialContextFactory class name. This class name change based on Naming Manager and Registry slw we used:

InitialContext object of java application represents connection with Registry slw and provides environment to perform operations on Registry slw.

To create this object we need following two details which are called JNDI properties:

① InitialContextFactory class name of Naming Manager

② Provider url of Registry slw.



→ JNDI properties values of weblogic Directory Registry slw

→ InitialContextFactory class name :-

weblogic.jndi.WLInitialContextFactory

This class is available in ~~BEA~~ 11

BEA-HOME \ weblogic \ server \ lib \ weblogic.jar file

→ provider url :- ~~to~~ t3 : // <hostname> : <portno>

Connection Pool in stand alone Environment based

Hibernate Application :-

Step 1. - Configure connection provider class, HIB Configuration file.

```
<property name = "hibernate.connection.provider_class" >  
org.hibernate.connection.DataSourceConnection  
Provider </property>
```

2. - Specify JNDI property values, related to weblogic directory Registry in HIB configuration file.

```
<property name = "hibernate.jndi.url" > t3://eh:7001 </property>
```

```
<property name = "hibernate.jndi.class" > weblogic.jndi.  
WLInitialContextFactory </property>
```

↓
InitialContextFactory class

3. - Specify JNDI Name (or) NickName of JDBC DataSource created in weblogic server

```
<property name = "hibernate.connection.datasource" >  
SathyaJndi </property>
```

Note - Since JDBC DataSourceName, DBURL, DB username,

DB password values are specified in server

admin console while creating JDBC Connection Pool

So there is no need of specifying them in

HIB configuration file.

file to the class path

→ Make sure that example server domain of weblogic
weblogic90x is Running Mode and execute the client Application

GlassFish

type :- Application Server SW

version :- 9.0x (compatible with j2sdk 1.5)

netbeans 6.0x SW use glassfish as built-in server

Vendor :- sun/micro systems (Oracle Corp)

open source SW

default port no :- 4849 for Admin Console.
8081 for running W/A.

for download :- www.java.sun.com/javaee/5/docs.

jar file that separates JLEE API - jarpe.jar

registry SW name :- Glassfish Registry

11/08/10

Procedure to create jdbc datasource and jdbc connection pool for Oracle in GlassFish application server :-

STEP 0 :- start Glassfish server

start → programs → sun/micro systems → Application Server
start default server ←

STEP 1 :- open admin console of Glassfish server

start → programs → sun/micro systems → Application Server → admin console
username :- admin
password :- admin

step 3 :-

create jdbc connection pool

~~admin console~~
admin console

↳ Resources

↳ jdbc

↳ connection pool

↳ new

↳

name : mypool1

Resource type : javax.sql.DataSource

DB Vendor : oracle

↳ next

→ Initial Capacity : 10

Maximum Pool size : 20

pool size : 2

~~DB name write~~ "satya"

DataS

✓ DataSourceName : myds

e pass

✓ Database Name : ~~for satya~~ satya

password : tiger

port name : 1521

server name : lh

:lh:

url : jdbc:oracle:thin:@lh:1521:satya

user : scott

Add properties

Driver class : oracle.jdbc.driver.OracleDriver

↳ finish

↳ mypool1

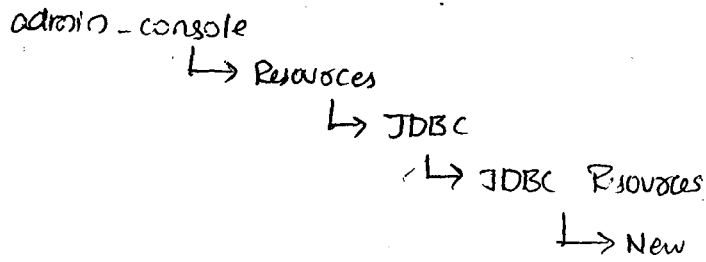
↳ ping

ping succeeded

↳ save

GlanFish | AppServer | Lib is contain @ jdbc4-jar

step 1) Create JDBC DataSource pointing to the above created connection pool



JNDI Name: GEDS Jndi ← <any-name>



⇒ Procedure to use GlanFish Server managed JDBC Connection pool in standalone environment based HIB application.

step 1:- same as weblogic server.

step 2:- specify JDBCDataSource JNDI name, in HIB configuration file.

<property name = "hibernate.connection.datasource" > GEDS Jndi </property>

⇒ NOTE:- there is no need of adding InitialContextFactory classname, provides url value in HIB configuration file, while working with GlanFish Server supplied Registry SW.

⇒ Add following two jar files to the classpath.

~~AppServer~~ @

① GlanFish Home \ AppServer \ appserver-07.jar

② " \ javaee.jar

⇒ execute the client Application.

WEB RESOURCE PROGRAMMES :-

→ For this add HBAPI related main jar file (hibernate.jar) in the classpath and add HBAPI related main and dependent jar files in WEB-INF/lib folder of web-application =

→ while developing HIB persistence logic in java application it is recommended to create HIB Session object in one time execution block and it is recommended to use this HIB Session object for persistence operations in repeatedly executing blocks.

Application Type	Place to create HIB Session obj	Place to use HIB Session obj
1) AWT/Swing Frame	static block / Constructor	Event Handling methods
2) Applet / JApplet	static block / constructor / init()	Event Handling methods
3) Servlet	static block / Constructor / init()	service (-, -)
4) JSP	<pre> <%! public void jspInit() { } </pre>	<pre> <% script-let-code %> </pre>

5) EJB Session Bean Component

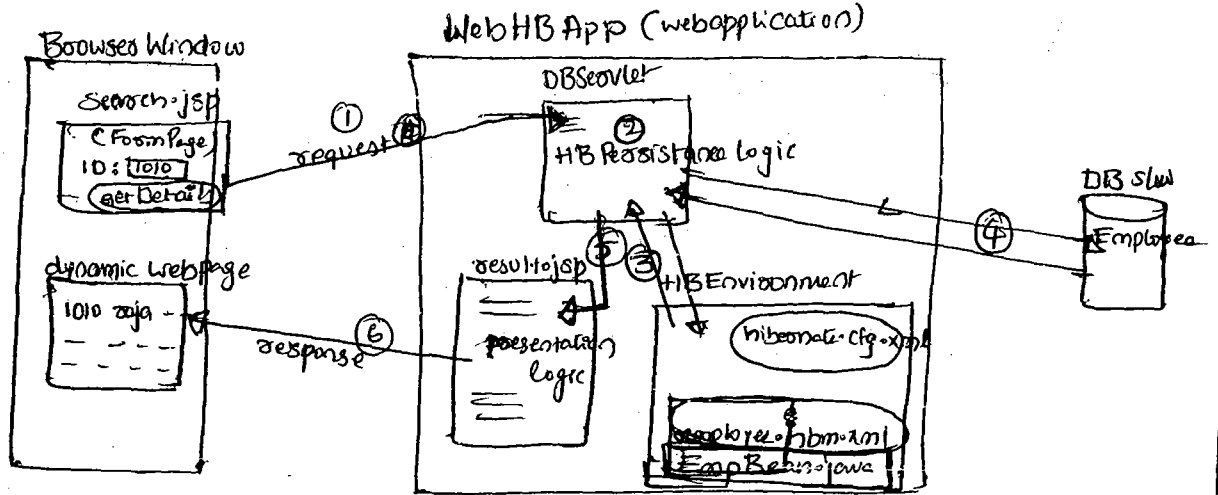
ejbCreate() / constructor / static block

Business Method

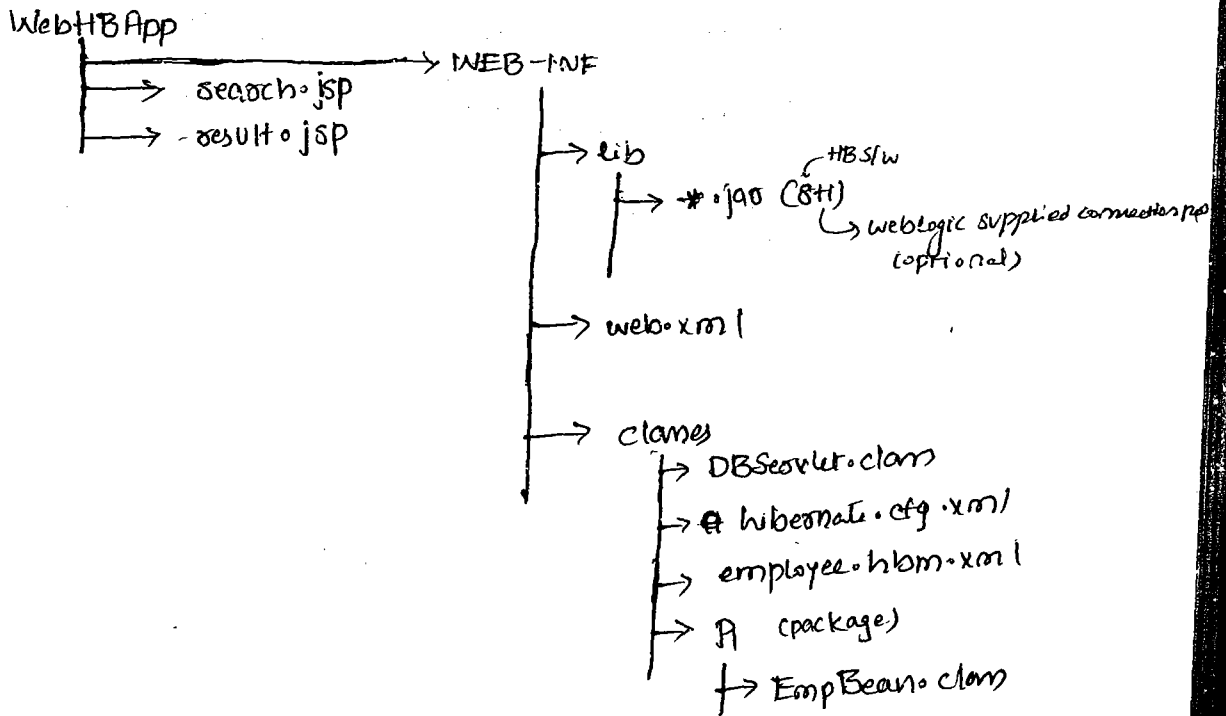
6) Struts App

static block / constructors

execute C-1-1-1



Deployment Directory structure:-



- In the above application DBServlet is using Hibernate logic to interact with DB/S/W.
- In order to pass data from server to JSP when request to JSP the server can use request attributes.

1.3 (108)
 id
 application
 to
 on
)
 29
 ing
 -)
 script
 let
 code

that class WEB-INF/classes folder than the class must be there in package.

→ The above application is going to be deployed in weblogic 9.x server and it is going to use weblogic9.x server managed. JDBC connection pool.

→ If any application that is running outside the weblogic server wants to interact with weblogic directory Registry then the application must use JNDI Properties like InitialContextFactory class name, provider url and etc..

→ If Application deployed in weblogic server wants to interact with weblogic directory Registry then there is no need of passing the above said JNDI properties.

According to this discussion there is no necessity of passing JNDI property values in the configuration file of above application.

Source Code: -

hibernate.cfg.xml :-

- DTD -

<hibernate-configuration>

<session-factory>

<property name = "hibernate.dialect" > org.hibernate.dialect.
dialect.Orcle9Dialect </property>

<property name = "hibernate.connection.provider-class" >
org.hibernate.

<property name = "hibernate.connection.datasource" > Sathya Jndi

<mapping resource = "Employee-hbm.xml" />

</session-facets>

</hibernate-configuration>

→ Do observe that the configuration file not having hibernate-jndi class, hibernate-jndi-usl properties holding JNDI properties. (InitialContextFactory class name, provider url)

12/08/10

EmpBean.java:-

same as first application but make sure that it is available in a package "P1"

web\BAPP\WEB-INF\classes > javac -d . EmpBean.java

Employee-hbm.xml:-

same as 1st applo

Search.jsp:-

<form action = "dbusl" method = "get">

 Employee ID

<input type = "text" name = "tid" />

<input type = "submit" value = "Get Details" />

</form>


```
import javax.servlet.* ;
import javax.servlet.http.* ;
import java.io.* ;
```

```
import org.hibernate.cfg.* ;
import org.hibernate.* ;
```

```
public class DBServlet extends HttpServlet
```

```
{
```

```
    Session sess sess = null ;
```

```
    public void init ()
```

```
{
```

```
        super ("init () of DBServlet");
```

```
        try
```

```
{
```

```
            sess = new Configuration ().configure ().
```

```
                buildSessionFactory ().openSession ();
```

```
        }
```

```
        catch (Exception e)
```

```
{
```

```
            e.printStackTrace ();
```

```
        }
```

```
    } //init
```

```
    public void doGet (HttpServletRequest req, HttpServletResponse
```

```
res) throws ServletException
```

```
{
```

```
    try
```

```
    { // read form data
```

```
        int no = Integer.parseInt (req.getParameter ("tid"));
```

```
        // write HB persistence logic.
```

```
        EmpBean eb = (EmpBean) sess.get (EmpBean.class, new Integer (no));
```

```
        // store result in request attribute to send to
```

```
        @ result.jsp
```

```
        req.setAttribute ("result", eb);
```

```
//forward req to result.jsp.
```

```
RequestDispatcher rd = req.getRequestDispatcher("result.jsp");
```

```
if (rd != null)
```

```
rd.forward(req, res);
```

```
} //try,
```

```
catch (HibernateException he)
```

```
{
```

```
he.printStackTrace();
```

```
}
```

```
catch (ServletException se)
```

```
{
```

```
se.printStackTrace();
```

```
}
```

```
} //doGet
```

```
public void doPost(HttpServletRequest req, HttpServletResponse res)
```

```
{ s.o.p("doPost() at DBServlet");  
doGet(req, res);
```

```
}
```

```
catch (Exception e)
```

```
{
```

```
e.printStackTrace();
```

```
} //do
```

```
} //doPost
```

```
public void destroy()
```

```
{
```

```
s.o.p("destroy() at DBServlet");
```

```
try
```

```
{
```

```
res.close();
```

```
}
```

```
catch (Exception e){
```

```
{
```

```
e.printStackTrace();
```

```
}
```

```
} //destroy
```

```
} //close
```

```
-
```

web.xml :-

configure DBServer by have ~~DB~~ /DBUSI ~~DB~~ url pattern

result.jsp :-

```
<%@page import = "pl.EmpBean" %>
```

```
<% EmpBean eb = (EmpBean) request request.getAttribute ("result"); %>
```

```
<body The Details are </body>
```

```
<% = eb.getNo() %></body>
```

```
<% = eb.getFname() %></body>
```

```
<% = eb.getLname() %></body>
```

```
<% = eb.getMail() %></body>
```

⇒ in WEB-INF/lib folder of above web application
place the HBAPI related HT files.

Procedure to Deploy web-app in weblogic 9.x servers -

Step 1 :- prepare war file on the Deployment Directory
structure of web-application.

> jar cf webHBAApp.war<space> ◀

above step gives webHBAApp.war file representing
jar file

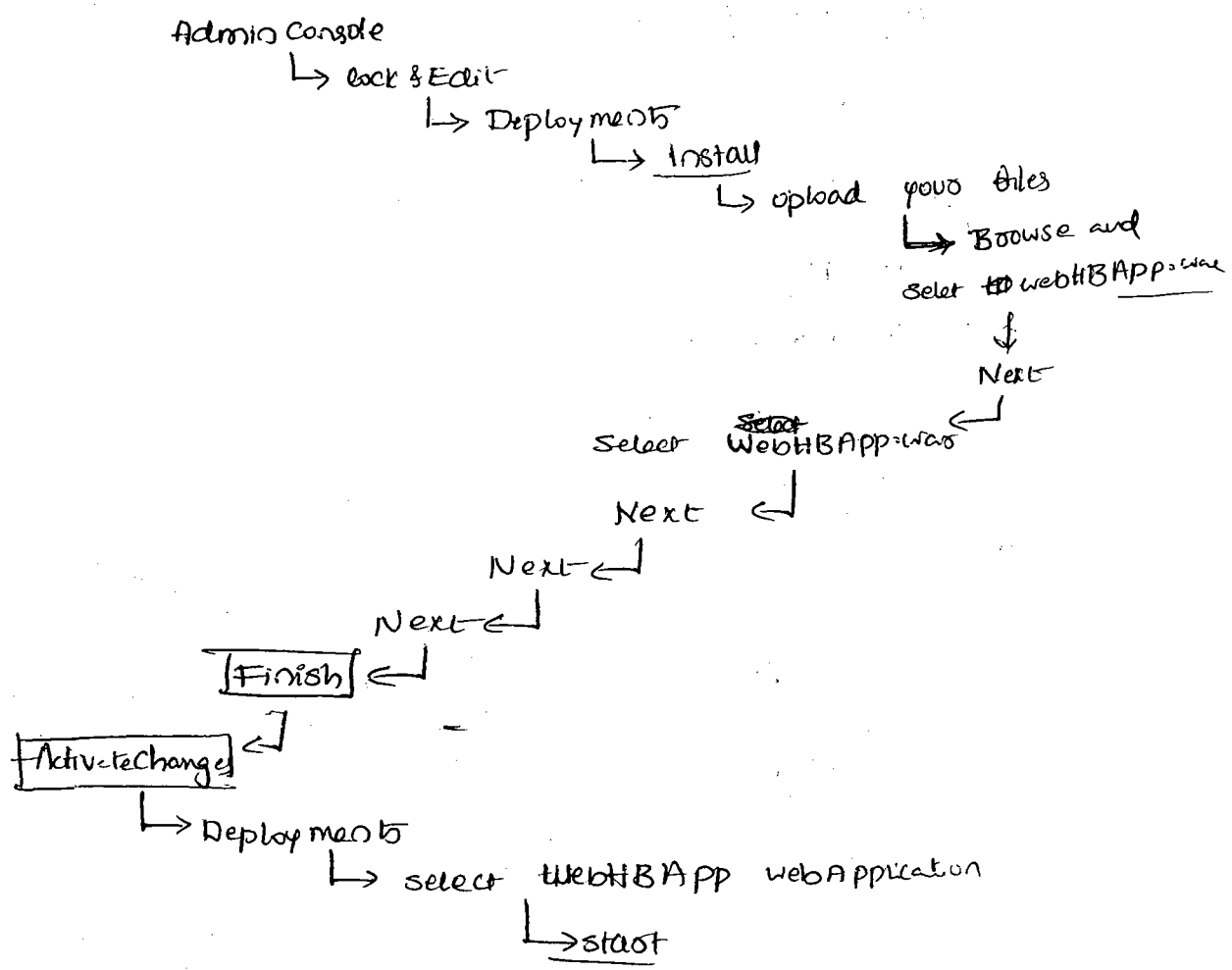
→ There are 3 ways to deploy applications in server

- ① Hand Deployment
- ② console Deployment
- ③ Tool Base Deployment

NOTE :- this application is using console Deployment process.

→ open administration console of example.com | services | deployment
weblogic & etc:-

→ Deploy the war file <webHBApp.war>



Test the webApplication

open Browser window:- <http://localhost:37001/webHBApp/search.jsp>

BULK OPERATION RELATED TECHNIQUES:-

In order to manipulate single row, by taking our choice value as criteria value or to ~~rep~~ manipulate more than one row at a time we can use one of the following technique.

But HQL Most Recommended Technique:

- 1) HQL
- 2) NativeSQL
- 3) Criteria API

HQL:-

it is the most popular persistence technique environment in Hibernate Programming

- ① HQL's are DB independent queries, so these queries based independent logic is DB independent.
- ② HQL queries will be written based on POJO classes and member variables of POJO classes.
- ③ HQL queries are object level queries, so they return HIB POJO class objects as results.
- ④ HQL queries and keywords are very much similar to SQL queries of Oracle.
- ⑤ HIB SW converts HQL queries into SQL queries and sends them to DB SW for execution.

HQL queries supports operators, expression, condition clauses, joins, sub queries, aggregate functions and etc

→ we can use HQL queries for both select & non select operations

→ QueryObject represents HQL Query in Hibernate application, this is the object of a class that implements org.hibernate.Query Interface

→ all Non-select HQL queries must be executed in transaction management environment

→ HQL query support two types of parameters

Limitation:- ① positional parameters ② named parameters.

① HQL query can't perform DDL operations

② HQL queries can't be used in PL/SQL programming

③ HQL queries can't be used to insert single record into table.

④ HQL queries are negligible performance degradation coz of conversions then compare to SQL.

13/08/10

SQL > select * from Employee ↑ table name

HQL > select eb from EmpBean as eb (os)

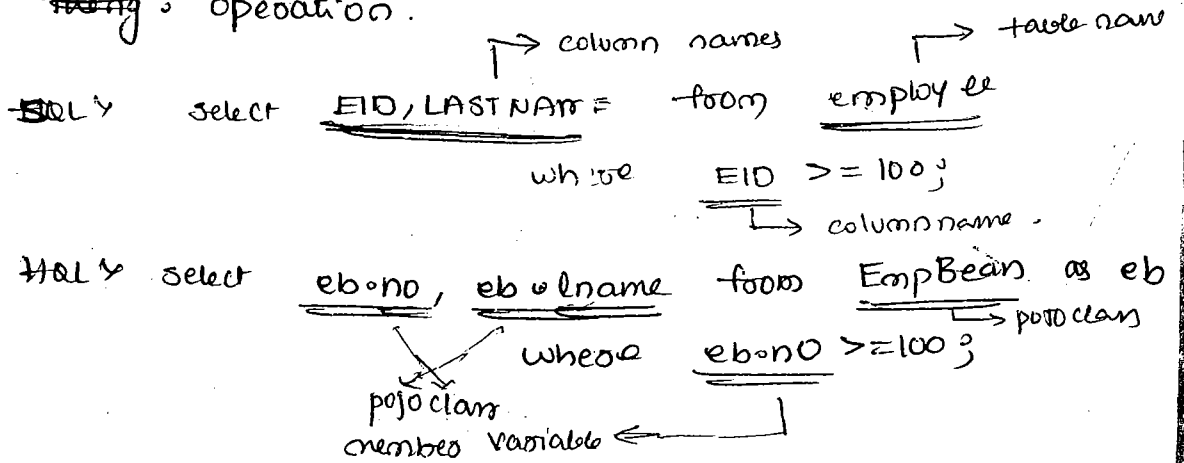
HQL > select from EmpBean as eb (os) ↑ POJO class name

HQL > from EmpBean (os)

HQL > from EmpBean eb

NOTE 1-

when select keyword is there or POJO class members variables are there in SQL query then creating alias name for POJO class is mandatory thing operation.



⇒ HQL keywords are not case sensitive but POJO class name and POJO class member variable used in HQL query are case sensitive.

NOTE ⇒ String values in HQL query should be represented using single code.

ex:- 'raja', 'hyd'

SQL > select count(*) from Employee *
↳ tablename

HQL > select count(*) from EmpBean
↳ pojo class

SQL > delete from Employee where firstname in ('raja', 'ravi')

HQL > delete from EmpBean ~~class~~ as eb where eb.fname in ('raja', 'ravi')

→ To execute select HQL queries use list()
iterate() or Query obj similar to execute

non-select HQL queries call executeUpdate() on
Query object.

→ execution of HQL Query is nothing but converting
HQL Query to the underlying DB specific SQL Query
and sending that SQL Query to DB SW for execution.

→ Hibernate 3.x SW internally uses AST Query Translator
to convert HQL queries into DB SW equivalent SQL
queries.

⇒ `import`

```
public class HQLTest
```

```
{
```

```
    public static void main(String[] args) throws Exception.
```

```
{
```

```
    // create HB session object
```

```
    SessionFactory factory = new Configuration().configure
```

```
        .configure("hib.cfg.xml").buildSessionFactory();
```

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

```
    Query q1 = ses.createQuery("select eb from EmpBean as  
                                eb");
```

```
    List l = q1.list(); // execute HQL.
```

```
    // display records
```

```
    for(int i=0; i<l.size(); ++i)
```

```
    {  
        EmpBean eb = (EmpBean) l.get(i);
```



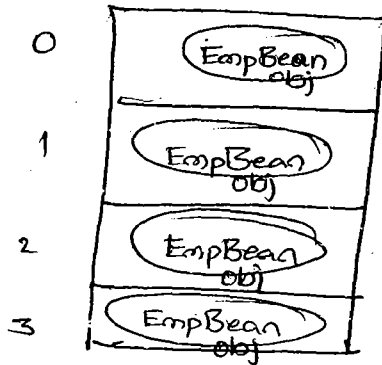
```

s.o.p ( e1.getNo() + " " + e1.getName() + " " + e1.getLname()
      + " " + e1.getEmail());
}
}

```

Object ~~represents~~ Memory representation:-

(java.util.List obj)



⇒ Executing \odot HQL select Query by using Iterate method.

Ex:-

```

Query q1 = ses.createQuery("select eb from EmpBean
                           as eb");

```

```

Iterator it = q1.iterate(); //executes HQL

```

```

while (it.hasNext())

```

```

{

```

```

EmpBean e1 = (EmpBean) it.next();

```

```

s.o.p ( e1.getNo() + " " + e1.getName() + " " +
        e1.getLname() + " " + e1.getEmail());

```

```

}

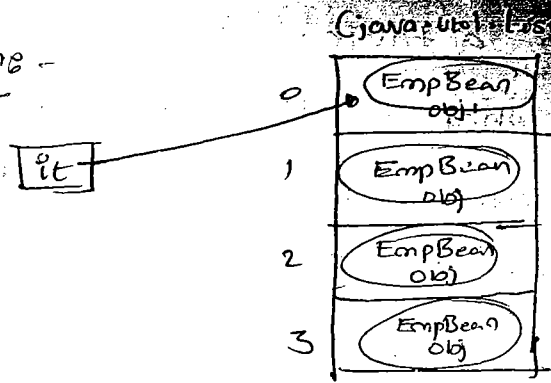
```

```

}

```

Memory diagram -



*** Q) what is the diff b/w executing select HQL query by using list() & by using iterate() ?

=> list() method generates result by selecting all records through the execution of single select SQL query -

(No Lazy Loading) that means if results objects are used (or) not used by the programmer (by calling ~~getObj~~ getXXX()), the pojo class object representing results will be initialized with data (records).

→ iterate() method, selects the records from DB table by executing multiple SQL select queries (almost one select query for record) and also generates first SQL select query only to retrieve column names.

iterate() performs lazy loading because it creates results related Hibernate and initializes results related HB pojo class object on demand.

* list() method is recommended to use

→ we can see the above diff b/w list() and iterate()

of a table.

Q1) ⇒ what is the different ~~is~~ b/w executing
SQL select query in JDBC & HQL select query
in HJB?

Ans:- JDBC code based select query execution
gives result set object, which is not serializable
object so we cannot send resultset object
over the network.

Hibernate based ^{select} HQL query execution
gives result in the form of collection Framework
List Data Structure. since List Data Structure object
is serializable object by default, we can send
the object over the network.

① Note: all the collection FW's are serializable
object by default.

② Note:- In order to get above said benefit in
HJB the programmer should make HJB POJO class
object implementing java.io.Serializable interface

HQL Query with conditions -

⇒

```
Query q1 = ses.createQuery("select eb from EmpBean  
as eb where eb.no >= 100 and eb  
eb.fname like '0%'");
```

```
List l = q1.list();
```

```
for (int i = 0; i < l.size(); ++i)
```

```
{
```

```
EmpBean e1 = (EmpBean) l.get(i);
```

```
SOP ( e1.getNo() + " " + e1.getName() + " " +
```

```
e1.getLname() + " " + e1.getEmail());
```

```
}
```

```
=
```

```
}
```

```
}
```

```
=
```

14/08/10

Comment :- Arrays are objects in java, so arrays can be added as element values of collection Framework.

→ when HQL select queries are selecting specific column values of a table then the result related List DS contains java.lang.Object class object arrays as element values.

ex:-

```
Query q1 = ses.createQuery("select eb.fname, eb.mail  
from EmpBean as eb where eb.no >= 100");
```

```
List l = q1.list();
```

```
//display results
```

```
for(int i=0; i<l.size(); ++i)
```

```
{
```

```
Object row[] = (Object[]) l.get(i);
```

```
for(int k=0; k<row.length; ++k) {
```

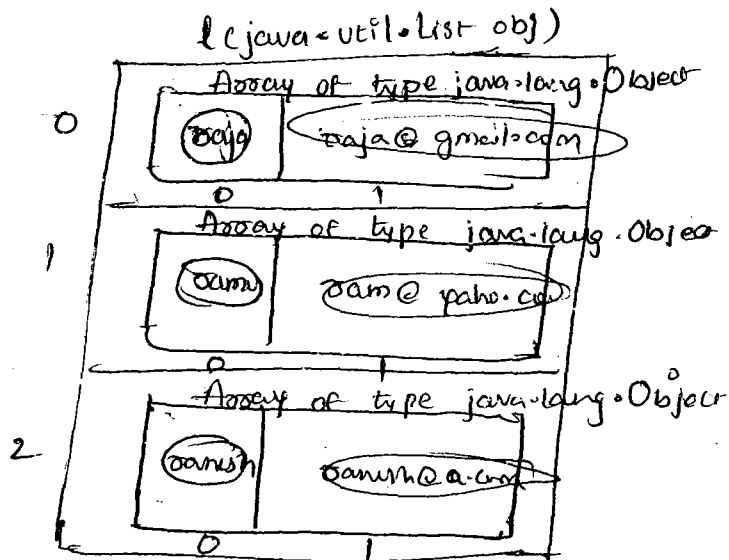
```
SOP ( row[k].toString() + " ");
```

```
} inner loop
```

```
SOP ( );
```

```
} out for loop
```

Memory Diagram :-



→ when Array type is Student class the elements in the Arrays are student ^{class} object
 when Array type is java.lang.Object class the elements in that are ~~object~~ java.lang.Object class objects

examples - existing SQL select Query that select specific column values by using 'iterate()'

```
Query q1 = ses.createQuery("select eb.no, eb.mail  

    from EmpBean as eb where (eb.no >= 100  

    or (eb.mail like '%.com') or  

    (eb.mail like '%gmail.com')");
```

```
Iterator it = q1.iterate(); // execute SQL Query
```

```
while (it.hasNext())
```

```
{  
    Object row[] = (Object[]) it.next();
```

```
    for (int k=0; k<row.length; ++k)
```

```
    {  
        s.o.p (row[k].toString() + " ");
```

```
    }
```

→ When `iterate()` is used to select specific column values of HQL select Query then no lazy loading will take place.

example:-

Executing single Aggregate function based HQL Select Query

Query `q1 = ses.createQuery("select count(*) from EmpBean eb");`

List `l = q1.list();`

S.O.P ("count of records in table are" + `l.toString()`);

Executing HQL Query having multiple Aggregate functions and utility functions

Query `q1 = ses.createQuery("select count(*), avg (eb.no), upper ("safe"), sum (eb.no), max (eb.no) from EmpBean eb");`

List `l = q1.list();`

S.O.P ("Results are" + `l.toString()`);

Object `res[] = (Object[]) l.get(0);`

S.O.P ("count is" + `res[0].toString()`);

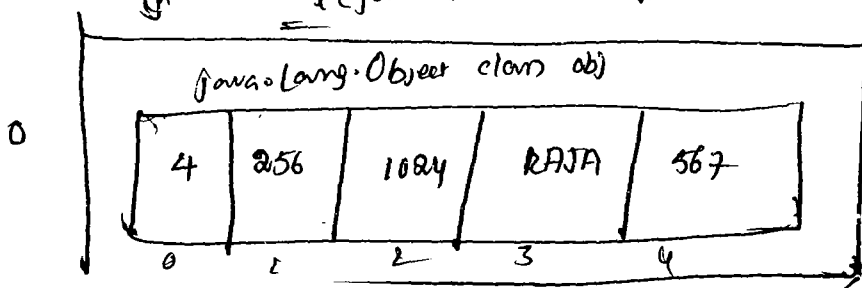
S.O.P ("avg is" + `res[1].toString()`);

S.O.P ("upper is" + `res[2].toString()`);

S.O.P ("sum is" + `res[3].toString()`);

S.O.P ("max is" + `res[4].toString()`);

④ `l (java.util.List obj)`



Example on subqueries in HQL environments -

```
Query q1 = ses.createQuery("select eb from EmpBean as eb  
where eb.no = (select max(ebi.no) from  
EmpBean as ebi)");
```

```
List l = q1.list();
```

```
for(int i=0; i<l.size(); ++i)
```

```
{
```

```
EmpBean e1 = (EmpBean) l.get(i);
```

```
System.out.println(" " + e1.getId() + " " + e1.getName() +  
" " + e1.getHail());
```

```
}
```

```
==
```

⇒ To make HQL query flexible by setting input values at query from outside Query and to set Query i/p values as javaside values without worrying about underlying DB sql syntax we can pass parameters to HQL queries.

Parameters passed to HQL Query never makes HQL Query as precompiled Query because HQL Query cannot directly go to DB sql.

16/08/10

→ In most of the situations, the #B s/w generated JDBC code internally uses precompiled SQL Queries with the support of prepared statement object to perform persistence operations on the table.

→ All HQL Queries related SQLQueries generated by #B s/w are precompiled Queries by default.

Parameters in HQL

- (1) positional parameters (?)
- (2) Named parameters (: <Name>) → Recommended to use

Example Code on positional parameters -

```
Query q1 = ses.createQuery ("select eb from EmpBean as e'b whose eb.no >= ? and eb.mail like ?");
```

// setting parameter values.

```
q1.setInteger (0, 30);
```

 → position starts from '0' in HQL

```
q1.setString (1, "x@gmail.com");
```

↓ ↓
parameter parameter value.
index

```
List l = q1.list();
```

```
for (int i=0; i<l.size(); ++i)
```

```
{  
    EmpBean eb = (EmpBean) l.get(i);
```

```
    sop (e1.getNo () + " " + .....);  
}
```

→ we can't pass HQL keywords, POJO classnames, POJO class number variable names as values of parameters.

HQL Queries, it is against of HQL Syntax.

Wrong HQL Query (as) invalid HQL Query.

Query q1 = ses.createQuery("select eb from ^{un-escaped token.} ? as eb where eb.no >= ? and eb.mail like ?");

q1.setString(0, "EmpBean");

q1.setInt(1, 30);

q1.setString(2, "x.gmail.com");

⇒ use parameters in HQL Query only to pass input values and condition values. [ie not table names]

Named Parameters -

Example code on Named Parameters :-

Query q1 = ses.createQuery("select eb from EmpBean as eb where eb.fname in(:P1, :P2) and eb.mail like :P3");

// setting values for parameters

q1.setString("P1", "saja");

q1.setString("P2", "savi");

q1.setString("P3", "x.s.com");

List l = q1.list();

for (--)

b

↓

=

*** → in JDBC programming, we can't pass Named parameters in SQL Query. In HQL Query both Named, positional parameters in a single Query.

But we must pass positional parameters before Named parameters.

Valid { Query q1 = ses.createQuery("select eb from EmpBean as eb where eb.fname in(?, ?) and eb.mail like ?P3");
q1.setString(0, "saja");
q1.setString(1, "savi");
q1.setString("P3", "x.s.com");

Invalid HQL Query:-

Invalid { Query q1 = ses.createQuery ("select eb from EmpBean as eb
where eb.fname 'in (:P1, :P2) and eb.mail like?");
q1.setString ("P1", "ajay");
q1.setString ("P2", "xx");
q1.setString (0, "x.y.com");

Executing Non-Select SQL Query (insert/update/delete):-

→ must be executed as transaction statement

→ HQL insert Query is not given to insert only one record at a time, but to insert multiple records by selecting them another table (To solve this problem use ses.save())

Example:- Executing HQL delete Query

Transaction txn = ses.beginTransaction();

Query q1 = ses.createQuery ("delete from EmpBean as eb
where eb.no = (select min (eb.no) from EmpBean as eb)");

int res = q1.executeUpdate();

s.op ("no of record affected : " + res);

txn.commit();

HQL Non-select * Query with Parameters:-

Transaction txn = ses.beginTransaction();

Query q1 = ses.createQuery ("update EmpBean as eb set
eb.mail = ? where eb.no >= :P1");

q1.setString (0, "x@gmail.com");

q1.setInteger ("P1", 1000);

int res = q1.executeUpdate();

s.op ("no of record updated : " + res);

txn.commit();

⇒ HQL Insert Query is not given in this form "insert into EmpBean as eb values (? , ? , ? , ?)" ⇒ Wrong Query

"insert into EmpBean eb values (p1, p2, p3, p4)" ⇒ Wrong

⇒ The supported form of HQL insert query is

** INSERT INTO <table1> SELECT <table2> FROM :-

ie using this syntax we can ~~insert~~ insert

⇒ we can't insert directly records to table, we can insert record only by selecting from another table.

Example:-

Source table <Employee>

③ same as first application.

destination table <Employee1>

EID → number PK

name → varchar2(20)

mail → varchar2(20).

EmpBean.java:-

same as first appln.

EmpBean1.java:-

int no, String name, String email

→ we can configure multiple HBM pers class in single mapping file by using <class> tag multiple times.

Employee.hbm.xml:-

<hibernate-mapping>

<class name="EmpBean" table="Employee">

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

=

100
=

4
≠

using

```
<class name="EmpBean" table="Employee1">
  <id name="eid" column="EID"/>
  <property name="name" column="NAME"/>
  <property name="mail" column="MAIL"/>
</class>
</hibernate-mapping>
```

Code in client Applet - 'HQLTest.java'

→ To select records from one table and to insert them in another table.

```
Transaction tx = ses.beginTransaction();
Query q1 = ses.createQuery("insert into EmpBean1 (no, name, mail) select eb.no, eb.name, eb.mail from EmpBean as eb where eb.no >= :p1");
q1.setInteger("P1", 300);
int res = q1.executeUpdate();
s.o.p("no of records affected : " + res);
```

For related information on HQL, see more examples on HQL see chapters 13 & 14 of PDB file.

17/8/10

Pagination -

when query selects huge no of records instead of displaying them in a single screen (or) single page, it is recommended to display them in multiple screens (or) in multiple pages through 'pagination' concept

- hibernate persistence logic provides Environment i.e. required for pagination.
- All the 3 bulk operations techniques support pagination.
- Even though select query selecting huge amount of records in order to select specific range of records from those records use setFirstResult(), setMaxResult() methods are shown below.
- we can also use this method for 'pagination'

Ex:-

```
Query q1 = sess.createQuery("select eb from EmpBean as eb");
```

```
// select 3 records from 300 records (0 based index)
```

```
q1.setFirstResult(2);
```

```
q1.setMaxResults(3);
```

```
List l = q1.list();
```

```
for(int i=0; i<l.size(); ++i)
```

```
{
```

```
EmpBean e1 = (EmpBean) l.get(i);
```

```
S.o.p (e1.getName() + " " + e1.getPhone() +
```

```
" " + e1.getMail());
```

```
}
```

=> Hibernate internally uses the pseudo column rownum while generating SQL Query for Oracle DB SW based on above code.

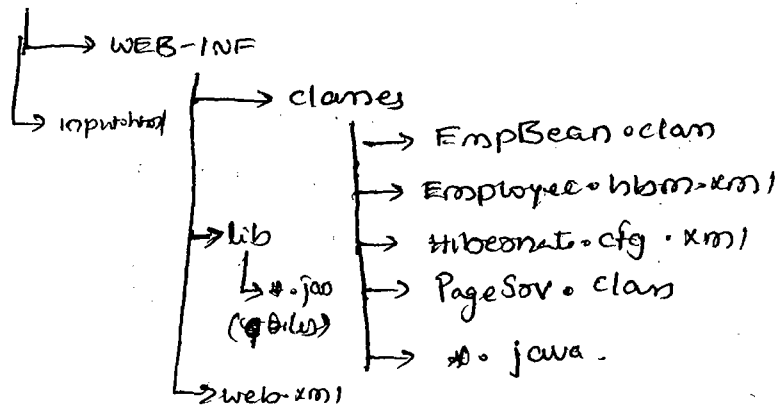
-> 'rownum' is a invisible pseudo column in DB tables at Oracle SW holding rowno's as column values.

-> To perform pagination by using the persistence logic in web-applications, take fixed no for maxResult and change FirstResult value based on the request no (or) page no., To make these operations specific to each client use session attribute to manage first result value at each page.

Deployment Directory structure of web-app that performs pagination by using the persistence logic :-

~~web-usb~~

page HBA App



java files in WEB-INF / lib folder

② -> lib jars files

① -> other jars files

java files in class path -> SERVLET-API.jar
-> hibernate.jar

EmpBean.java -

← same as 1st appln →

Employee.htm.xml -

← same as 1st appln →

hibernate.cfg.xml -

← same as 1st appln →

input.html -

<center> Generate Report
↓ </center>
url pattern as PageServlet

web.xml -

configure PageServlet by having /page01 as
url pattern

PageServlet.java -

```
import javax.servlet.*;  
import javax.servlet.http.*;  
import java.io.*;  
import org.hibernate.*;  
import org.hibernate.cfg.*;  
import java.util.*;
```

```
public class PageServlet extends HttpServlet
```

```
{
```

```
    Session ses = null;
```

```
    int initVal;
```

```
    int totalRecords;
```

```
    public void init()
```

```
    {  
        try
```

```
        {  
            ses = new Configuration().configure().build  
                SessionFactory().openSession();
```

```
catch (Exception e)
```

```
{  
    e.printStackTrace();
```

```
}
```

```
} //logic
```

```
public void service (HttpServletRequest req, HttpServletResponse res)  
    throws SE, IOE
```

```
{
```

```
    // logic to get count of records in db table for every request
```

```
    Query q2 = ses.createQuery ("select eb from EmpBean as eb");
```

```
    List l2 = q2.list();
```

```
    totRecords = l2.size();
```

```
    PrintWriter pw = res.getWriter();
```

```
    res.setContentType ("text/html");
```

```
    // create / locates session for client
```

```
    HttpSession se = req.getSession();
```

```
    if (session.getAttribute ("counter") == null)
```

```
    {
```

```
        // logic of first request
```

```
        int val = 0;
```

```
        ses.setAttribute ("counter", new Integer (initVal));
```

```
    }
```

```
    else
```

```
    {
```

```
        // logic for other than 1st record.
```

```
        Integer i1 = (Integer) session.getAttribute ("counter");
```

```
        initVal = i1.intValue () + 2;
```

```
        session.setAttribute ("counter", new Integer (initVal));
```

```
    }
```

```
    // logic to display records in form of html table
```

```
    by applying pagination,
```

```
    Query q1 = ses.createQuery ("select eb from EmpBean as eb");
```



```
q1 = setFirstResult (initVal);
```

```
q1 = setMaxResults(2);
```

```
List l = q1 = list();
```

```
pw = println (" <table border = '1' > <tr > <th > No </th >  
<th > FIRSTNAME </th > <th > LASTNAME </th >  
<th > EMAIL </th > </tr >");
```

```
for (int i = 0; i < l.size(); ++i)
```

```
{
```

```
EmpBean e = (EmpBean) l.get(i);
```

```
pw = println (" <tr > <td > " + e.getNo() + " </td > &
```

```
<td > " + e.getFname() + " </td >
```

```
<td > " + e.getLname() + " </td >
```

```
<td > " + e.getMail() + " </td > </tr >");
```

```
}
```

```
pw = println (" </table >");
```

|| logic to display home or next hyperlinks in the page

```
if ((initVal + 2) < (totalRecords))
```

```
{  
pw = println (" <a href = '#page01' > next </a >");
```

```
}
```

```
else
```

```
{
```

```
pw = println (" <a href = 'input.html' > home </a >");
```

```
session.invalidate();
```

```
}
```

```
pw.close();
```

```
} // service
```

```
public void destroy()
```

```
{
```

```
try
```

```
{  
ses.close();
```

```
catch (Exception e)
```

```
{
```

} // destroy

} // clear

18-08-10

Native SQL :-

→ Native SQL queries are underlying DB SW specific SQL queries. These queries based persistence logic of HIB is DB dependent persistence logic. If programmer feels certain operation is complex (or) not possible with HQL then it is recommended to use "NativeSQL" queries.

→ Native SQL Query will be written DB Table names and column names.

→ There are 2 types of Native SQL Query.

① Entity Queries :-

these queries return all the column values of a table.

② Scalar Queries :-

these queries return specific column values or non-column values as results.

Native SQL Query results must be mapped with Hibernate POJO classes or HIB datatypes.

Native SQL Queries allow both named, positional parameters.

→ SQLQuery object represents one NativeSQL Query.

This is the object of a class, that implements org.hibernate.SQLQuery interface.

This interface is sub interface of org.hibernate.Query interface.

→ Native SQL Query programming environment is very useful for the programmer to call PL/SQL procedures and functions.

ab DB slw form HIS programming language logic -

⇒ Example on Entity Native SQL Query (or) select Query
that select all column values :-

~~Test Case~~

SQLTestClient.java :-

```
String sql = "select {e.*} from Employee e";
```

↑ table name
↑ alias name.

```
SQLQuery q1 = ses.createSQLQuery(sql);
```

```
// map entity query result with HIS pojo class
```

```
q1.addEntity("e", EmpBean.class);
```

```
// execute NativeSQL query
```

```
List l = q1.list();
```

```
// display result
```

```
for(int i=0; i<l.size(); i++)
```

```
{
```

```
EmpBean e1 = (EmpBean) l.get(i);
```

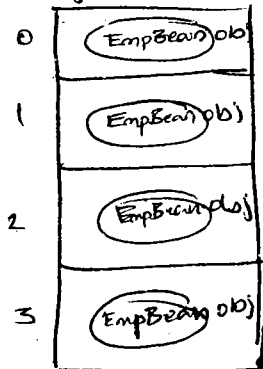
```
System.out.println(" " + e1.getFname() + " "
```

```
e1.getLname() + " " + e1.getHw());
```

```
}
```

```
}
```

java.util.List obj



 => we can't call iterate() on SQL Query object to execute entity Query (oo) select Query of NativeSQL.
 this indicates lazy loading (oo) lazy initialization of
 this pojo class objects is not possible with NativeSQL Query

Another form of above example:-

```

=>
String sql = "select * from Employee";
SQLQuery q1 = ses.createSQLQuery(sql);
//map results with this pojo class
q1.addEntity(EmpBean.class);
//execute Native SQL Query.
List l = q1.list();
//display Result.
for (int i=0; i<l.size(); i++)
{
  EmpBean e1 = (EmpBean) l.get(i);
  s.opc =
  {
  =
  }
}

```

Entity - NativeSQL must be mapped with this pojo class

we can pass both Named and positional parameters in NativeSQL Queries.

in order to pass both types of parameters in a single NativeSQL Query, the positional parameters must be placed before Named Parameters.

```

String sql = "select * from Employee where eid >= ? and eid <= ? P1";
SQLQuery q1 = ses.createSQLQuery(sql);

```

```

// set parameter values.
q1 = session.createQuery("0, 300");
q2 = session.createQuery("p1", 800);

// map result with HB pojo class
q1.addEntity(EmpBean.class);

// execute NativeSQL Query
List l = q1.list();

// display results
for (int i = 0; i < l.size(); i++)
{
    =
}
=

```

⇒ Like JDBC datatypes in the form of `Types.INTEGER`, `Types.VARCHAR` and etc, HB also giving its own built-in datatypes as constants of `org.hibernate.Hibernate` class,

ex- `Hibernate.INTEGER`, `Hibernate.FLOAT`, `Hibernate.LONG`, `Hibernate.STRING` etc..

These `hibernate` datatypes are useful while catching scalar query results, scalar NativeSQL Query results in `hibernate` environment.

public static final member variables of java classes interface are called constants.

Example on Scalar Query -

```

// Native SQL scalar query.
String sql = "select max (eid) as mval from Employee";
SQLQuery q1 = ses.createQuery(sql);

// mapping scalar query result with HB datatype

```

// executes Native SQL Query

List l = q1.list();

s.o.p ("max value of eid col is " + l.toString());

~~==~~

=

}

⇒ making scalar Query returning more than one result.

// Native SQL scalar Query

String sql = "select max(eid) as mval, count(*) as cnt from Employee";

SQLQuery q1 = ses.createSQLQuery(sql);

// mapping scalar query result with Hib data types

q1.addScalar("mval", Hibernate.INTEGER);

q1.addScalar("cnt", Hibernate.INTEGER);

// executes Native SQL Query.

List l = q1.list();

// display results

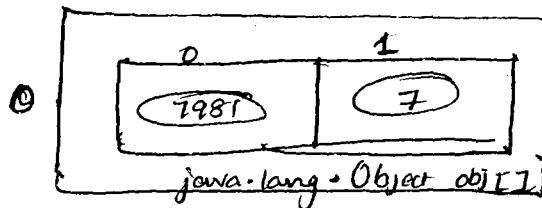
Object res[] = (Object[]) l.get(0);

s.o.p ("max val in EID col is " + res[0].toString());

s.o.p ("count of records" + res[1].toString());

=

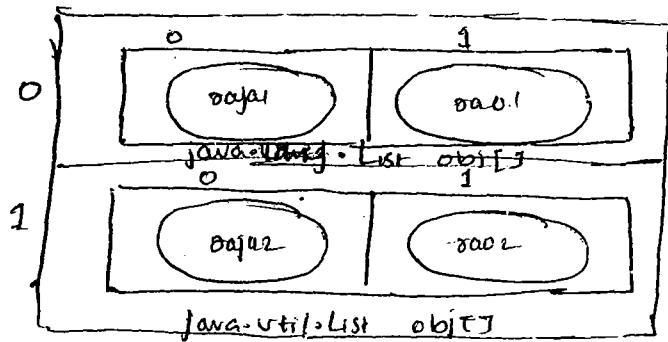
l (java.util.List obj)



Example on Native SQL Scala Query that is selecting specific

columns values of table :-

(Cjava.util.List obj)



String sql = "select FIRSTNAME, LASTNAME from employee
where email like 'P%'";

SQLQuery q1 = Session.createSQLQuery(sql);

// set parameters values

q1.setString("P1", "x@gmail.com");

// map scala query results with HIB data type

q1.addScalar("firstname", Hibernate.STRING);

q1.addScalar("lastname", Hibernate.STRING);

// execute Scala Query.

List l = q1.list();

for(int i=0; i<l.size(); ++i)

{

Object row[] = (Object[]) l.get(i);

for(int k=0; k<row.length; ++k)

{

s.o.p(" ");

s.o.p(row[k].toString() + " ");

}

}

19/08/17

⇒ All non-select NativeSQL Query must be executed as transactional statement?

⇒ By using NativeSQL Query we can directly insert single record into DB table, by directly passing values.

```
Transaction tx = ses.beginTransaction();
```

```
String
```

```
String sql = ses.createSQLQuery(sql);
```

```
q1.setIntegers("P");
```

```
String sql = "insert into Employee values (:P1, :P2, :P3, :P4)";
```

```
SQLQuery q1 = ses.createSQLQuery(sql);
```

```
q1.setIntegers("P1", 324);
```

```
q1.setString("P2", "oaja");
```

```
q1.setString("P3", "sam");
```

```
q1.setString("P4", "oaja@gmail.com");
```

```
int res = q1.executeUpdate();
```

```
s.op("no. of records affected : " + res);
```

```
tx.commit();
```

⇒ we can execute DDL Queries as NativeSQL Queries,

```
Transaction tx = ses.beginTransaction();
```

```
String sql = "Drop table xyz";
```

```
SQLQuery q1 = ses.createSQLQuery(sql);
```

```
q1.executeUpdate();
```

```
s.op("table dropped");
```

```
tx.commit();
```


→ Native SQL Query, that is placed in HBM Mapping file, having logical name is called Named Native SQL Query, these queries are required.

① To make Native SQL Query visible across the multiple session object of hibernate application.

(All these session object must be created by using same SessionFactory object).

② To make Native SQL Queries or Flexible Queries by putting them from outside persistence logic.

~~Ex:-~~

→ The standard principle in the industry is don't hardcode any values in java application that are possible to change in future.

ex:- ① To make JDBC persistence logic as flexible logic and to get the effect of developing DB independent persistence logic then use properties file support having SQL Query

⇒ Named Native SQL Queries will be written in ~~hbm~~ HBM mapping file but instructions of executing those queries will be given in. HB Persistence logic of client application.

ex- on Named Native SQL Query (Entity Query)

step ①:- propose NamedQuery in HBM Mapping file as shown below.

```
<hibernate-mapping>  
<class name="pi-EmpBean" table="Employee">
```

<property name="mail" column="EMAIL" />

</class>

<sql-query name="my-test">

<return class="EmpBean" />

select * from Employee where eid=?
and eid <= ?P1

</sql-query>

</hibernate-mapping>

<lt => '<' symbol if we give like this in XML
it treats subtag in so keep "<lt"

step 2:- write following code in client application
to execute NamedSQLQuery.

// represents named sql query

Query q1 = ses.getNamedQuery("my-test");

// setting parameter values

q1.setInteger(0, 100);

q1.setInteger("P1", 500);

// executes named native sql query

List l = q.list();

// display records

for(int i=0; i < l.size(); ++i)

{
EmpBean e1 = (EmpBean) l.get(i);

s.o.p(e1.getNo() + " ");

}

=> Native SQL Queries are written directly in client apps along with HB persistence logic they become specific to one HB session object using which ~~SQL~~ SQL Query object is created

By Making ^{Native} SQL Query as Named Queries we make them visible to multiple session object in client application. (In ~~each~~ single client app)

```

// repository named sql query
Query q = ses.getNamedQuery ("my-test");

//

ses.close();
Session ses1 = factory.openSession();
q
Query q2 = ses1.getNamedQuery ("my-test");
//

```

Example on executing ScalaO-NamedNativeSQL Query:-

Step 1:- Prepare NativeSQL query in mapping file.

```

<sql-query name="my-test">
  <return class="pl.EmpBean"/>
  select * from Employee where eid >= ? and
  &lt; ; = ?P,
</sql-query>

```

```

<sql-query name="mytest1">
  <return-scalao column="FIRSTNAME" type="string"/>
  <return-scalao column="EMAIL" type="string"/>

```

select FIRSTNAME, EMAIL FROM employee where
email like ?

<sql-query>

<hibernate-mapping>

step 1 :- write coding in client app to execute NamedSQL

~~SQL~~ NativeSQLQuery.

```
Query q1 = ses.getNamedQuery("my-test1");
```

```
// setting parameter values
```

```
q1.setString(0, "y.gmail.com");
```

```
// execute named native sql query
```

```
List l = q1.list();
```

```
for (int i = 0; i < l.size(); ++i)
```

```
{
```

```
Object row[] = (Object[]) l.get(i);
```

```
for (int k = 0; k < row.length; ++k)
```

```
{
```

```
s.o.p (row[k].toString() + " ");
```

```
} // inner for
```

```
s.o.p ();
```

```
} // outer for
```

```
<
```

Note :- it is always recommended to write NativeSQL

Queries as Named Queries.

NativeSQL Queries, Named NativeSQL Queries, also support
pagination, use same methods as JDBC programming.

⇒ we can keep NamedNativeSQLQuery in a single
HBM mapping file.

⇒ Executing Non-Select NativeSQLQuery as NamedQuery -

Step 1: - prepared Named in HBM Mapping file.

```
<sql-query name="my-test2">
  delete from Employee where eid = (select max(eid)
  from Employee)
</sql-query>
```

Step 2: - execute NamedQuery in client appln.

```
Transaction tx = ses.beginTransaction();
```

```
Query q1 = ses.getNamedQuery("my-test2");
```

```
int res = q1.executeUpdate();
```

```
ses.flush("do as expected records" + res);
```

```
tx.commit();
```

```
ses.close();
```

}

=

20/8/10

→ we can make HQL Query as NamedQueries by passing

them in HBM mapping file using <query>

→ By making Queries as NamedHQLQueries, we can make them visible for multiple session object that are created by using SessionFactory object.

example appln that deals with Named HQL Queries -

① prepare NamedHQL Query in HBM mapping file.

```
<query name="my-test2">
  delete from EmpBean as eb where eb.no = 20
```

step 1:- execute Named ^HSQL Query in the client application.

```
Transaction tx = ses.beginTransaction();
Query q1 = ses.getNamedQuery("my_test2");
//setting parameter value
q1.setString("P1", 67);
int res = q1.executeUpdate();
System.out.println("no of records that are affected is " + res);
tx.commit();
}
```

⇒ example on select Named HQL Query

step 1:- place HQL Query in mapping file.

```
<query name = "my_test3" >
    select eb from EmpBean as eb where eb.mail like :P1
</query >
```

step 2:- execute this query from client application

```
Query q1 = ses.getNamedQuery("my_test3");
q1.setString("P1", "y.gmail.com");
List l = q1.list();
for (int i = 0; i < l.size(); i++)
{
    EmpBean eb = (EmpBean) l.get(i);
    System.out.println(eb.getNo() + " " + ...);
}
```

⇒ In order to centralise certain persistence logic or Business logic for multiple modules or a project or as for multiple projects of a company then place that logic in DB SW as PL/SQL procedure (or) function

→ HB persistence logic can call PL/SQL procedure (or) function of DB SW from client application only when

→ These rules of developing PL/SQL procedure or function for HB will change based on the DB SW we use.

→ A 'CURSOR' in PL/SQL programming of Oracle is a Datatype whose variable can store zero or more selected records from DB table.

→ common rules to prepare PL/SQL procedure or function for the Oracle.

① pagination is not possible, as the results generated by queries of the Oracle specific PL/SQL procedure. i.e. we can't use `setFirstResult()`, `setMaxResults()` methods in this environment.

② we must follow the following syntax, to call ~~procedure~~ PL/SQL procedure.

```
{ call procedurename (<parameters>) }
```

③ we must follow, following syntax to call PL/SQL function.

```
{ ? = call functionname (<parameters>) }
```

Note:- PL/SQL procedure does not return result value, where as PL/SQL function returns a value.

Rules specific to ORACLE :-

① PL/SQL procedure 1st parameter must be an out parameter returning resultset. [cursor having records]

② PL/SQL function must return a resultset [cursor having records]

Note :-

SYS_REFCURSOR is cursor type Datatype given by PL/SQL programming of Oracle. having the ability to store selected records [one or more] like resultset object as JDBC programming.

→ For rules specific to Sybase, Microsoft SQL DB SW see
Chapter 16 of PDF file.

NOTE! - HB applications use NativeSQL programming to call
PL/SQL procedures or functions.

⇒ Example Appn to call PL/SQL procedure of Oracle from
HB persistence logic -

Step 1! - prepare PL/SQL procedure in Oracle, in the angle it is
required for HB.

[see above Oracle rules]

define cursor
Data type in Oracle

```
> create or replace procedure get_empdetails (mycursor out sys_refcursor,  
        ename in employee)  
begin  
    open mycursor for  
        select * from Employee where first_name like ename;  
end;
```

→ the NativeSQL query generated records will be stored as

step 2! - execute the above PL/SQL procedure in Oracle DB SW

> /

step 3! - call the above PL/SQL procedure from HB mapping file
as Named NativeSQL Query in employee.hbm.xml

```
<sql-query name = "my-test" callable = "true" >  
    <return class = "EmpBean" /> ↳ mandatory  
    { call get_empdetails ( ?, ?PI ) } ↳ HB protocol  
    </sql-query >
```

* → this '?' marks placeholders or parameters of the above
PL/SQL procedure (cursor) it must always be '?' symbol
you can't place named parameter here.

** (P1) :- P1 represents 'in' parameter of above

PL/SQL procedure, it can be either named or positional parameter.

when it is taken as positional parameter, it index becomes zero.

Step 1:- write following code in client application to ^{call &} execute

PL/SQL procedure (code in client app)

```
Query q = ses.getNamedQuery("my_test");
```

```
1) setting parameter values
```

```
q.setParameter("P1", "8%");
```

```
List l = q.list(); // execute update PL/SQL procedure.
```

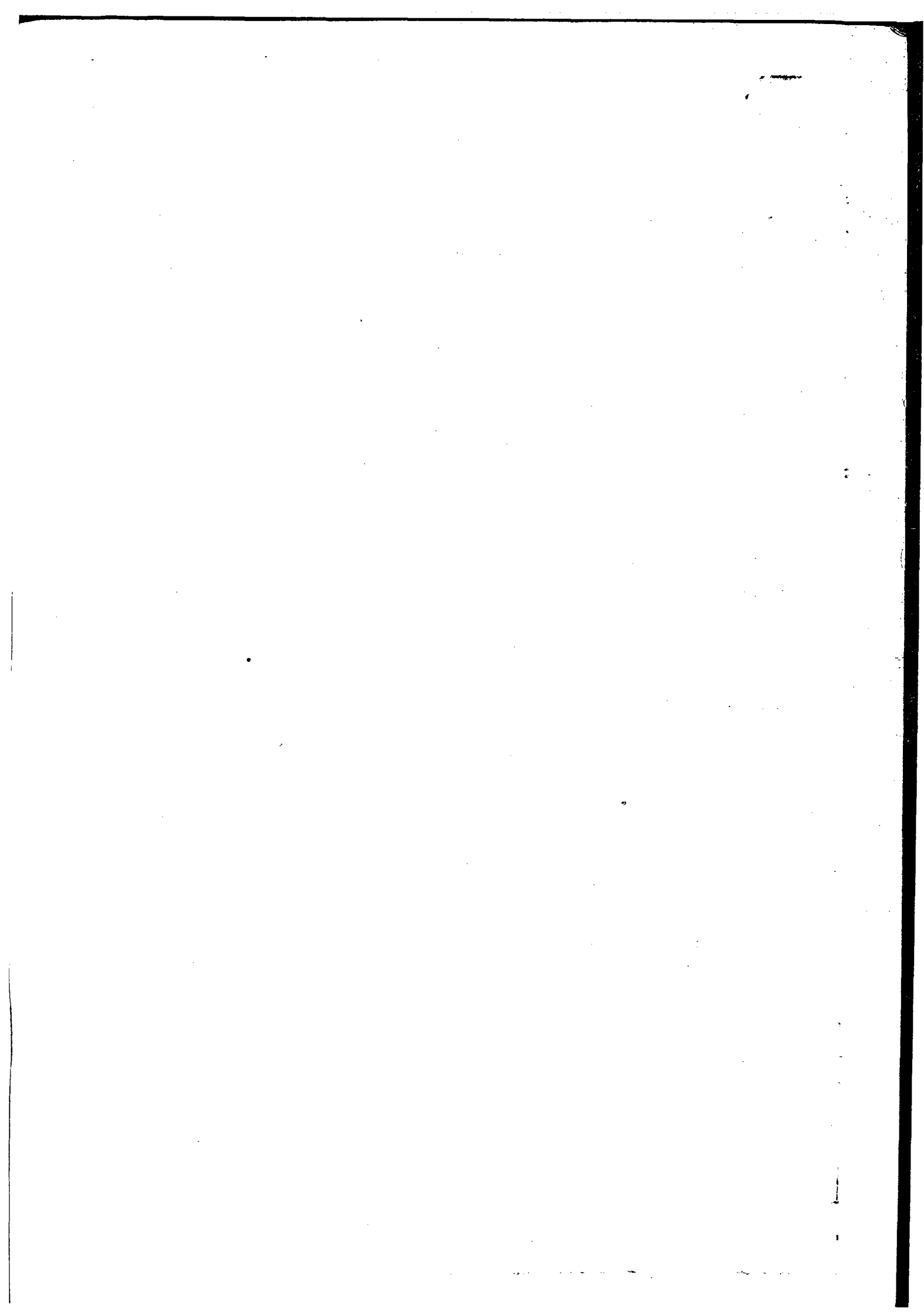
```
for (int i = 0; i < l.size(); ++i) {
```

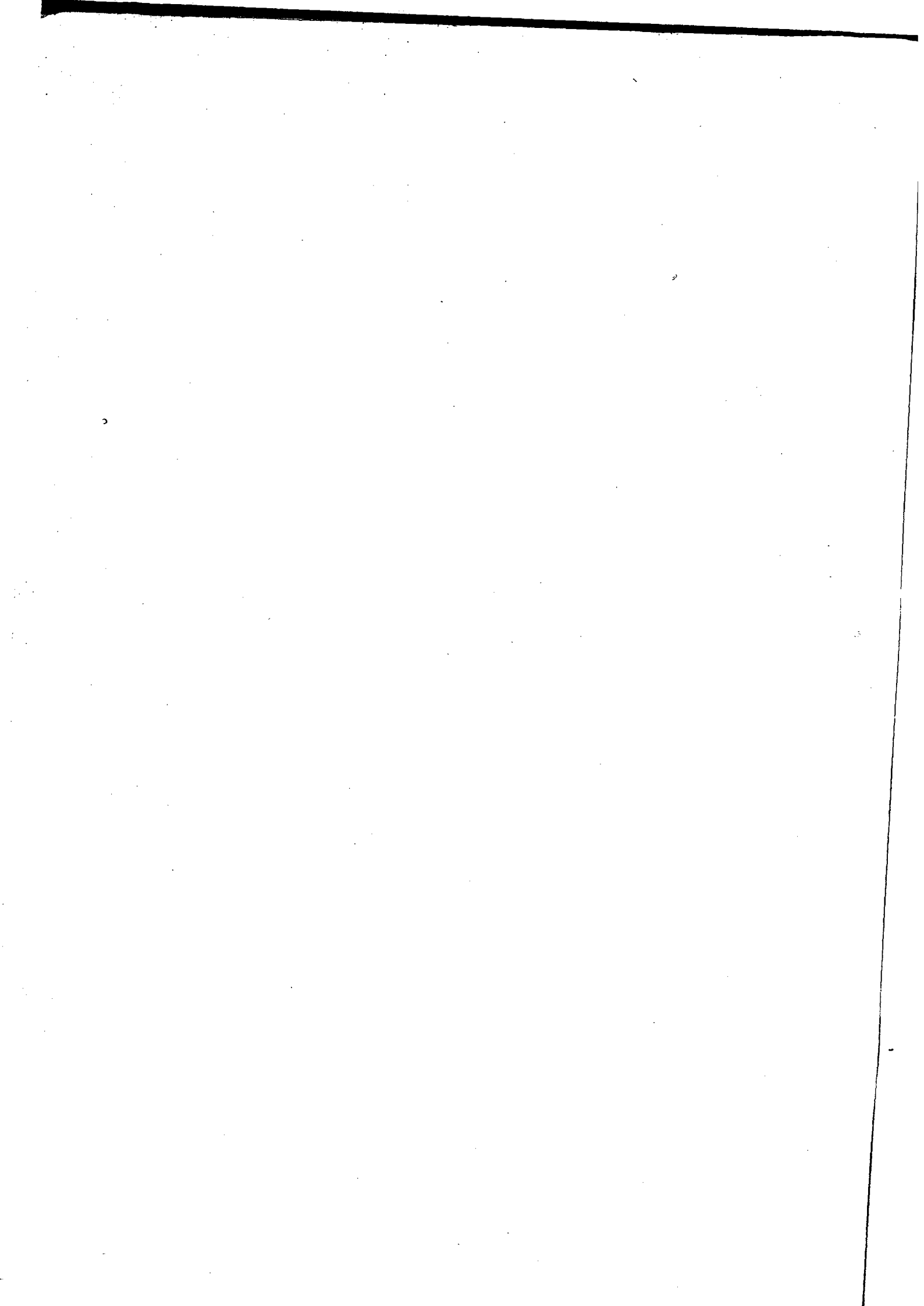
```
    {
```

```
    }
```

```
}
```

5) Develop remaining services of the application like ~~1st~~ application





21/8/10

* procedure to call PL/SQL function at Oracle DB side
by using hibernate persistence logic :-

step 1 :- prepare PL/SQL function in Oracle DB side
returning Result Set (cursor)

> create or replace function emp_function (no in number)
return SYS_REFCURSOR

as

~~my_cursor~~ my_cursor SYS_REFCURSOR;

BEGIN

open my_cursor for

select * from Employee where eid >= no;

return my_cursor;

END;

select * from uses_procedures;

step 2 :- execute this PL/SQL function in Oracle DB side.

step 3 :- call the above PL/SQL function from HB mapping
file by using standard syntax.

in Employee.hbm.xml :-

SQL

<sql-query name = "my_test" callable = "true">

<return class = "EmpBean" /> ↳ mandatory

can't be
taken as
Named parameter

emp_function = call emp_function (:PL)

</sql-query>

</hibernate-mapping>

step 4 :- write following persistence logic in client app
to call and execute that PL/SQL procedure.

```
Query q1 = ses.getNamedQuery ("my_test");
```

```
// setting parameter value
```

```
q1.setParameter ("p1", 350);
```

```
List l = q1.list (); // execute PL/SQL function in  
DB slw.
```

```
for (int i = 0; i < l.size (); i++) {
```

```
EmpBean eb = (EmpBean) l.get (i);
```

```
=
```

```
}
```

```
=
```

** We can't perform pagination / paging on the results generated by PL/SQL procedure or function called from HB persistence logic. (limitation of HB).

** Limitations of HB slw -

① since HB is not a distributed technology, the HB persistence logic is not distributed persistence logic.

So this persistence logic can't be used from remote client.

② calling PL/SQL procedure (or) function from HB persistence logic is quite complex moreover these PL/SQL function or procedure must be written in the angle they are required for HB.

③ pagination is not possible on the results not generated by PL/SQL procedures or functions.

④ HB can't be used to interact with Non-conventional DB slws like text file, MS Excel and etc

(JDBC can do this)

⑤ HIB can't interact with few conventional DB SW like MS-Access

⑥ HIB persistence logic use, negligible performance degradation compare to JDBC.

⇒ → Calling PL/SQL procedure (or) function for HIB persistence logic when it is not written based on HIB rules is not directly possible but it is possible indirectly by injecting JDBC code in HIB persistence logic.

Example app:-

step 0:- develop and execute normal PL/SQL function in oracle.

```
create or replace function my_function(x in number)
return number as y number;
begin
    y := x * x;
return y;
end;
```

step 1:- write following code in client application to call that PL/SQL function.

```
Transaction tx = ses.beginTransaction();
```

```
// jdbc logic
```

```
Connection con = ses.getConnection(); // gives jdbc con obj
```

```
CallableStatement cs = con.prepareCall("{? = call my_function?}");
```

```
cs.setInt(2, 10); // this JDBC so starts from 1
```

```
cs.registerOutParameter(1, Types.INTEGER);
```


DB 25
cs.execute() & if execute - PL/SQL function of DB SW

```
s.op ("select * from cs.getint(1));
```

```
tx.commit();
```

```
cs.close();
```

=

=

NOTE:- to execute, the above persistence logic there is no need of HB POJO class and HBMapping file, Just the HBConfiguration file is enough.

The above coding scenario is very much needed to call already available and running PL/SQL procedures from HB application even though they are not developed in the angle they are required for hibernate.

⇒ procedure to call a PL/SQL function at oracle, performing Non-select operation on the table:-

step 1:- prepare & execute ~~prepare~~ PL/SQL function as shown below.

↳ CREATE OR REPLACE FUNCTION my_function1(no in number)

RETURN NUMBER AS

~~BEGIN~~

delete from Employee where eid=no;

return SQL%ROWCOUNT;

END;



Built-in cursor in oracle representing the no of record that are effected by a Non-select Query execution done in PL/SQL programming.

→ to call above PL/SQL function in client app in as shown below.

```
Non-ORMapping
Transaction tx = ses.beginTransaction();
// jdbc logic
Connection con = ses.getConnection(); // gives jdbc con obj
CallableStatement cs = con.prepareCall("call my_function(?)");
cs.setInt(1, 567);
cs.registerOutParameter(1, Types.INTEGER);
cs.execute(); // executes PL/SQL function DB side
s.o.p("no. of records effected : " + cs.getInt(1));
tx.commit();
```

Note: - SQL Queries based PL/SQL programming is not possible

23/08/10

CRITERIA API :-

- This API is given to develop HB persistence logic plug-in in java statements and without using SQL/HQL queries
- criteria API means working with classes & interfaces of org.hibernate.criteria package classes interface to develop PoL
- we can use criteria API only select operations based persistence logic development.
- we ~~can~~ have to use criteria API, only to select out column values of a table. ie criteria API is not design to select specific column values of a table.
- while developing criteria API based persistence logic POJO classes, ~~Member~~ memberVariable names will be utilised
- criteria API based persistence logic is DataBase independent persistence logic.

Example on criteria API to select all records of a

table :-

gives
select * from Employee
object.

```
Criteria ct = ses.createCriteria(EmpBean.class);
```

```
List l = ct.list(); // execute criteria api based Pol
```

```
for (int i=0; i<l.size(); i++)
```

{

```
EmpBean e1 = (EmpBean) l.get(i);
```

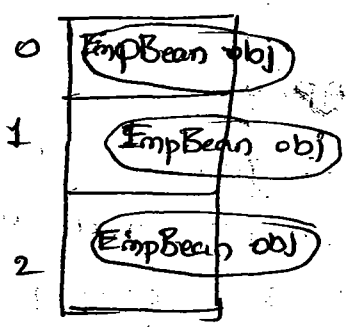
```
S.o.p (e1.getId() + " " + e1.getName() + " ");
```

}

}

Memory Diagram :-

l (java.util.List obj)



→ Iteration based select Query execution is not possible with criteria API, because iterate() is not available in org.hibernate.Criteria interface

⇒ To execute logic with conditions - (greater than equal)

```
Criteria ct = ses.createCriteria(EmpBean.class);
```

```
Criteria cond1 = Restrictions.ge("no", new Integer(100));
```

~~ct =~~

```
ct = ct.add(cond1);
```

List l = (cf. listC); // gives select * from employee

```
for (int i=0; i<l.size(); i++)  
{  
    // ...  
    // ...  
    // ...  
}
```

All methods of Restriction, Expression class are given to frame condition while developing Criteria API based persistence logic.

All these methods directly (or) indirectly return

Criteria object, these two classes are available

org.hibernate.Criteria package, Expression is the subclass of Restrictions class.

(Q) What is the diff b/w criteria object and criterion object?

ans) Criteria object represents the total single persistence operation related persistence logic of Criteria API. The conditions in this persistence logic will be represented by Criterion object.

→ we add criterion objects to Criteria object to add conditions to the basic select operation.

→ Criteria object means, it is the object of class that implements org.hibernate.Criteria interface.

Criterion object means, it is the object of class that implements org.hibernate.Criteria.Criterion interface.

⇒ criteria object is mutable object, i.e.,
 After creating criteria object, if you add new data
 or if ~~you can~~ modify existing data of that object
 changes will be reflected dynamically.
 in the same object

⇒ when multiple conditions are added to criteria
 object these conditions will be executed having
~~and~~ 'and' class by default.

Criteria

Example with multiple conditions: - (greater than equal / like)

```
Criteria ct = ses.createCriteria(EmpBean.class);
Criterion cond1 = Restrictions.ge("no", new Integer(100));
Criterion cond2 = Restrictions.like("mail", "x.gmail.com");
ct.add(cond1);
ct.add(cond2);
List l = ct.list(); // gives select * from Employee
                       where eid >= 100
                       and email like 'x.gmail.com'
```

using in condition: - (in, or)

```
Criteria ct = ses.createCriteria(EmpBean.class);
Criterion cond1 = Restrictions.lt("no", new Integer(100));
Criterion cond2 = Restrictions.in("fname", new String[]
{ "oaja", "oavi" });
cond1
Criterion orcond = Restrictions.or(cond1, cond2);
```

```
ct.add(oocond1);
```

```
List l = ct.list(); // select * from employee where  
eid < 100 or (firstname in ("saja", "savi"))
```

```
foo(- - -);
```

```
{  
=
```

```
}
```

```
=>
```

⇒ placing multiple conditions with OR clauses :-

```
Criteria ct = ses.createCriteria(EmpBean.class);
```

```
Criterion cond1 = Restrictions.eq("id", new Integer(100));
```

```
Criterion cond2 = Restrictions.in("fname", new String[] {  
"saja", "savi"});
```

```
Criterion oocond1 = Restrictions.or(cond1, cond2);
```

```
Criterion cond3 = Restrictions.ne("mail", new String("saja@gmail.com"));
```

```
Criterion oocond2 = Restrictions.or(oocond1, cond3);
```

```
ct.add(oocond2);
```

```
List l = ct.list();
```

```
foo(- - -);
```

```
}
```

```
// select * from Employee where (eid < 100 or fname  
( 'saja', 'savi' )) or email <> 'saja@gmail.com'
```

```
=
```

like)

));

om");

);

om");

});

Applying And, OR clauses or based on condition in

Query execution

→

```
Criteria ct = ses.createCriteria(EmpBean.class);
```

```
Criterion cond1 = Restrictions.lt("no", new Integer(100));
```

```
Criterion cond2 = Restrictions.in("fname", new String[]  
    {"saja", "savi"});
```

```
Criterion andCond = Restrictions.and(cond1, cond2);
```

~~Criteria~~

```
Criterion cond3 = Restrictions.ne("mail", new String  
    {"saja@gmail.com"});
```

```
Criterion orCond = Restrictions.or(Cond1, Cond3);
```

```
ct.add(orCond);
```

```
List l = ct.list();
```

for
for
for

⇒

we can pass conditions in criteria API based persistence logic in the form of SQL Query statements by using expansion = sql()

```
Criteria ct = ses.createCriteria(EmpBean.class);
```

```
Criterion cond1 = Expansion.sql("email like '%.gmail.com'");
```

```
Criterion cond2 = Expansion.sql("first name in ('saja', 'savi')");
```

```
ct.add(cond1);
```

```
ct.add(cond2);
```

```
List l = ct.list();
```

for
for
for

=

HQL is the most recommended technique to develop persistence logic in HB environment.

24/8/10

→ HQL, Native SQL, Criteria API based programming supports paging operation but the NativeSQL programming that deals with PL/SQL procedures and functions does not support paging operation.

Filters:

→ HB Filters are Hibernate 3.x feature that allows the programmer to specify predefined criteria / condition for the persistence logic of Hibernate programming.

24/8/10

→ Filters are present where condition clause related, condition in HB mapping file and they are visible for multiple session objects of Hibernate application.

~~They~~ These filters can be enabled or disabled dynamically at runtime on each session object level.

→ A Hibernate Filter is a Global Named parameterised condition that can be enabled or disabled per specific HB session, when enabled all queries executed using that session object will be executed with condition of the Filter.

Example Application to HB Filter on HB persistence logic:

① → define filter in HB mapping file specifying its parameter names and types.
in

in employee-hbm.xml file:

```
<hibernate-mapping>
```

```
<class name =
```

```
=====
```

```
</class>
```

↳ logical name of filter.

```
<filter-def name = "myfilter">
```

Parameters
names & types

```
<filter-param name = "myid1" type = "int" />
```

```
<filter-param name = "myid2" type = "int" />
```

```
</filter-def>
```

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

step 2:-

link filter with specific hibernate POJO class and write condition.

```
<hibernate-mapping>
```

```
<class name = "pl EmpBean" table = "Employee">
```

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

```
<property
```

```
=
```

the filter that is taken above.

```
<filter name = "myfilter" condition = "
```

```
condition = "eid >= :myid1 and eid <= :myid2" />
```

```
</class>
```

```
<filter-def name = "myfilter">
```

```
≡
```

```
</filter-def>
```

```
</h-m>
```

```
=
```

condition separates
by filter

step 2 :- enable filter on HIB session object and execute
HIB persistence logic.

code in client application :-

~~#end~~

```
ses = sf.openSession();
```

```
// enable filter on hibernate ses obj
```

```
org.hibernate.Filter f1 = ses.enableFilter("myfilter");
```

```
// set parameter values
```

```
f1.setParameter("myid1", new Integer(100));
```

```
f1.setParameter("myid2", new Integer(400));
```

```
// HQL Query
```

```
Query q1 = ses.createQuery("from EmpBean");
```

```
List l = q1.list(); // execute HQL query with condition
```

```
for(int i=0; i<l.size(); i++)
```

```
{
```

```
EmpBean e = (EmpBean) l.get(i);
```

```
s.op("e.getId() + ' ' + e.getName() + .....");
```

```
}
```

```
// disabling filter on ses object
```

```
ses.disableFilter("myfilter");
```

```
Query q2 = ses.createQuery("from EmpBean");
```

```
List l1 = q1.list(); // execute HQL query with out condition
```

```
for(int i=0; i<l1.size(); i++)
```

```
{
```

```
EmpBean e = (EmpBean) l1.get(i);
```

```
s.op(.....);
```

```
}
```

```
==
```

- Hibernate Filters makes no conditions abt HIB Filter logic as flexible condition by making them coming to applns from outside the persistence logic (comes from HIB mapping file)

step 3:- Develop remaining resources abt the application like most application.

=> For complete source code above application refer page appln (3) given in page no: 82-85

=> we can enable or disable filters on ^{only} ~~all~~ ~~all~~ types abt HIB persistence logic environment HQL, ~~SQL~~ Criteria API

=> it is not applicable for "Native SQL"
(we cannot enable filters on Native SQL programming related persistence logic development).

Q) => when and where to use interfaces and abstract classes in your project development?

Ans - For every project project specification will be design before starting coding abt the project - this specification contains rules and guidelines to develop the project.

every project specification contains its own project API designed by PL, this API represents methods declared in the interfaces, abstract classes as rules and represents concrete method definitions abt abstract classes and concrete classes as guidelines.

while implementing project based on project specific rules will be implemented. Interfaces methods and guidelines will be followed (concrete method definitions)

→ If PL wants to give everything as rule in project specification then he takes interfaces and declares method depending rules.

⊕ If PL wants to give rules and guidelines in project specification then he takes abstract classes and declares ~~method~~ abstract methods depending rules and defines concrete methods depending guidelines

→ In coding level of project development programmer never takes user defined interfaces and abstract classes.

But he takes user defined interfaces only ~~the~~ when they are required as resources of certain technology based application development services.

→ Spring interfaces in Spring appln.

→ Business interface in RMI appln, EJB component.

⊕ =

⇒ SW specification contains rules and guidelines to develop new softwares, every SW specification contains built-in API, the interfaces of this API dependant rules of SW specification having method declarations. The concrete methods of this API, available in concrete, abstract classes dependant guidelines of SW specification

Ex → ⊕ JDBC specification contains rules and guideline to dev. JDBC drivers

- ② Servlet JSP specifications contain to develop Servlet Containers / web containers

25-8-10

INHERITANCE MAPPING :-

Hibernate persistence classes are POJO class, so they can participate in inheritance to give the advantage of reusability and extensibility.

When HIB persistence classes are there in inheritance, i.e. DB table's Related to these persistence class we also try to maintain data having relationship.

To make HIB understanding this inheritance b/w POJO classes, we need to configure them in HIB mapping file based inheritance mapping concepts.

→ Inheritance b/w two Servlet components is possible, but inheritance b/w two EJB components is not possible.

→ HIB allows the following ORM mapping operations in mapping file.

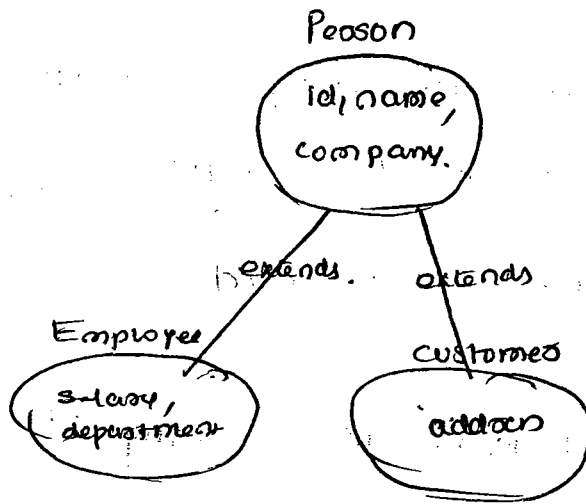
- ① Basic OR Mapping
- ② Inheritance Mapping
- ③ Collection Mapping
- ④ Association mapping ~~and~~ etc...
- ⑤ Component mapping and etc....

→ HB slw use ③ approaches of mapping inherit mapping to get Reusability, extensibility advantage of inheritance in different ways.

- they are
- ① Table per class hierarchy
 - ② Table per subclass
 - ③ Table per concrete class.

① Table per class hierarchy :- (54 page no)

All the classes of inheritance hierarchy use single DB table by having discrimination column.



all these classes inheritance hierarchy will use same DB +

in-persons

(PK) id	name	company	salary	department	address	discrimn column ↑ person
101	rajee	HCL	-	-	-	person
104	navi	TCS	10,000	1001	-	emp p
108	ram	Wipro	15,000	-	hyd	cust 1
210	saksh	HCL	-	-	Sec	cust pa

→ in table per class hierarchy, it is recommended to

take DB table having discriminator column because

this column maintains logical values representing

records in the table are inserted using which POJO

class or inheritance hierarchy.

see => person-type column of above DB table

→ the logical values of this discriminator values are

very useful to select specific POJO class related

records separately for DB table even though DB

table contains other POJO-classes related records.

→ In table per class hierarchy inheritance mapping

<class> will be used to configure super class

and <subclass> will be used to configure

sub class.

This <subclass> is <class> tag.

→ in this approach discriminator column is table

doesn't contain any property in

the POJO class but its configuration will

be done in its mapping file.

→ For Example Application on table per class

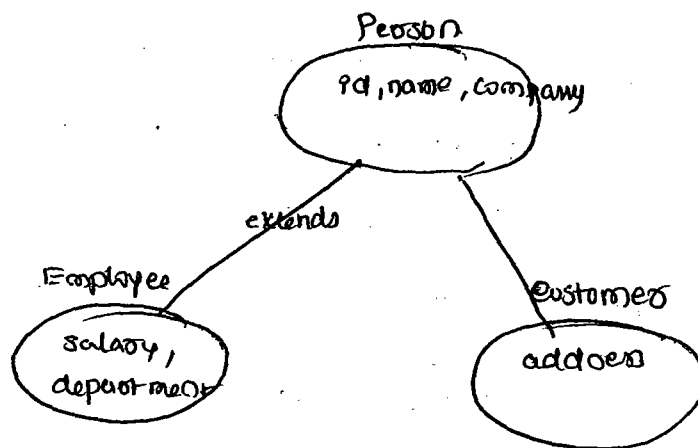
hierarchy model of inheritance mapping

see the code given in page 57-58

→ while working with table per class hierarchy inheritance model the columns related to child pojo class properties in DB table can't be applied with 'not null' constraint and discriminator column can't be applied with unique key or 'primary key' constraint.

25/8/10
63page no ⇒ The advantage of any model of Inheritance mapping, is nothing but the ability of access child pojo class objects selected records through parent pojo class object.

② Table per subclass model of Inheritance mapping -



→ In this model, every pojo class of inheritance will have its own database table. The table ~~table~~ of child pojo classes will have a/s with one-to-one parent pojo class table.

<u>in person a</u> (parent table)			
id (pk) (N)	name (V2)	company (V2')	V2 → Varchar 2 (20) N → Number Number
101	xxx	TCL	
101*	yyy	PARS	
101**	zzz	WIPRO	

Salary department

Salary (N) 6000
department (N) 1001

(child table)

person-id (FK with rel ab in-person)
(N) 121*

Number
N - ~~1001~~

in - customer 2 (child table)

address (N) Hyd
person-id (N) 131*

FK with 121 color in person

→ In this model of inheritance mapping all columns of all DB tables can be applied with not null constraint and there is no necessity of discriminator column.

One - one relationship b/w two tables is nothing but one record of parent table will always represent one record of child table.

→ In table per subclass model of inheritance mapping ~~class~~ <class> will be used to configure parent POJO class and <joined-subclass> used to configure child - POJO classes.

<joined-subclass> is the sub tag of <class> ,

→ In this model of inheritance, child POJO class object data will be splitted and will be inserted parent, child tables having relationships.

This relationship comes because of Foreign Column of child table, so this Foreign key column should be configured along with child POJO classes in HB Mapping File.

Table per subclass inheritance mapping model is most regularly used inheritance mapping model in company level programming because.

① it recognises inheritance b/w POJO classes and gives reusability of properties.

② it allows to design table having relationship satisfying database designing principles.

③ allows to apply, Not Null constraint on all the columns of all the tables

For Example application on Table per Subclass inheritance mapping see application ⑦ in page 59-63

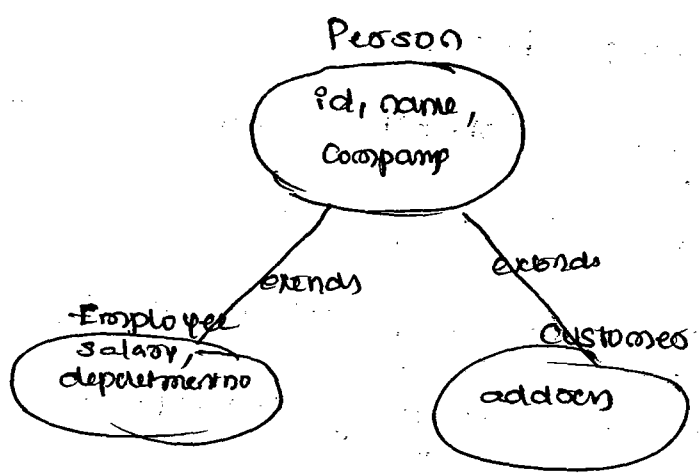
→ while working with multiple table, advanced ~~table~~ OR mapping concepts, it is recommended not to use reverse engineering of MyEclipse IDE.

use ~~the~~ MyEclipse IDE for these operations but develop resources manually

③ Table per concrete class

in this inheritance mapping, even though POJO classes are there in the inheritance, their tables will not participate in relationships and every POJO class in inheritance hierarchy will use one separate table without having relationship with other table.

→ These POJO classes will be configured in HBM mapping file as individual, independent, concrete classes without showing inheritance and without showing reusability or properties configurations.



In - persons 3

id (n) (pk)	name (vc2)	company (vc2)
101	saja	xyz

In - Employees 4

id (n) (pk)	name (vc2)	company (vc2)	salary (n)	deptno (n)
102	sauri	abc	8,000	1001

In - Customers 3

id (n) (pk)	name (vc2)	company (vc2)	address (vc2)
103	samush	mno	hyd

⇒ The above table designing is very poor designing because multiple tables of DBs are having same col so Table per concrete class model is inheritance mapping is not industry standards.

⇒ In this model all columns of all tables can be applied with 'Not-Null' constraint.

⇒ For Example application on table per concrete class App

Page no 63-68

27/08/2010

* Component mapping :-

The property of HIB POJO class that represents single column of DB table is called simple property.

→ The property in hibernate POJO class whose type is class name (user defined) and represents multiple columns of a DB table is called 'Component Property'.

→ A component property is a contained object in hibernate POJO class object representing specific multiple column values of a record.

→ component property of hibernate POJO class represent partial data of record whereas its hibernate POJO class object represents total record data including this partial data.

→ public class Person

{
int pid;

String pname;

JobType pjob; // component property

= getxxxx(x);

setxxxx(x);
}

```

public class JobType
{
    String job;
    double salary;
    int department;
    = getxxx()
    setxxx()
}

```

person-fab (DB table) :-

pid	pname	job	salary	department
(n)	(v(2))	(v(2))	(n)	(n)
101	raja	SE	18000	1001

- pid (property) → pid col
- pname (" ") → pname col
- pjob (" ") → job, salary, department cols

→ To configure "simple property"

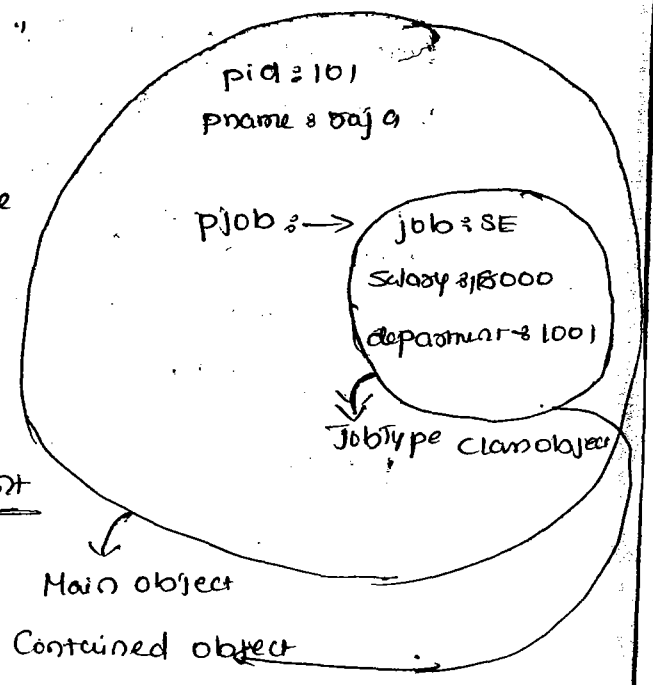
in this pojo class use

<property> tag in mapping file

similarly <component> tag in mapping file.

→ For example application on component

mapping better apply :- 5 ab
material in the pages :- 51-54



Main object

Contained object

→ If you are looking to maintain partial values of see in the form of separate java class object take compon property in the part class representing selected multiple values as a table.

Q) How to find out no. of objects created for java class

ans:- Take static member variable in java class and increment that member variable in constructor. So this static member variable keeps track of the number of objects created for that class.

ex:- TestApp.java

```
class Test
```

```
{
```

```
public static void main int counter;
```

```
Test ()
```

```
{
```

```
System.out.println("constructor of Test class");
```

```
counter++;
```

```
}
```

```
}
```

```
public class TestApp
```

```
{
```

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

```
{
```

```
Test t1 = new Test();
```

```
Test t2 = new Test();
```

```
Test t3 = new Test();
```

```
System.out.println("NO of objects are created = " + Test.counter);
```

```
}
```

```
}
```

00
1001

object

28-08-20

Association Mapping :-

* DB designing team designs the tables of project according to "Normalization Rules".

There are total six Rules (or) norms. The Second normalization rule says design tables having integrity constraints that means tables should be designed having r/s like one-to-one, one-to-many, many-to-many and etc.

When tables are there in Association (or) R/s we can access one table data based on another table data because records of one table represents the records of another table.

→ DB team take the support of primary key, Foreign key constraint to design the table having relationships

→ The relationship b/w student table and rank table is one-to-one because one student ~~can~~ contains only one rank will be given or assigned to only one student.

→ The relationship b/w user, phone numbers table is ~~one-to-one~~ one-to-many because one user can have multiple phone numbers.

→ The r/s b/w emp, dept tables is many to one because many employees can belong to one department.

→ The r/s b/w programmers and project in small scale organization is many to many because in one project multiple programmers are there, one programmer can work for multiple projects, multiple projects can be

assigned to one programmer (or) multiple programmers.

→ To define any JIS except many-to-many two tables are enough (parent, child) but to define many-many and three tables are required (table 1, table 2, relationship table)

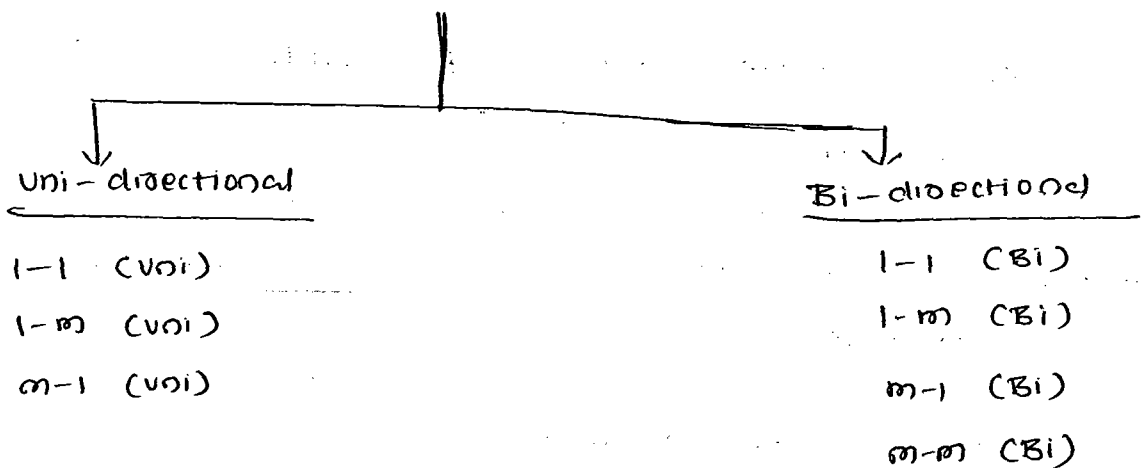
→ when two database tables are in relationship their

"JIS pojo classes" must be designed and configured supporting that relationship. This work is called

"Association mapping" in JIS mapping file.

→ when JIS pojo classes are designed supporting JIS then the objects of these classes actually participate in object level relationship (or) association.

JIS object level relationships



→ By using parent pojo class object data we are able to access the associated child pojo class "object/objects" data and if reverse is not possible then it is called "unidirectional" association. If reverse is also possible then it is called "bidirectional" association.

Ex! - If student pojo class object (parent) pointing one bank pojo class object (child) having one-to-one association it programmer is able to get associated bank object data through student object class and reverse is not possible then it is called "unidirectional" association.

If reverse is also possible then it is called

"bidirectional association".

→ The JIB programmer should always design JIB POJO classes based on the E-R diagrams given by db team.

→ ER Diagrams (Entity - Relationship diagrams)

To perform association mapping in Hibernate mapping file we can work with following tags <one-to-one>, <one-to-many>, <many-to-many>, and <many-to-one> tags.

inverse = "true" attribute of these tags allows to enable bi-directional association/relationship.

inverse = "false" allows to enable unidirectional association.

⇒ Example JIB POJO classes for unidirectional one-to-one association :-

```
public class Student
```

```
{
```

```
    int sno;
```

```
    String name, address;
```

```
    Rank r;
```

```
    // represents one object of Rank
```

```
    // POJO class
```

```
    // 4 no of setter/getter methods.
```

```
}
```

```
public class Rank
```

```
{
```

```
    int sno;
```

```
    String subject;
```

```
    String exam;
```

```
}
```

setter & getters

→ Example: Two POJO classes: two bidirectional one-to-one association -

```
public class Student
```

```
{
```

```
    int sno;
```

```
    String name;
```

```
    String address;
```

```
    Rank r; → Represents one object of Rank pojo class
```

```
    → getters & setters
```

```
}
```

```
public class Rank
```

```
{
```

```
    int rno;
```

```
    int sno;
```

```
    String subject;
```

```
    String exam;
```

```
    Student st; → Represents one object of Student pojo class
```

```
    → getters & setters
```

```
}
```

→ Example: Two POJO classes: two unidirectional one-to-many association -

```
public class User
```

```
{
```

```
    int uid; → Represents foreign key column of child table
```

```
    String username;
```

```
    List phones; (oo) Set phones;
```

allows duplicates ←

↓
Represents one (oo) more (oo) zero phone number pojo objects

```
    get xxx & set xxx
```

```
}
```

```
public class PhoneNumbers {
```

```
{
```

```
    long pb;
```

```
    String type;
```

```
    int uid;
```

```
    --
```

```
    --
```

```
    --
```

```
}
```

⇒ Example +18 post classes for Bi-Directional one-to-many Association :-

```
public class User
```

```
{
```

```
    int uid;
```

```
    String username;
```

```
    Set phones;
```

```
    ==
```

```
}
```

```
public class PhoneNumbers
```

```
{
```

```
    long pb;
```

```
    String type;
```

```
    int uid;
```

```
    User u;
```

```
    --
```

```
    --
```

```
    --
```

```
}
```

→ Example HIB POJO classes for UNIDIRECTIONAL MANY-TO
Association :-

```
public class Emp
```

```
{
```

```
int eno;
```

```
String ename;
```

```
float sal;
```

```
int deptno; //deptno is Foreign key column of
```

```
Department dept; → deptno is one object of Department  
POJO class
```

setters & getters

```
}
```

```
public class Department
```

```
{
```

```
int deptno;
```

```
String name;
```

```
String loc;
```

setters & getters

```
}
```

* In the above discussion "dept" properties belong into multiple Emp class objects whereas single.

(m) Same Department class object, to get the

object of many-to-one association.

Example HIB POJO classes for BIDIRECTIONAL MANY-TO
one association :-

```
public class Emp
```

```
{
```

```
int eno;
```

```
String ename;
```

```
float sal;
```

```
int deptno;
```

```
Department dept;
```

```
}
```

-many

```
public class Department
```

```
{  
    int deptno;  
    String name;  
    String loc;  
    Set e;  
    =  
}
```

→ Example HRB POJO classes too many-to-many
association :-

```
public class Project
```

```
{  
    int proj id;  
    String projname;  
    Set programmes;  
    =  
}
```

↳ represents zero (or) more objects
of programmes class.

30/8/10

In Association mapping, parent and child POJO
classes that are representing parent child DB
table respectively will not participate in any
kind of JLS inheritance. But there is a possibility
of property of parent POJO class representing
one or more objects of child POJO class and
vice versa.

Example application on one-to-many undirected Association mapping based on user and phone numbers

Information :-

DB tables :-

Parent table :-

```
create table USER_TABLE (USER_ID number(10)
primary key, FIRST_NAME varchar2(20));
```

Child table :-

```
create table PHONE_NUMBERS (NUMBER_TYPE
varchar2(20), phone number(10),
UNID number references USER_TABLE (USER_ID))
```

Foreign key column allowing duplicates

→ values to this foreign key UNID should not be inserted

or updated directly because these values comes automatically based on the values inserted in PK column of parent table (user-id).

Application Resources :-

→ hibernate.cfg.xml

→ user.java

→ phoneNumbers.java

} Hibernate POJO classes

→ user.hbm.xml

→ phoneNumbers.hbm.xml

} HBM mapping files

→ TestClient.java - client app ①

→ SelectTest.java - client app ②

hibernate.cfg.xml :-

Note :- configure details for Oracle DB SW connection having two mapping files configuration

```
<hibernate-configuration>
```

```
  <session-factory>
```

```
    --  
    --
```

```
  <!-- mapping files -->
```

```
  <mapping resource = "user.hbm.xml" />
```

```
  <mapping resource = "PhoneNumbers.hbm.xml" />
```

```
</session-factory>
```

```
</hibernate-configuration>
```

user.java (parent pojo class)

```
import java.util.*;
```

```
public class User
```

```
{
```

```
  private long userid;
```

```
  private String firstName;
```

```
  private Set phones; → To hold zero (or) more  
                        objects of phone number pojo class
```

```
  //getters & setters
```

```
  =
```

```
  public Set getPhones ()
```

```
  {
```

```
    return phones;
```

```
  }
```

```
  public void setPhones (Set phones)
```

```
  {
```

```
    this.phones = phones;
```

```
  }
```

```
}
```

```
=
```

PhoneNumbers.java :- (child POJO class)

```
public class PhoneNumbers
{
    private String numberType;
    private long phone;
    private long id; → represents Foreign key Column 0

    // setters & getters
}
=
```

User.hbm.xml :-

```
<hibernate-mapping>
<class name="User" table="USER_TABLE" >
<id name="userId" column="USER_ID"/>
<property name="firstName" column="FIRST_NAME" />
<set name="phones" table="PHONE_NUMBERS"
    cascade="all" />
<key column="UNID"/>
    ↳ FK Column is child table.
<one-to-many class="PhoneNumbers" />
</set>
</class>
</hibernate-mapping>
=
```

Note :- If this POJO class property type is collection
F/W data structures like List, Set, Map and etc of java.
util package. Then we need to configure those
properties in mapping file through "Collection Mapping"
for this we use <set> tag, <map> tag, <list> tag
<bag> tag and etc tags

Note:- since phone's property should hold and represent zero (or) more child pojo class objects (phone number class objects) the child table name (phone_numbers) child pojo class name (phone numbers) are specified in mapping file while configuring "phones" property.

→ Key attribute of any tag (or) ~~class~~ <key> tag always points to foreign key column name indicating that this is column that responsible for R/S b/w tables.

** cascade = "all" indicates any operation like insert, update, delete performed on the parent pojo class object will be reflected in child pojo class object if necessary.

PhoneNumbers.hbm.xml:-

```

<hibernate-mapping>
  <class name="PhoneNumbers" table="PHONE_NUMBERS">
    <id name="phone" column="PHONE"/>
    <property name="numberType" column="NUMBER_TYPE"/>
    <property name="id" column="ONID" insert="false"
      update="false"/>
  </class>
</hibernate-mapping>

```

* since the mapped column of "id" property is the FK column UNID and to disable direct updates and insertion in that column as manual operation insert = "false", update = "false" values kept. The values in FK column should come dynamically based on the values inserted on the referenced PK column other tables.

TestClient.java :

```

import org.hibernate.cfg.*;
import java.util.*;

public class TestClient
{
    public static void main (String args[])
    {
        try
        {
            Session ses = ----- ;
            Transaction tx = ses.beginTransaction();

            User u1 = new User();
            u1.setUserid (101);
            u1.setFirstName ("oaja");

            PhoneNumber ph1 = new PhoneNumber();
            ph1.setPhoneType ("residential");
            ph1.setPhone (65538968);
            PhoneNumber ph2 = new PhoneNumber();
            ph2.setPhoneType ("office");
            ph2.setPhone (65538958);

            Set s = new HashSet();
            s.add (ph1);
            s.add (ph2);
        }
    }
}

```

//Setting phoneNumbers class object to phones property

~~to~~ user obj

u1.setPhones(cs);

//to insert record

ses.save(u1);

tx.commit();

ses.close();

//try

catch (Exception e)

{

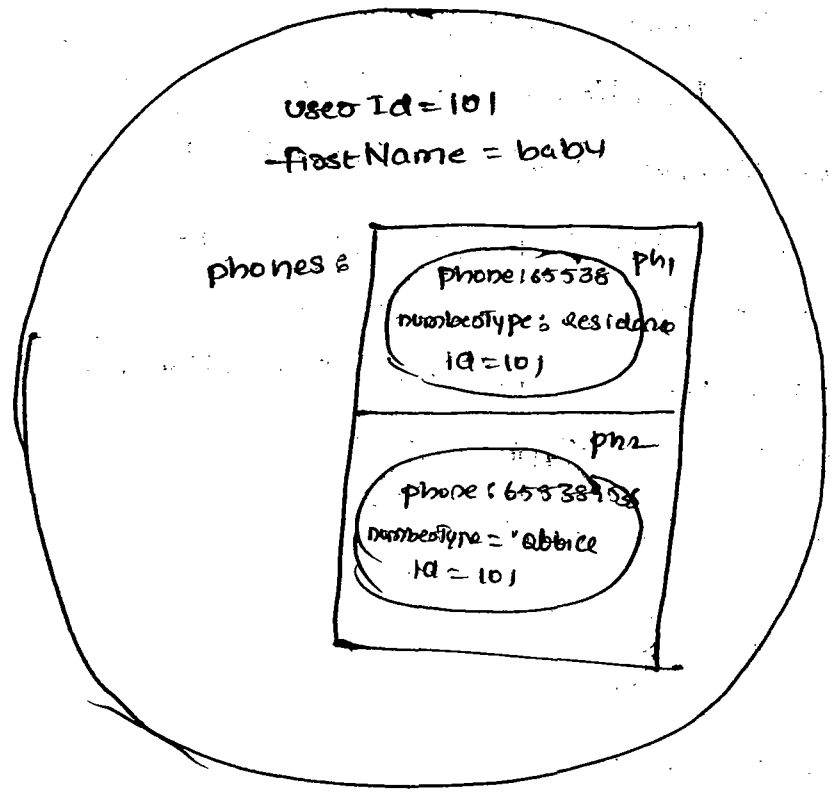
}

}

}

Memory Diagrams -

User obj (u1)



→ When the above application executed one record will be inserted in USER_TABLE and two records will be inserted in PHONE_NUMBERS table having one-to-many association as values in foreign key column UNID will be set dynamically.

31/05/2010

* SelectTest.java (gives parent table records and associated child table records): -

```

public class SelectTest
{
    public static void main (String args [])
    {
        try
        {
            Session ses = ... ;

            Query q1 = ses.createQuery ("from user");
            // gives parent table records and associated child
            table records

            List l = q1.list ();

            for (int i = 0; i < l.size (); i++)
            {
                User u1 = (User) l.get (i);

                S.o.p ("parent is " + u1.getUserId () + " " + u1.getFullName ());

                Set s = u1.getPhones ();

                Iterator it = s.iterator ();

                while (it.hasNext ())
                {
                    PhoneNumber ph = (PhoneNumber) it.next ();

                    S.o.p ("|t|t|t child is " + ph.getPhone () + " " +
                        ph.getNumberType () + " " +
                        ph.getId ());
                }
            }
        }
    }
}

```

Query q2 = ses.createQuery ("from phoneNumbers");

// gives only child records

l = q2.list();

for (int i=0; i < l.size(); i++)

{

PhoneNumbers ph = (PhoneNumbers) l.get(i);

S.o.p (ph.getNumberType() + " " + ph.getPhone() + " "

+ ph.getId());

}

ses.close();

} // try

catch (Exception e)

{

=

}

} // main

Note:- since the above application is having one

to many unidirectional association, we can get access to

child pojo class objects from parent pojo class object

but reverse is not possible. To make reverse operation

possible go for bidirectional one to many association.

* DeleteTest.java :- (to delete records)

public class DeleteTest

{

public static void main (String args[])

{

try

Session ses = ...;

Transaction txn = ses.beginTransaction();

Query q1 = ses.createQuery ("from users"); // delete parent
table records and associated records in child table.

List l = q1.list();

for(int i=0; i<l.size(); ++i)

{

User u = (User) l.get(i);

ses.delete(u);

}

tx.commit();

}

=

Developing one-to-one Bidirectional association based
web-applications using User, Phone Number details :-

DB tables! - same as above application

Note! - the designing of tables (parent & child) remains

same for unidirectional (or) bidirectional association
but changes will come in POJO class and mapping file

Resources of the application :-

→ hibernate.cfg.xml

→ User.java

→ PhoneNumbers.java

→ user.hbm.xml

→ PhoneNumbers.hbm.xml

→ TestClient.java

→ SelectTest.java

→ DeleteTest.java

hibernate.cfg.xml! -

→ same as above application ←

User.java! -

→ same as above application ←

PhoneNumber.java

```
public class PhoneNumber
{
    private String numberType;
    private long phone;
    private long id;
    User parent;
    // write getXxx & setXxx
    public void setParent (User parent)
    {
        this.parent = parent;
    }
    public User getParent ()
    {
        return parent;
    }
}
```

Users-hbm.xml

same as previous applo.

PhoneNumbers-hbm.xml

```
<hibernate-mapping>
```

```
<class name="PhoneNumber" table="PHONE_NUMBERS">
```

```
<id name="phone" column="PHONE"/>
```

```
<property name="numberType" column="NUMBER_TYPE"/>
```

```
<property name="id" column="UNID" insert="false"
```

```
update="false"/>
```

property is child pojo class to represent one object
↑ of parent pojo class.

```
<many-to-one name="parent" class="User"
```

```
column="UNID" cascade="all"/>
```

↳ parent
pojo class
name.

Foreign key column in
child table having ability to
perform both this ops

```
</class>
```

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

operations performed on child
pojo class will be reflected in
the associated parent pojo
class object is necessary

TestClient.java :-

Deber TestClient.java source code of page no :- 79880

Select Test.java :-

01/09/10

Deber 10 page no :- 81 & 82

⇒ DeleteTest.java :-

```
public class DeleteTest
{
    public static void main (String args[]) throws Exception
    {
        Session ses = — ;
        // parent to child
        Transaction tx = ses.beginTransaction();
        User u1 = new User();
        u1.setUserId (101);
        PhoneNumber p1 = new PhoneNumber();
        p1.setPhone (63636363);
        p1.setParent (u1);
        PhoneNumber p2 = new PhoneNumber();
        p2.setPhone (61616161);
        p2.setParent (u1);
        Set s = new HashSet();
        s.add (p1);
        s.add (p2);
        u1.setPhones (s);
        ses.delete (u1);
        tx.commit();
        // child to Parent
        User u2 = new User();
        u2.setUserId (102);
        PhoneNumber p1 = new PhoneNumber();
        p1.setPhone (81818181);
        p1.setParent (u2);
    }
}
```



```
PhoneNumber q2 = new PhoneNumber();
```

```
q2.setPhone(71717171);
```

```
q2.setParent(u2);
```

```
PhoneNumber q3 = new PhoneNumber();
```

```
q3.setPhone(91919191);
```

```
q3.setParent(u2);
```

```
Set s1 = new HashSet();
```

```
s1.add(q1);
```

```
s1.add(q2);
```

```
s1.add(q3);
```

```
u2.setPhones(s1);
```

```
ses.delete(q1);
```

```
ses.delete(q2);
```

```
ses.delete(q3);
```

```
tx.commit();
```

```
ses.close();
```

```
} finally
```

```
{ cleanup
```

→ For complete source code of above sources representing Bidirectional one-to-many association refer app10 (12) available in page no :- 77-82 in course material

→ inverse = "true" is optional in association-mapping, collection-mapping related tags. But it is recommended to place to give hint to hibernate slw that the current property also participate in reverse association mapping at Bi-directional mapping

→ By default all association mappings that are done by using association mapping tags and collection mapping tags will enable lazy loading by default.

To disable this lazy loading (or) to make this lazy loading as extra lazy loading we can use "lazy" attribute in the above said tags by having

lazy = "false" lazy = "extra" values.

↓
disable lazy loading
and goes for eager loading

↳ enables extra lazy loading

→ When lazy = "true" (default value) :-

When query.list() is called in association-mapping environment only empty parent POJO class objects will be created but the records into these objects and creation of child POJO class objects, records into these child objects will be stored by hibernate s/w only when programmer starts using those objects in his application.

→ When lazy = "false" :-

When query.list() called in association-mapping environment all objects will be created and initialized with records selected from the table irrespective of whether these objects are used in the application (or) not.

→ When lazy = "extra" :-

All objects (creation and initialization with table records (even for parent POJO objects) only when application starts working with result data. This is more lazy than lazy = "true" mode.

Q) ⇒ What is the difference b/w HQL joins query with batch keyword and without batch keyword?

Ans:- When HQL queries are executed without batch keyword

- ① lazy initialization takes place for child POJO class or right side POJO class related objects.
- ② generates two separate SQL queries one for parent table records (left side table) another one for the associated child table records (right side)

When HQL queries are executed with batch keyword

- ① Eager loading takes place for child POJO class object
- ② single select query will be executed to get parent ~~POJO~~ table and associated child table records.

→ better example app on HQL joins ~~is~~
HQLJoinsClient.java application in page no 2-803-81

⇒

You can go for unidirectional and bidirectional one-to-one association in 2 ways.

- ① Based on primary key based one-to-one association
- ② Based on Foreign key based one-to-one association

5
→ For unidirectional one-one association it is recommended to work with primary key base one-to-one association mapping.

→ For bidirectional one-one association it is recommended to work with "Foreign key" based one-to-one association

→ If there is possibility of taking parent pojo class object identity value as child pojo class object identity value then we can go for one-to-one (FK) based association.

ex:- every student of college should have library membership. The relationship b/w student and library membership is one-to-one r/s. Since student ID can be taken as library membership id, we can use one-to-one primary key association here. That means there is no need of foreign key column in child table or parent table. This indicates even though parent & child tables are not there in r/s their pojo classes and pojo class objects are participating in relshp based on identity value.

Student - tabs -

<u>sid (pk)</u>	<u>name</u>	<u>address</u>
101	zoya	hyd
102	chanki	hyd

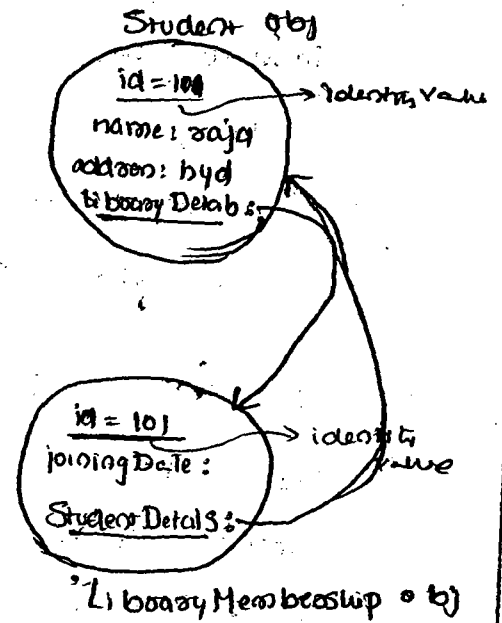
lib-membership:-

<u>sid (pk)</u>	<u>join date</u>
101	28-jan-2009
102	21-dec-2009

related POJO classes :-

```
public class Student
{
    int id, String name, String address;
    LibraryMembership libMembership;
    getXXX(), setXXX()
}

public class LibraryMembership
{
    int id; Date joiningDate;
    Student studentDetails;
    getXXX(), setXXX()
}
```



→ In the above diagram & discussion student object and library membership object are there in one-to-one relationship because the identity value of library membership obj is nothing but the identity value of student object. Because of this objects, objects are there in one-to-one r/s even though their related tables are not having foreign key constrained column.

→ In one-to-one primary key association the foreign algorithm is utilized to generate identity value of child pojo object based on the identity value of parent pojo object. This foreign algorithm is identity value generator algorithm and no way related with FK constraint at DB SW.

→ Based on the above discussion the id members of library membership class should be configured in mapping file having foreign algorithm as identity value generator.

4/19/10

→ It there is a guarantee every post class object will have one child post class object go for primary key based one-one association. It that guarantee is not there go for foreignkey based one-to-one association.

→ Every student of college will have one library membership. So we can go for primary key based one-to-one association mapping like student i library membership details.

→ Every citizen of india need not to have license but every license belongs to one citizen, so we can go for foreign key based one-to-one association b/w citizen, license details.

→ For example apply on pk based one-to-one association see app given in page no 88-90 ⇒ supplementary client application for application-9 (pk based one-to-one association to select and delete records.)

Select Test Java

```
import org.hibernate
```

```
// parent to child join allows to pass
```

```
Query q1 = session.createQuery("from StudentStatements
```

```
List l = q1.list();
```

```
for (int i=0; i<l.size(); i++)
```

```
{
```

```
Student st = (Student) l.get(i);
```

```
System.out.println("parent: " + st.getId() + " " + st.getName() +  
" " + st.getAddress());
```

```
LibraryMembership lib = st.getLibraryMembership()
```

```
st.getLibraryDetails();
```

```
System.out.println("child: " + lib.getId() + " " + lib.getJoiningDate());
```

```
}
```

```
System.out.println("-----");
```

```
// child to parent
```

```
q1 = session.createQuery("from LibraryMembership");
```

```
l = q1.list();
```

```
for (int i=0; i<l.size(); i++)
```

```
{
```

```
LibraryMembership lib = (LibraryMembership) l.get(i);
```

```
System.out.println("child: " + lib.getId() + " "  
+ lib.getJoiningDate());
```

```
Student st = lib.getStudentDetails();
```

s-o-p ("parent" + st.getId() + " " + st.getName() + " " + st.getAddress())

public void

// for deletion of records (parent to child)

Transaction tx = _____ ;

LibraryMembership lb = new LibraryMembership();

lb.setId(1);

Student st1 = new Student();

st1.getId(1);

st1.setLibraryDetails(lb);

lb.setStudentDetails(st1);

session.delete(st1);

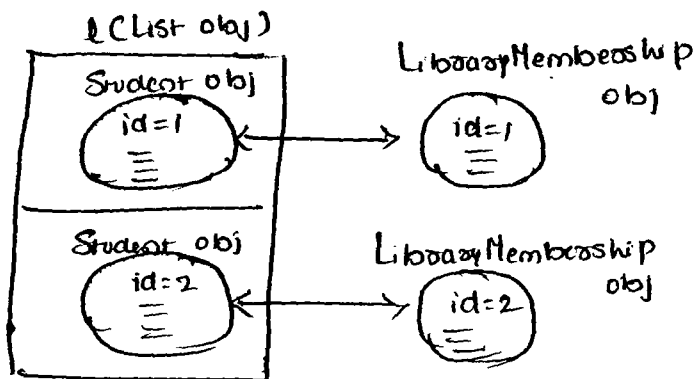
tx.commit();

session.close();

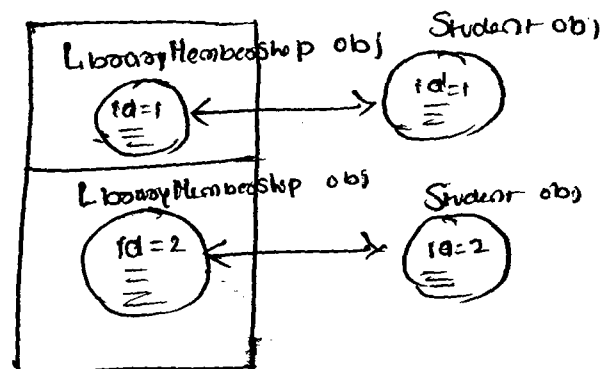
}

}

parent to child



child to parent



Handwritten text at the top of the page, mostly illegible due to blurriness.

from on 619110 Contine - from back-side

Main body of handwritten text, consisting of several lines of notes or a list, which are mostly illegible due to blurriness.

6/9/10

Q) what is the diff b/w Restrictions and Expressions class while working with CriteriaAPI?

ans! - The methods of Restrictions class allows to pass conditions to CriteriaAPI logic in the form of Java statements, whereas the methods of Expressions class allows to pass same condition as SQL query statements.

methods of Restrictions class are GE, LE, LD, GT and etc.

the methods of Expression class are sql (-)

One-One association (FK) :-

in one-one (FK) association mapping, child table should contain Foreign key column, pointing to PK column of parent table. This FK column should not have allow duplicates and NULL values, so this FK column should be applied with unique, not null constraint

*→ If there is a guarantee every parent POJO class object will have one associated child object then go for PK based one-one association.

If there is no guarantee every parent POJO object will have one child object but there is a guarantee for every child object to have one association parent object, go for "FK based one-one association mapping."

ex:- Citizen and licence contain one-one obj since there is no guarantee every citizen to have

license but there is a guarantee for every

license points to one citizen, so go for

FK based one-one association mapping in this situation

Example - For example application on FK based one-one

undirection based association see also APP (10)

given in page no 8-70-73

=> If child POJO class object can take, this identity
value from the identity value of ~~the~~ the associated
parent POJO class object, then use one-one FK
association.

=> If child and parent POJO class object are
looking take separate and independent identity
two values then use One-One FK Association.

** In FK based one-to-one association we can't work
with one-one tag because this tag is not having
column attribute to specify Foreign key column name.
based on which the association will be formed.
to overcome this problem use many-to-one having
unique = "true" attribute.

To configure the property that is separately
relationship see also line:- 1874 - 1878 of book let.

programmes - projects

programmes_id (PK with project)	project_id (FK with programmes)	composite PK
101	1001	
101	1002	
102	1001	
103	1002	
102	1002	

- First table shows Parent data
- Second table shows child data
- third table shows many-many association b/w parent table and child table data.

⇒ For example apply on many-many association mapping
 seebes app ⑩ given in 73-77

Bag - the Bag DS, given by Hibernate 5.1w is same as List DS. ab Collection. Itw. so ~~and~~
 Bag DS also allows ~~as~~ List DS duplicates.

to configure Bag DS type property in our HIB mapping file, we have to use the tag called 'Bag'

For example application on Bag DS seebes App ⑪

ab page no :- 98-101

- The app ⑪ ab booklet, shows association b/w (one-to-many) speaker and his sessions, speaker and his phone numbers, that means one parent table is there having two child table.

→ If HIB POJO class property type is java.util.List
 (or) org.hibernate.mapping.Bag then we can config
 that property in our HIB Mapping file either by
 using <list> tag or <bag>.

⇒ DB table is having single column, then programmer
 take the predefined wrapper classes Integer, Float
 String as HIB POJO classes.

especially this is useful, when associated table
 (child table) of certain table is having only one col

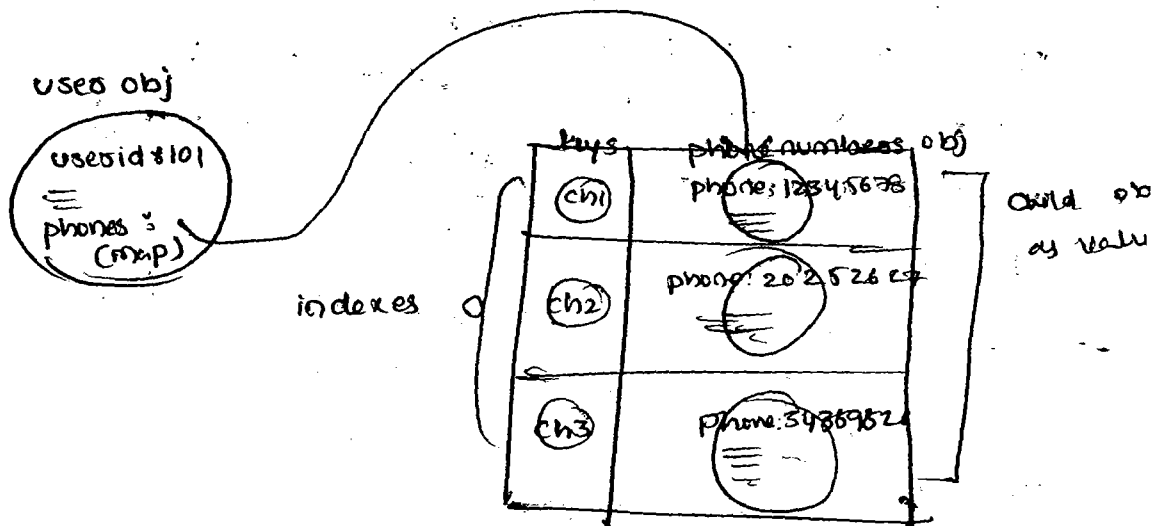
819110

Map -

each element of map DS, contains one key and one value
 key can be any object, and value can be any object, we can
 use key here to maintain index (or) count of value related
 objects.

In hibernate, we can use map DS property of parent POJO
 to maintain one (or) many child POJO class object to achieve
 one-many rls.

in this process, we can provide index for the child POJO
 class objects that are linked with parent object.



→ If you want to link child objects with parent objects without providing indexing for child object, we can use Set, List, Bag type property in Parent POJO class to link child POJO dependent child POJO class objects.

⇒ For example apply on ~~Map~~ Map ~~tag~~ tag based collection mapping see the supplementary handout :- 8/9/16

9/9/16

Cache - Bubbles is a temporary memory, that holds results (or) data across ~~the~~ multiple and uses that data

(or) result across the multiple same request given by application.

The process of storing result in cache (or) bubbles and using that result across the multiple request or ~~execution~~ execution of the application is called "caching"

(or) "Bubbling".

→ In client-server application, if caching is enabled at client side, it reduces network round trips b/w client and server applications.

→ Every browser window contains bubbles and stores, Request related response ~~data~~, the bubbles of browser window reduces NW round trip b/w browser window and webserver across the multiple same request given by browser window.

→ The caching (or) bubbling done in HB based application stores results collected from the DB SW in the form of HB POJO class objects. This reduces NW Round trip b/w HB based client application and DB SW across the multiple same request given by HB based client appln.

→ HIB supports two levels of caching.

① 1st level caching \ L1 caching (HIS session object)

② 2nd level caching \ L2 caching (HISessionFactory object)

⇒ 1st level cache is built in cache.

⇒ 2nd level cache is configurable cache.

→ 2nd level cache will be enabled only when program configures it manually.

→ every session object contains one level I cache storing its session object specific results.

→ Level II cache is called Global cache, bcz, we can store and manage the results generated by all HIS session objects.

⇒ (a1 → g1) → First request operation

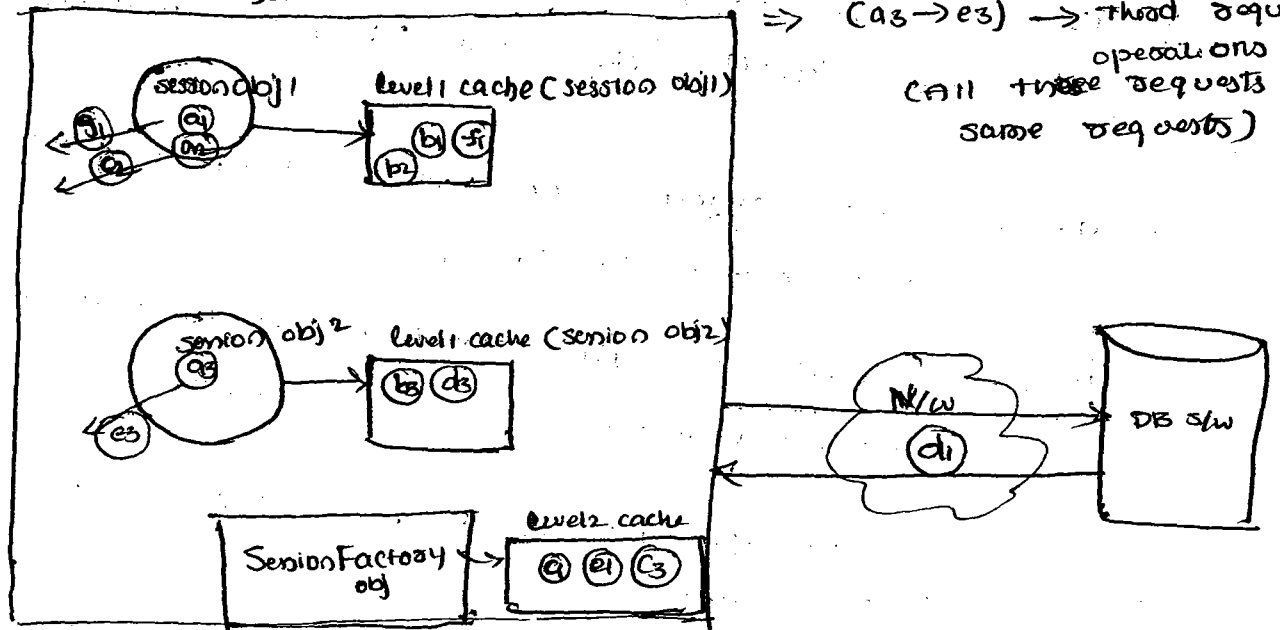
⇒ (a2 → a2) → Second request operation

⇒ (a3 → e3) → Third request operation

(All these requests are same requests)



HIS Based Client App



- ⇒ a₁ ⇒ session object ① gives initial request (select query)
- ⇒ b₁ ⇒ searches for that query request in level ① cache at session object ① (not available)
- ⇒ c₁ ⇒ searches for query result in level 2 cache (not available)
- ⇒ d₁ ⇒ HBS application interact with DB SW and gets query execution related results.
- ⇒ e₁ ⇒ HBS SW registers the results in level ② cache
- ⇒ f₁ ⇒ HBS SW also registers the results in level ① cache at session object ①
- ⇒ g₁ ⇒ session object ① gives the result to application

-
- ⇒ a₂ ⇒ session object ① gives same request once again
 - ⇒ b₂ ⇒ searches for the result in level ① cache at session object ①. (available)
 - ⇒ c₂ ⇒ session object ① collect the results from its own level ① cache and passes to client appn.

-
- ⇒ a₃ ⇒ session object ② at same client appn gives same request to DB SW
 - ⇒ b₃ ⇒ searches for the result in level ① cache at session object ② (not available).
 - ⇒ c₃ ⇒ searches for the result in level ② cache (available)
 - ⇒ d₃ ⇒ collect the result from level ② cache and registers with level ① cache at session obj ②.
 - ⇒ e₃ ⇒ session obj ② gives the result to client appn

⇒ Results in level ①, level ② caches will be stored in form of HIS POJO class objects.

⇒ The Results of level ① cache and level ② cache updated automatically at regular intervals.

⇒ We can specify parameters to disable level ② completely after certain amount of time or to delete Results Related POJO class objects from level ② cache after certain amount of time.

⇒ The level ① cache, keeps track of all the changes on HIS POJO class in a transaction and instead of generating multiple SQL queries, for this multiple changes it generates only one SQL update query reflecting all changes done on that object at end of the transaction.

⇒ For related info on level ① caching page 106 of the booklet.

⇒ HIS also support Query level cache, as sub portion of level ① cache. This Query cache holds Native SQL Query Result directly in its original format not in the format of HIS class object.

⇒ It is recommended to empty level ① and level ② cache time to time to know the changes done in DB slow time to time.

⇒ To get control on, level ① cache we need to work with built methods that are invocable on HIS Session object.

① session.evict(pojo obj) :-

Removes given POJO object, from level ① cache

② ses.clear() :-

empties level ① cache associated with current session.
Removes all HIS POJO objects from cache.

close() closes HB session, so also closes the level 0 cache associated with, current session object.

Q. what is dirty state object in HB environment?

Ans:- If you modify, Data of persistent state HB pojo class object manually in the client application, these changes will be synchronized with associated records in the DB table, only when session is flushed by calling session.flush();

Before calling this flush() changes done in persistence state pojo class object will be pending mode to synchronised with DB in this situation [before calling session.flush()] ~~method~~ we can say HB pojo class object is there in dirty state.

session.isDirty() method return 'true' → if session object is having any dirty state objects.

```
EmpBean eb = (EmpBean) ses.get(EmpBean.class, new Integer(1010));
S.o.p ("session is dirty mode?" + ses.isDirty()); → false
eb.setFname ("akash");
S.o.p ("session is dirty mode?" + ses.isDirty()); → true
ses.flush();
S.o.p ("session is dirty mode?" + ses.isDirty()); → false.
```

In the above code state of 'eb' object is dirty state after calling eb.setFname(), before calling ses.flush();

13/9/16

2nd level cache is configurable cache in HB Level, there are multiple vendors supplying slw to enable second level cache in HB applications.

some vendors are supplying the second level cache related jar files along with HB slw installation.

⇒ some example second level and opensource cache slw's are :-

- EHCACHE
- OSCACHE
- SWAPPACHE (SWOONCACHE)
- JBOSS EECACHE and etc.

⇒ most of the time developer prefer with EHCACHE, bcz it is fast, light weight, easy to use cache.

→ for details about various second level cache slw's and their caching strategies refer refer page 101

⇒ Procedure to enable 'EHCACHE' as second level cache in HB Application :-

step 1 :- every second level cache will have, one provide class name, try to know the provider class name for EHCACHE.

→ for EHCACHE, the provider class name is :-
org.hibernate.cache.EhCacheProvider

Note :- refer hibernate-home \ etc \ hb.properties file for various second level cache related provider class names.

Step 1:- Configure EhCache provider class name, in
hibernate.cfg.xml file.

in hibernate.cfg.xml:-

<session-factory>

=

<property name="hibernate.cache.provider_class">

~~org.hibernate.cache.EhCacheProvider~~

org.hibernate.cache.EhCacheProvider</property>

=

Step 2:- Add hibernate/lib/ehcache-1.2.3.jar file to
the class path environment variable.

Step 3:- enable second level cache in hibernate application

in hibernate.cfg.xml:-

⇒ <property name="hibernate.cache.use_second_level_cache">
true </property>

=

Step 4:- write EhCache.xml file along with other resources
of hibernate application, specifying the parameters related to
EhCache.

ehcache.xml:-

<ehcache>

<diskStore path="user.dir"/>

<defaultCache maxElementsInMemory="1000" ②

eternal="false" ③

timeToIdleSeconds="120" ④

timeToLiveSeconds="200" ⑤

overflowToDisk="true" ⑥

</ehcache>

1>

Note:- all second level caches uses primary memory (RAM), if needed we can also use disk memory (HDD) as an enhancement memory.

Note:- for details regarding ①-⑥ numbers refer the XML comments given in Hibernate/etc/ehcache.xml.

Step ①:- compile and execute HIB client application.

⇒ Query Cache is part of second level cache, it basically stores HQL, Native SQL Queries generated Results.

⇒ procedure to enable Query Cache on your HIB Application.

Step ②:- enable Query Cache, from HIB Configuration file.

```
<property name="hibernate.cache.use-query-cache" value="true" />
```

Step ③:- make HQL, Native SQL Query Results as cacheable Result from client application.

```
Query q1 = ses.createQuery("from EmpBean");
```

```
q1.setCacheable(true);
```

```
List l = q1.list();
```

Step ④:- set logical name for QueryCacheRegion from client application.

```
Query q1 = ses.createQuery("from EmpBean");
```

step ④ - make sure that 2nd level cache also enabled, before enabling the QueryCache.

Performing various operation on second level cache from Client app :-

Note - use SessionFactory object for these operation

- factory.close() → releases second-level cache, associated with session factory object
- factory.evict() (EmpBean.class) → Removes all emp bean pojo class object, from 2nd level cache.
- factory.evictCollection("phones") → If phones property (unambiguous in HBM mapping file, is related to collection F/w DS, all collection F/w DS objects representing phones property will be removed from second-level cache
- factory.evictQueries() :- Removes HQL, NativeSQL Query results from default Query Cache Region
- factory.evictQueries("test") :- Removes HQL, NativeSQL Queries from the Named Query Cache Region called "test"

Transaction Management

→ The process of combining set of related operations into single unit and executing them applying "do everything or nothing" principle is called transaction management.

While executing sensitive B/L and P/L, we must deal with Transaction Management.

ex! - Transfer money operation is composed with two

operations ① withdraw amount from source account

② ~~withdraw~~ deposit amount into destination account,

so we need to execute these two operations by applying do everything or nothing principle through transaction management.

→ Transaction Management applied on the code, applies ACID properties support on DB SW.

A → (Atomicity), C → (Consistency), I → (Isolation),

D → (Durability)

① → The process of combining related sub operation into single unit is called 'Atomicity'

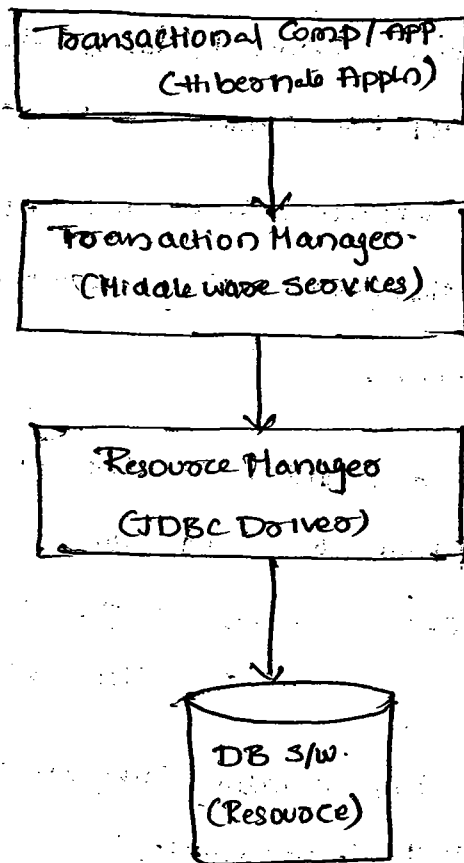
② → The process of getting guarantee, that rules kept on DB SW (like balance must not be negative) or not violated at the end of the transaction even though they are violated in the middle of Transaction is called "Consistency"

③ → The process of preventing concurrent operation on DB SW from multiple users and application by applying 'locks' is called "Isolation"

④ → The ability of bringing the DB SW back to normal state by using log files and Backup files when DB is crashed and using DB data for long time is called "Durability".

14/9/10

Architecture of Transaction Management



→ The appln (or) component on which transaction management is enabled, is called Transactional Application/component.

→ Transaction Manager is responsible to begin transaction, to commit (or) roll back the transaction on the application code.

→ Based on ^(DB S/W) ~~non~~ resources that are involved, there are two types of transaction

① Local Transaction:-

all the operations of application code, on which transaction management is enabled will deal with single resource (DB SW).

② Distributed

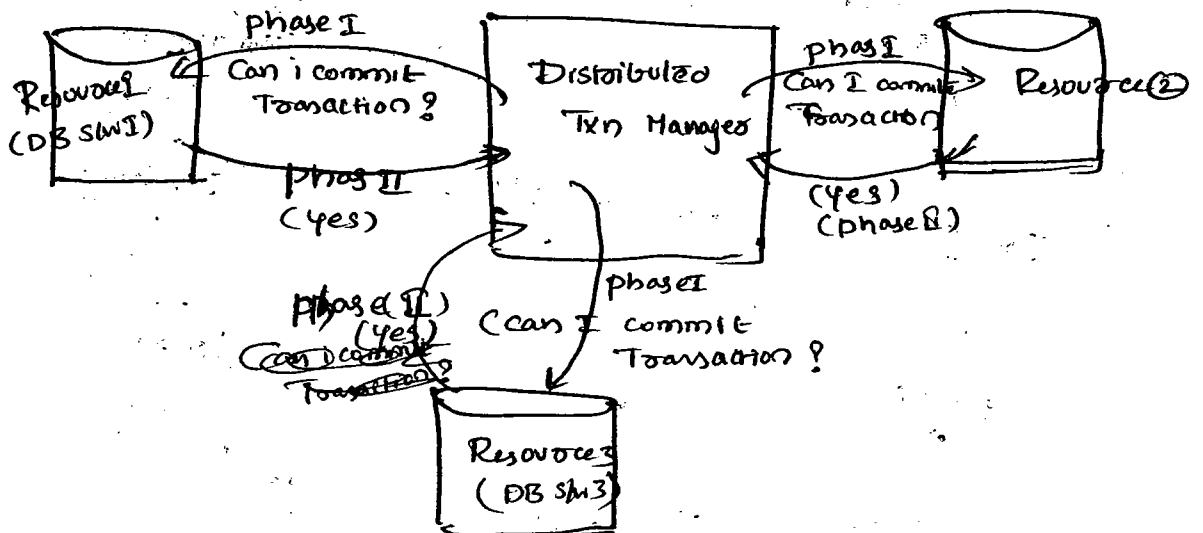
ex:- Transfer money operation b/w ~~same~~ two accounts of same bank.

② Distributed Transaction:- If multiple resources or DB SW are involved for various operations of application code on which transaction management is enabled then that is called Distributed Transaction.

ex:- Transfer money operation b/w two accounts of two different banks.

Distributed transaction runs based on 2PC prot (Two phase commit protocol).

① 2PC protocol :-



According to 2 PC protocol, in the phase I, Distributed

Transaction manager asks all DB S/W permission to commit the Transaction,

in phase II, all the DB S/W gives permission to commit the Transaction, then the distributed Transaction will be committed (OK) the Distributed Transaction will be roll backed.

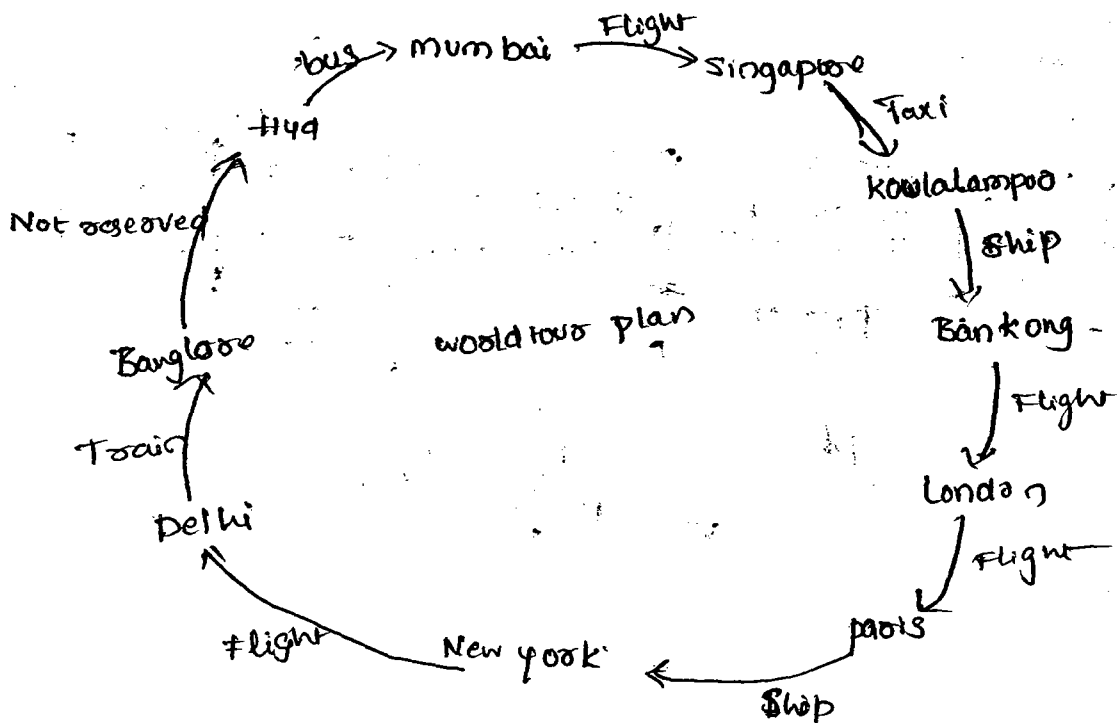
⇒ HJB support the local transaction management, but does not support distributed transaction management.

⇒ Spring, EJB Technology support both, distributed and local transaction management.

⇒ there are two transaction models

① Flat Transaction

② Nested Transaction.



→ when the above tour plan is given to a

sim application that runs with nested Flat transactions

if one (or) other journey ticket are not available it will cancel all other journey tickets.

Because each journey ticket booking operation will be taken as direct operations of main transaction so if one operation fails all other operations will rollback.

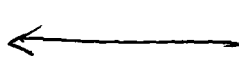
Flat Transactions:-

Main Transactions &

Journey 1 Ticket Booking,

Journey 2

Journey 3



when the above tour plan application gives to that runs with "nested transaction", even though one or other journey tickets are not available it will be ~~take~~ concerned the remaining journey ticket because, each journey ticket booking operation will be taken as the sub transaction as main transaction, so the success or failure of one sub transaction does not effect other sub transactions at main transaction.

Nested Transaction

Main Transaction {

subTxn1 {

journey1

subTxn2 {

Journey2

subTxn3 {

Journey3

*** → EJB, HB, and spring support Flat Transaction.

→ EJB & HB doesn't support Nested Transaction.

→ spring support Nested Transaction.

⇒ Sample Code that performs transaction management in HB environment:

```
public void txm1 ()  
{  
    Transaction txn = null;  
    try  
    {  
        tx = ses.beginTransaction();  
  
        persistence operation 1;  
        persistence operation 2;  
        persistence operation n;  
    }  
}
```

```

        txn.commit();
    } //try
    catch (Exception e)
    {
        try
        {
            txn.rollback();
        }
        catch (Exception e1)
        {
            //
        }
    } //outer catch.
} //method

```

⇒ Example application to perform Transfer money operation b/w two account of same bank having Transaction management?

> Alter table account add constant xyz primary key (acno);

step 1:- create account table in oracle db slw.

> ~~select~~ create table account (acno number(5), pk, accname varchar2(20), ~~id~~ number(8,2));

step 2:- Launch MyEclipse IDE and create java project having name "TXHBAPP"

step 3:- create DB profile for oracle by using myeclipse DB ~~exp~~ explorer.

step 4:- Add JTB capabilities to the project.

step 5:- perform reverse engineering on account table of oracle DB slw, by using DB profile.

NOTE:- After reverse engineering you get, account.java as pojo class, account-hbm.xml as mapping file.

step 6:- develop the following client application, having the support of Transaction management

ant - java :-

```
import org.hibernate.*;
import org.hibernate.SessionFactory;

public class TestClient {

    public void static void main (String args[]) throws Exception

    Sessions ses = HibernateSessionFactory.getSession();

    Transaction tx = null;

    try
    {
        tx = ses.beginTransaction();

        // operation 1 (with draw operation from acc amount)
        Query q1 = ses.createQuery("update Account set balance = (balance - ?)
                                   where accno = ?");
        q1.setLong(0, 4000);
        q1.setLong(1, 101);
        int res1 = q1.executeUpdate();

        // operation 2 (deposit amount operation to dest account)
        Query q2 = ses.createQuery("update Account set balance =
                                   (balance + ?) where accno = ?");
        q2.setLong(0, 4000);
        q2.setLong(1, 102);
        int res2 = q2.executeUpdate();

        if (res1 != 0 && res2 != 0)
        {
            tx.commit();
            s.o.p ("Tx is committed");
        }
        else
        {
            tx.rollback();
            s.o.p
        }
    }
    catch (Exception e)
    {
        try
        {
            tx.rollback();
        }
        catch (Exception e1) {}
    }
}
```

⇒ add objabely.jar to built path of the project.

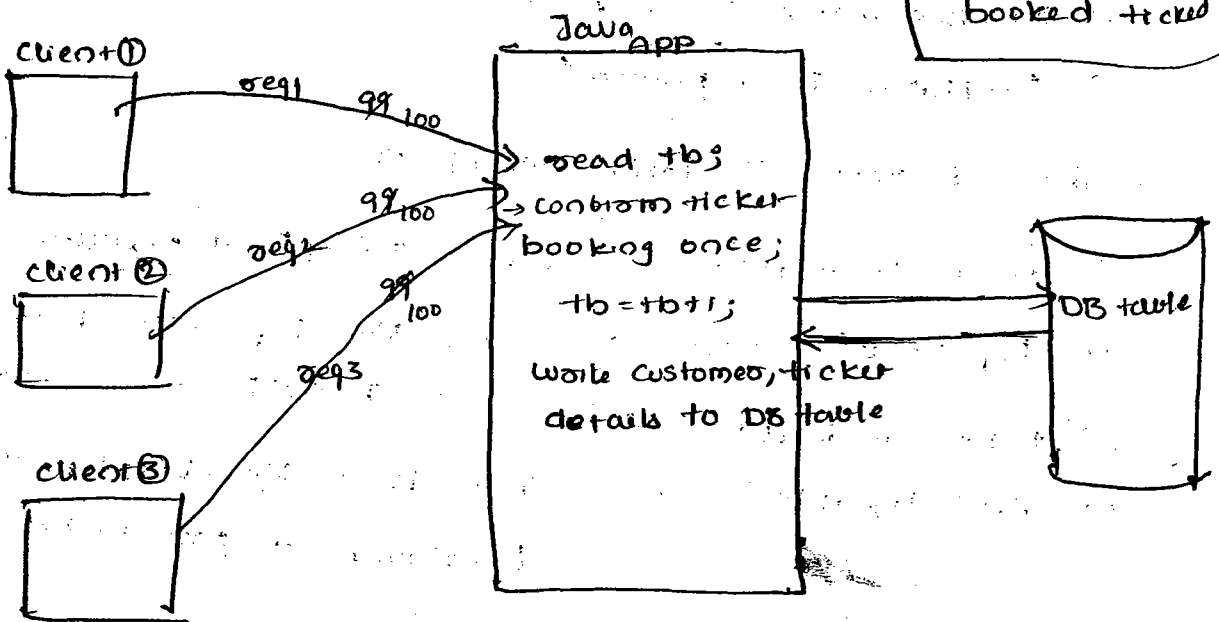
⇒ run the client application.

15/9/10

ISOLATION :-

→ If multiple users trying to access (or) manipulate DB data simultaneously & concurrently there is a possibility of getting corruption to the DB data. To prevent this problem apply locks on DB at various levels by using 'ISOLATION' levels & allow only one user (or) one app to manipulate DB data.

problem:-



In the above diagram, application and DB are allowing concurrent operation, so there is a possibility of booking same ticket for multiple passengers.

To prevent this problem use synchronization concept at java class level and apply locks on DB side by using isolation levels.

→ If application or DB is allowing another user (or) client to use the same data before complete

... (or) concurrent operation is called "concurrent operation" (or) "simultaneous operation" (or) "interleaving operation"

→ when DB allows, these interleaving or simultaneous operation there is a possibility of getting following problems.

To solve these problems we need to use different

Isolation levels

problems are:-

- ① Dirty Read problem
- ② Non-repeatable Read problem
- ③ Pantherm Read problem.

The solution Isolation levels are :-

- ① Read committed → solves Dirty Read problem
- ② Repeatable Read → solves Non-repeatable Read problem and Dirty-Read problem.
- ③ ~~Pantherm Read~~
- ③ Serializable → Dirty Read, Non-Repeatable Read, Pantherm Read problems.

⇒ we can set all these isolation levels on DB side, being from java applications by using JDBC code (or) HB code (or) any other persistence logic code.

⇒ Dirty Read problem -

⇒ If user A, user B have joint account holders in B.

⇒ DB SW is allowing uncommitted reads.

user A

user B

1) user A reads the balance of joint a/c
(Rs: 5000)

2) user A begins txn and deposits Rs: 3000 into joint a/c then
 $bal = bal + amt;$
(Rs: 8000)

(3) user B withdraws Rs: 1000 from account (joint)
 $bal = bal - amt;$
(Rs: 1000)

④ user A abort / roll backs

txn, so the balance becomes Rs: 5000. Here withdraw operation done by user 'B' is called "Dirty Read operation".

→ To solve above Dirty Read problem, Apply Read Committed Isolation level on DB SW. This may Applications to read only committed data of DB

→ In most of the DB S/W S Read Committed is default Isolation level.

ex:- ORACLE, MYSQL.

Non-Repeatable Read Problem :-

user A

user B

(1) At beginning of Tx user A gives
select query and gets few records
(let us assume 10 records) from
DB s/w.

(2) user B executes update query
that updates selected records
of user A in DB s/w.

(3) At the end of Tx user A,
he issue same select query
he gets 10 records with modified
date. this is called "Non-repeatable
read" problem.

To solve above Non-Repeatable read problem
use Isolation level Repeatable Read, which applies
'write locks' and 'Read-committed' mechanism on

DB. Due to this, ~~So~~ This Isolation levels also solve
Dirty Read problem.

=> Panthon-Read problems

User A

User B

① At beginning of Tx User A gives select query and gets few records (let us assume 10 records) from DB SW

② User B insert more records (let us assume 4 records) in same table which also satisfies select query condition of User A

③ At the end of Tx User A reissues same select query and he gets 14 records. getting these 4 extra records is called Panthon read problem.

To solve Dirty Read, Non-Repeatable Read, Panthon problem Serializable Isolation level, but it uses apply Read & Write Locks on DB.

FL

=> TL (or) PL (or) DB designing team decides the Isolation level but programmer is responsible to configure apply this Isolation level on DB SW.

-> ~~Some~~ DB SWs does not support all Isolation level

for example

-> Oracle doesn't support Serializable Isolation level

-> MySQL support all Isolation levels

⇒ Setting "Isolation level" on DB SW, from JDBC code :-

con • setTransactionIsolation (Connection • TRANSACTION_SERIALIZABLE);

(OR)

con • setTransactionIsolation (8);

here all possible values are :-

	JDBC
Connection • TRANSACTION_READ_COMMITTED	→ 2
Connection • TRANSACTION_REPEATABLE_READ	→ 4
Connection • TRANSACTION_SERIALIZABLE	→ 8

→ setting Isolation level from ^{on DB SW} HB application :-

In HB configuration file,

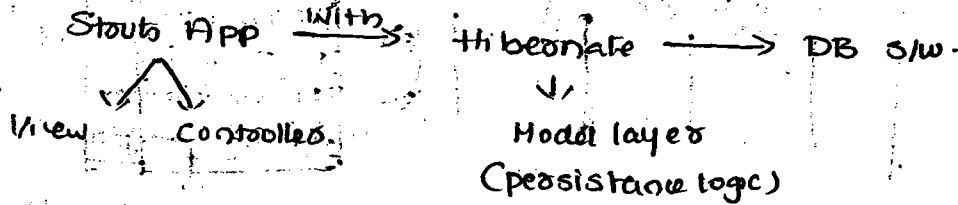
{
<property name = "hibernate • connection • isolation" > 4 </property>
(OR)
<property name = "hibernate • connection • isolation" >
REPEATABLE_READ </property>
}

Other possible values are :-

	HIBERNATE
READ_COMMITTED	→ 2
REPEATABLE_READ	→ 4
SERIALIZABLE	→ 8

STRUTS WITH HIBERNATE :-

Struts with hibernate Application means, we want to make struts application interacting with DB sw by using HIB persistence logic.



→ In struts with hibernate Application, Struts Action class contains Business Logic and uses P.O.L logic of HIB to interact with ~~DB~~ DB sw.

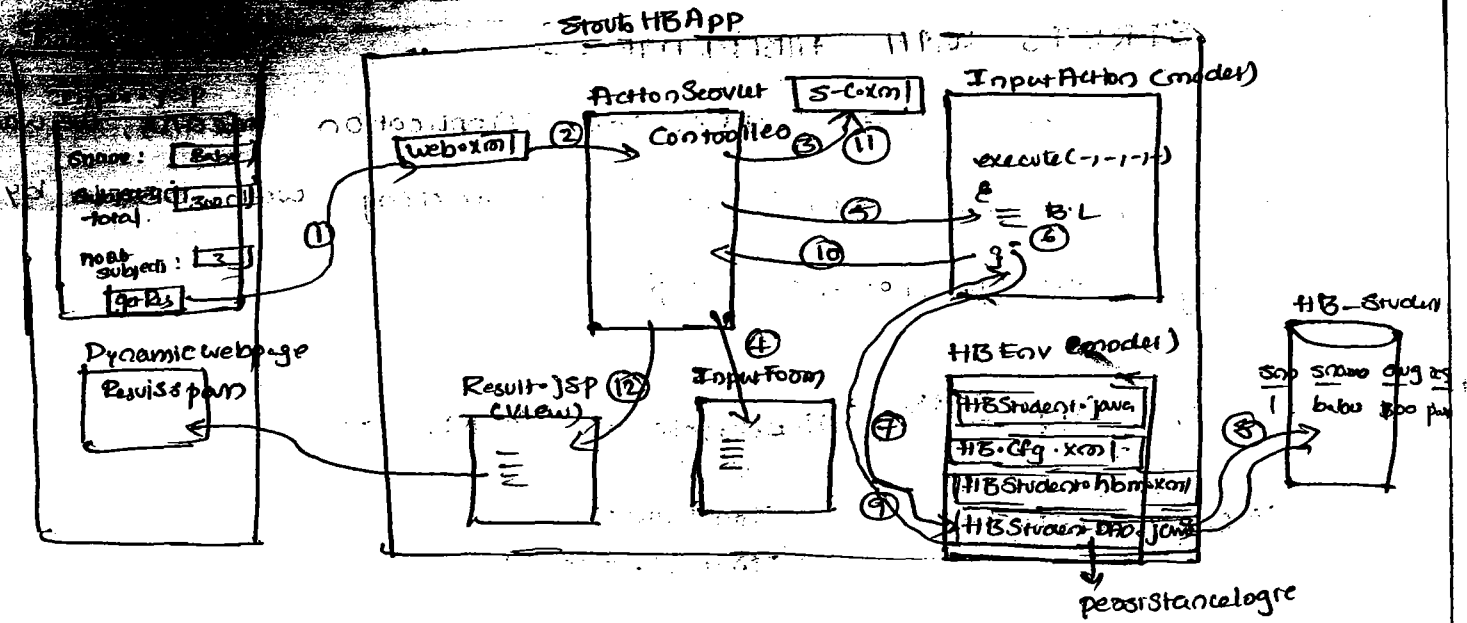
→ The java class are component that contains purely persistence logic and separate this P.O.L from other logics of the application is called DAO (Data Access Object).

→ The DAO makes P.O.L as reusable, flexible logic.

• Reusable means the persistence logic of DAO can be used by multiple Resources of the application.

• flexible means any modification in P.O.L, does not affects other logics.

→ The MyEclipse IDE can generate HIB persistence logic based DAO dynamically.



There are two approaches, to develop struts with Hibernate Application.

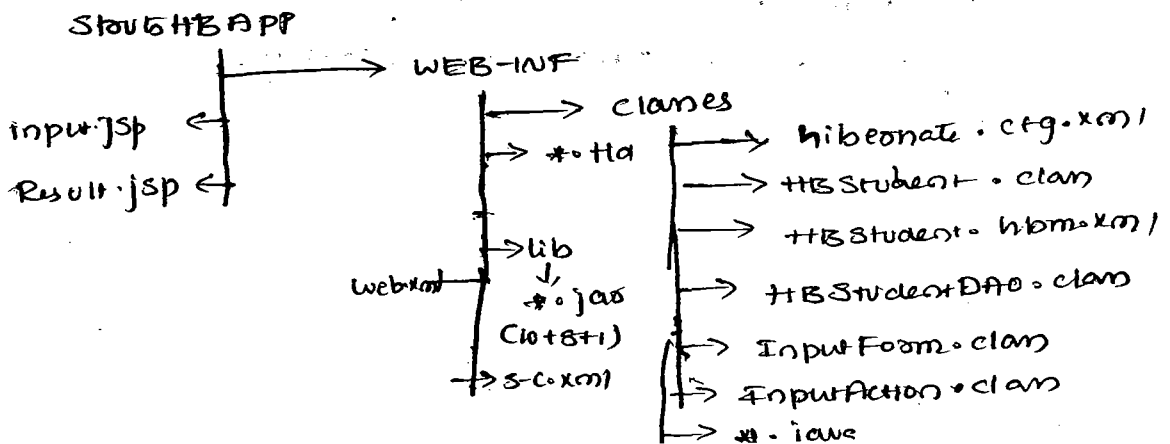
Approach 1 :-

By creating HBE session object in struts
Action class OR DAO class

Approach 2 :-

By creating HBE session object in user-defined
plug-in class of struts

→ If above application is developed manually based on Approach 1 Deployment Directory structure as shown below.



Jar files in the class path:-

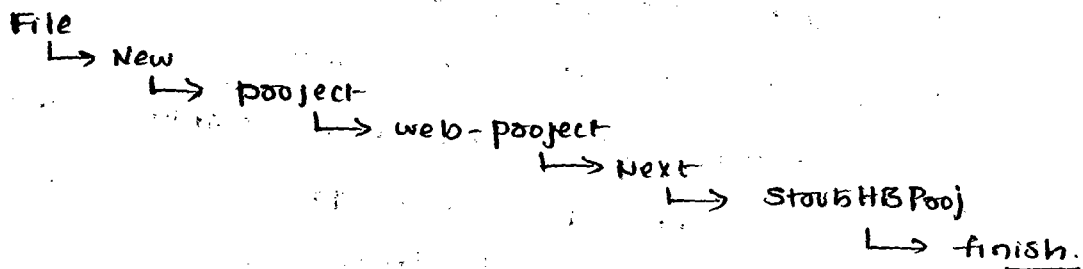
- Servlet-api.jar
- struts-core-version.jar
- hibernate3.jar

jar files Required in WEB-INF/lib folder

- 8 HIB jar file
- 10 struts jar file
- 1 ojdbc14.jar.

→ Procedure to develop above diagram based Applet, by using MyEclipse IDE

step 1:- create web project in myEclipse IDE.



Make sure that HIB-student table is there in oracle DB slw, by having PK constraint on sno column.

o HIBstudent

sno (pk)	sname (v2)	total (num)	AVG (num)	Result (v2)

step 2 create ORACLE DB slw Related DB pool in My-Eclipse IDE.

step 3:- Add HIB capabilities to the project

Right click on project

↳ myeclipse

↳ Add HIB capabilities

↳ select HIB-3.1 core Librar

HIB-3.1 Advance Lib

↳ Next

↳ Next

oop (already create DB problem)
 → next
 → de-select & Create SF class
 → finish
 add show_sql property

Step 4 :- problem @ HJB Reverse engineering on HJB_Student table.

goto DB Browser window

→ Right click on HJB-STUDENT TABLE
 → HJB Reverse engineering

→ java src folder :- /src/HJB proj/src

→ select 1st, 2nd, 3rd check box.

→ de select 4th check box.

→ select 5th check box. (JAVA DataAccess Object)

DAO type :- @ basic DAO

→ Next

→ @ HJB DType.

→ next

→ finish

NOTE :- change IDE generate algorithm to increment

algorithm in the generated mapping file (HJBStudent.hbm.xml)

NOTE :- Add Transaction management support for

non select operations related logics in the generated DAO class.

Step 5 :- Add stub capabilities to the project.

Right click on project

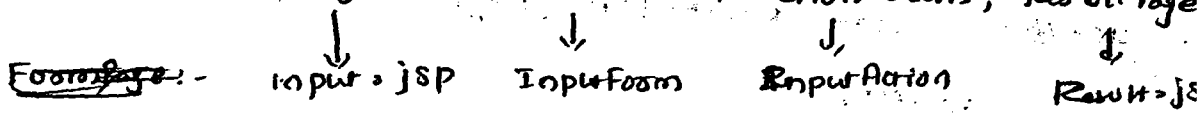
→ my Eclipse

→ Add stub capabilities, *

→ select stub 1. x

↓
 finish.

Step ①:- Add FormPage, FormBean Class, Action Class, ResultPage



Right click on project

↳ new

↳ other

↳ MyEclipse

↳ webStu

↳ stu1-2

↳ stu1-2 Form, Act, JS

Next

create all mentioned class

Step ②:- Right following code, in the execute (-, -, -, -) in InputAction class.

InputAction.java

≡

execute (-, -, -, -)

{

InputForm if1 = (InputForm) form;

String name = if1.getName();

int nosub = Integer.parseInt(if1.getNoSub());

int total = Integer.parseInt(if1.getTotal());

float avg = (float) total / nosub;
 ~~if (avg < 35) res = "fail";~~
 if (avg < 35) res = "fail";
 else res = "pass";

HBStudentDAO dao = new HBStudentDAO();

~~dao =~~

HBStudent st = new HBStudent();

st.setName(name);

st.setTotal((Long)total);

st.setAvg((double)avg);

st.setResult(res);

dao.save(st);

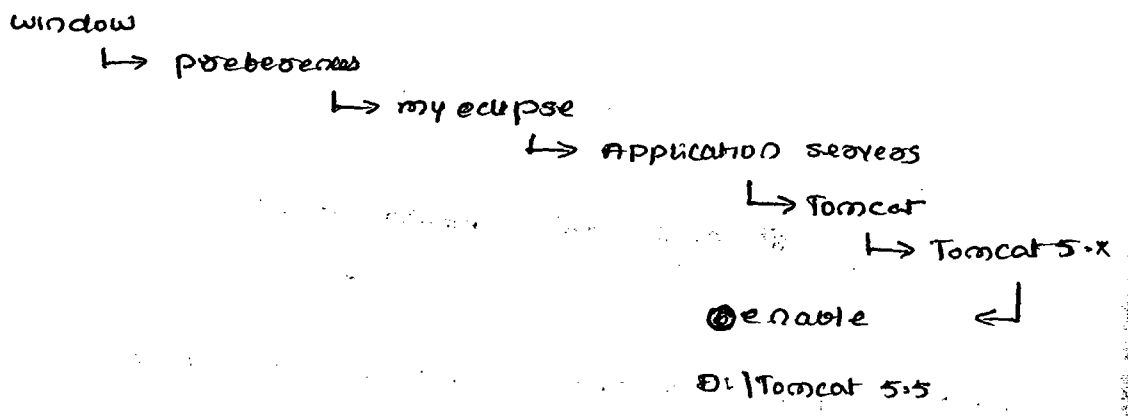
request.setAttribute("result", res);

return mapping.findForward("success");

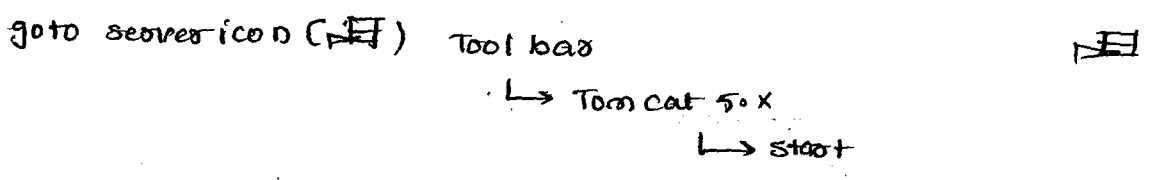
add Result.jsp page to the web-root folder of project.

 Student Result is <x= request.getAttribute("result") x>

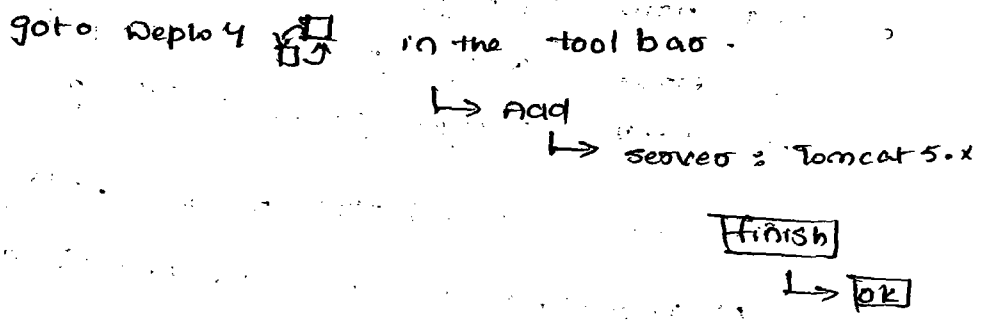
Step 1:- Configure External Tomcat server to myEclipse IDE



Step 2:- Start Tomcat server from MyEclipse IDE.



→ Deploy the project in Tomcat server from MyEclipse IDE



Step 3:- Test the Application.

⊗ open browser window from tool bar
→
http://1h:2020 / HB3100b

17/9/10

struts with Hibernate using plugIn

If we want to execute certain logic in struts appln, only for one time, for the all request coming to all the action classes, then we can define plug-in class and place logic in that plug-in class.

The java class that implements org.apache.struts.action.PlugIn interface is called plug-in class in ~~hibernate~~ struts.

→ every PlugIn class must be configured in struts configuration file by using <plug-in> tag.

→ The struts plugIn class execute only one time, that is when ActionServlet object is created in struts fw. It is create activated.

→ If you want to make all action classes as struts appln using single HIB session object for all the request, then place that logic in the user define PlugIn class as struts application, and use those objects in all action class by receiving them in the form of ServletContext attribute values.

* ServletContext attributes are global attributes in web-app. So they are visible all the web-resources as web-app.

→ procedure to develop yesterday's application without DAO class and by adding user defined struts-plugIn class -

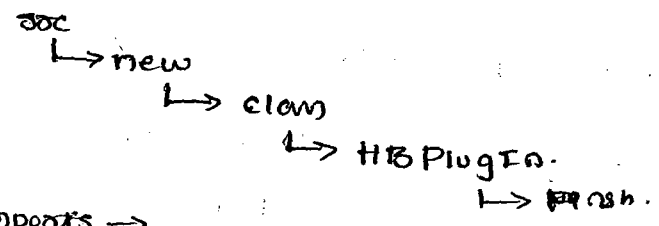
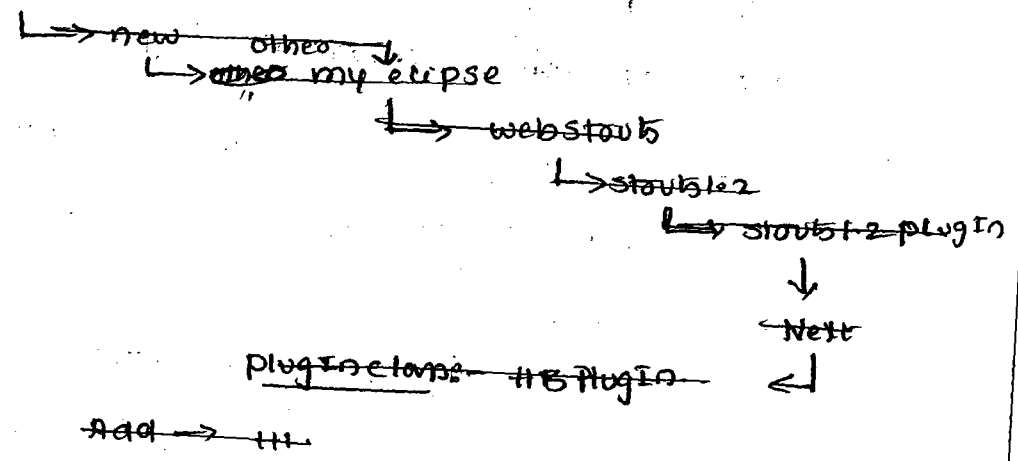
Note:- user define plugIn class as struts application always resides in WEB-INF folder as struts application.

perform HB Reverse engineering on HB student table. [don't select DNO option]

step 1:- same as previous application.

step 2:- ~~use~~ Add user defined struts-plugin class to the project. and place logic to create HBSession object in that class.

Right click on project



=> imports -> `public class HBPlugin implements Plugin {`

```

import javax.servlet.*;
import org.apache.struts.action.*;
import org.apache.struts.config.*;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class HBPlugin implements Plugin {
    Session ses = null;

    public void init(ActionServlet server, ModuleConfig mcg)
        throws ServletException {
    }

    public void init(HBPlugin plugin, ActionServlet server)
        throws ServletException {
    }
  
```

try {

// create HBSession obj

ses = new Configuration().configure().buildSF().openSession();

// get ServletContext obj

ServletContext sc = servlet.getContext();

// keep HB session obj in ServletContext attributes

sc.setAttribute("HBSES", ses);

} catch (Exception e)

{ e.printStackTrace();

}
} finally {
public void destroy() {

try {
ses.close();

} catch (e)

}
} // close

Step 1: - configure above created plugin class with struts applet, in struts-configuration file.

right click on project

↳ New

↳ other

↳ myeclipse

↳ webstruts

↳ struts1-2

↳ struts1-2 plug

↓

Plugin class:-
HBPlugin

finish

→ The init() of struts Plugin class, executes during the start up of the struts applet, when ActionServlet object is created.

→ The destroy() of struts Plugin class, executes when struts application is un-deployed, stopped (or) reloaded.

same as step ⑥ of previous project.

! - write following logic in the execute() of

Action class →

step ⑩! -

```
class InputAction
{
    =
    AF execute() throws Exception.
    {
        InputForm if1 = (InputForm) form;
        // read the data
        long total = Long.parseLong(if1.getTotal());
        String name = if1.getName();
        long nosub = Long.parseLong(if1.getNoSub());
        // get ActionServlet obj
        ActionServlet servlet = getServlet();
        // get access to ServletContext obj
        ServletContext sc = servlet.getServletContext();
        // read HB session obj from sc attribute
        Session ses = (Session) sc.getAttribute("HBses");
        // HB logic
        double avg = (double) total / nosub;
        String res = null;
        if (avg < 35)
            res = "fail";
        else
            res = "pass";
        // write HB persistence logic to insert the data
        HBStudent st = new HBStudent();
        st.setName(name);
        st.setTotal(total);
        st.setAvg(avg);
        st.setResult(res);
    }
}
```

step ①:-

step ⑩ → step ④ are same as .

step ⑥ → step ⑫ of previous app.

```
Transaction tx = ses.beginTransaction();
```

```
ses.save(st);
```

```
tx.commit();
```

```
// send session to result.jsp
```

```
request.setAttribute("result", ses);
```

```
action mapping.findForward("success");
```

```
}
```

```
}
```

** Note ①:- Plugin class creates session object only once and makes all Struts Action classes using that object for all the request

→ The DAO class separates persistence logic from other logics at the application and also makes the persistence logic as reusable logic in multiple Struts Action classes.

so you can add both plugin, DAO support in struts with HB application.

But the DAO class should use that HBSession object that is created in Plugin class.

⇒ For EJB SessionBean with HB application refer application given in page no:- 92 to 96

Annotations Based Metadata :-

- Data about data is called "metadata".
- configuring resources, to pass their details to underlying SW where the resources will be utilized for execution also comes under Metadata operation.
- configuring server in web.xml file, to pass the details of server to underlying webserver comes under Metadata operation.
- In earlier days programmers have taken, the support of xml file, for resources configurations and for metadata operations.
- In recent days, we can also use the java statements called Annotation as alternate to xml files based annotations.
- The Metadata operations done on the resources or on the code of resources will apply marking and provide different identity for them during execution.
- xml files based Metadata operations gives flexibility towards modification, but does not give good performance because reading data from xml files is always quite complex operation.
- since Annotations are java statements, they give better performance but does not provide flexibility towards modification.
- Annotations are there in java from JDK initial versions or Documentation related Annotations like

@author, @param, @since, and etc.

→ Annotations per programming are introduced from JDK 1.5

Syntax -

→ @ <annotation-name> (param1 = value1, param2 = value2
.....)

(No semicolon at end)

⇒ Each annotation is like an XML tag and parameters of annotations are like XML tag attributes

⇒ @Override, @Override, @Deprecated are built-in annotations of JDK 1.5

⇒ all high-end technologies that are given based on JDK 1.5, supply their own annotations to configure resources of the application.

⇒ J2E 3.0.x onwards, Spring 2.5 onwards, EJB 3.x onwards, Struts 2.x onwards, Servlet 3.0 onwards all supporting annotations.

⇒ If annotations and XML file, both are configured for metadata operation, then settings done in XML file will be effected.

⇒ If programmer is using XML file, for metadata operations even though the application is already having annotation based metadata configurations, then it indicates that programmer is interested

Annotations, configurations, without disturbing the source code.

→ we can apply annotations at 3 levels

① Resource level (on class or interface)

② Field level (on the member variables)

③ method level

⇒ To work with annotations based HB programming use either hibernate 3.4 slw completely

or

use hibernate 3.2.5 slw ⊕ HB 3.3 annotations

software to gether.

⇒ All ~~HB~~ annotations of HB programming are designed based on JPA (Java persistence API), EJB 3.0 specification

⇒ all annotations are special interface, having

@interface keyword.

implementation of this interfaces having @interface keyword

which allows to use annotations for metadata ~~operations~~ operations

⇒ For API documentation integration annotation of HB programming, go to ~~hibernate~~ hibernate 3.3 annotations ~~home~~ -home\

doc\JPA-API\index.html file.

⇒ For reference example applications based on annotations

examples

the test codes of HB 3.3-annotations home directory

⇒

19/10

There are three strategies to perform Inheritance

1 Table per hierarchy

2 " " subclass

3 " " concrete class

we can use

@Inheritance Annotation per Inheritance

mapping, use

@DescriptorValue annotation to spe

descriptor value.

use @DescriptorValue

annotation to con

Table per hierarchy strategy

@Inheritance (strategy = InheritanceType.SINGLE_TABLE

Table per subclass strategy

@Inheritance (strategy = InheritanceType.SUBCLASS

Table per concrete class hierarchy

@Inheritance (strategy = InheritanceType.TABLE_PER_CLASS

for example application on table precision hierarchy model @

Inheritance mapping suber app @ in supplementary

Handout given on 18/10

* FOR Example application on one-one PK annotation

Table application @ @ supplementary booklet on

along with @one-to-one annotation, if you write

@Primary key join column annotation two that will

One-to-one PR Association

along with @One-to-One Annotation, join column

Annotation is written specifying PK column, then the

will act as One-to-One PK Association.

Generators in Annotations

We can use generator algorithm to generate identity values of POJO class dynamically by using some algorithms

```
→ public class Student
```

```
{
```

```
int sid;
```

```
@Id
```

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

```
int sid;
```

```
---
```

```
---
```

```
}
```

GenerationType = AUTO

↓

picks up the algorithm dynamically

based on the capabilities of underlying

DB like native algorithm

Other possible values for strategy parameter:

GenerationType =

GenerationType = SEQUENCE

(uses sequence algorithm)

GenerationType =

GenerationType = IDENTITY

uses identity algorithm

GenerationType = TABLE

uses hilo algorithm

~~class Stu~~

Procedure to configure uses above sequence as Identity

generator:

```
class Student
```

```
{
```

```
@Id
```

```
@Column (name = "sno")
```

```
@GeneratedValue (strategy = GenerationType.SEQUENCE,
```

```
generator = "my_seq")
```

```
Integer sno;
```

```
---
```

```
}
```

Composite Identity Configuration:

If only one member variable of the POJO class is configured as Identity field, then it is called singular Identity field. use <id> at the mapping file are use @Id anno for this operation =

If more than one variable of the POJO class are configured as Identity fields then it is called composite

Identity field configuration. Note: - we cannot work with Identity Value generator algorithms while working with composite Identity field.

mapping file based composite Identity field configuration:

```

public class Student {
    int sno;
    String name;
    String address;
    int age;
    get xxx() & set xxx
}

```

from hbm file

```

<hibernate-mapping>
<class name="Student" table="Student">
<composite-id>
<key-property property="sno" column="sno"/>
<key-property property="sname" column="sname"/>
</composite-id>
</class>
</hibernate-mapping>

```

Annotation: Based Composite Identity Field Configuration :-

```
⇒ @Entity
   @Table (name = "Student_Tab")
   @IdClass (StudPk.class)
   public class Student
   {
StudPk
   @Id
   @Column (name = "stid")
   StudPk id;
   String address;
   int age;
   =====
   getters & setters
   }
```

```
@Embeddable
public class StudPk implements Serializable
{
   int sid;
   String name;

   getxxx & setxxx
   =====
}
```

← End of Hibernate →

