www.tutorial4us.com

Tutorial4Us

College Projects

```
void main()
{
 int no;
 clrscr();
 printf("Hello");
}
```

Basic Program

C Tutorial

JAVA Programming

Hibernate

(Sekhar Sir)

www.tutorial4us.com

Persistence :-

→ Persistence is a term. which denotes perminent storage (or) perminent area.

→ An application data stored perminently is called persistent data.

→ The logic used to make some data as persistent is called as "persistence logic".

→ As a Java programmer, we have the following options to make some data as persistent data.

① A Flat file

② An xml file

③ A DataBase

→ A Flat file is nothing but a file which is not related with any technology (or) any framework

→ In Java, persistent data means, the data which has more lifetime then the execution of a Java application. It means eventhough the Java appn execution is terminated but still the data produced (or) generated by the appn is still alive

→ As a Java programmer, if we use a Backend as a flat file then we have I/O streams for Reading (or) writing the data in a file (or) using a file.

→ A Java program use InputStream for read the data from a file, output stream for writing the data into a file.

→ with files, we got the following drawbacks.

① files are compleeted complexed coork, Bcz we need to write some programing code

② for files, we donot have any common query Language like DataBases.

③ files can't be used for storing large amounts of data.

④ files are very less secured

⑤ we can't maintain integrity b/w the data

⊛ Xml as a backend :-

→ Using Xml as a backend is almost equal to a Flat file as a backend. But the difference is a flat file can just store the data, an Xml file can store the data and also can describe the data

Eg:- Flat file

a.txt

100	ABC	500

→ It is not describe the file, just store the data only.

Xml

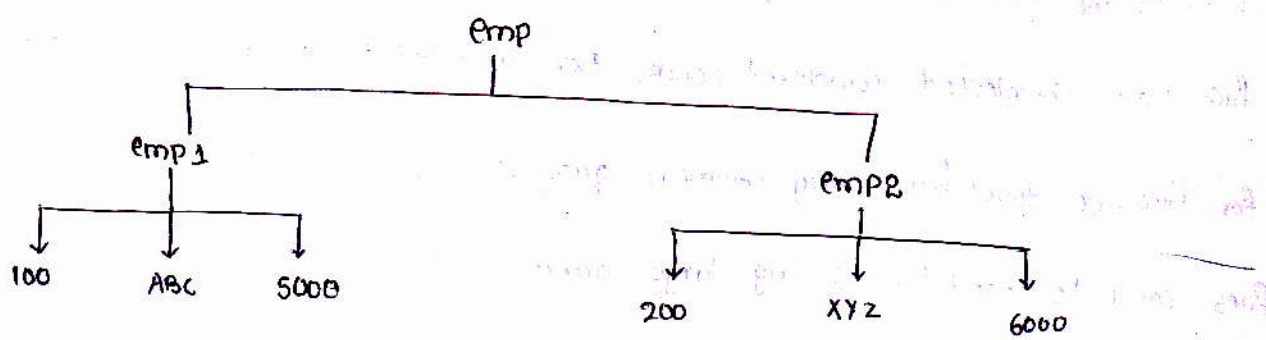
```
<emp>
  <empid> 100 </empid>
  <ename> ABC </ename>
  <sal> 500 </sal>
</emp>
```

→ This file store the data and also clearly describe the data

→ with Xml as a backend, we have the following problems

① for constructing an Xml file we must follow some rules.

② for searching for some data, always an Xml must be loaded and parsed & then a tree structured is created and then search begins. It means Xml is slower then databases.



③ Xml files are less secured.

④ We can't manage integrity blw the data (or) referential integrity blw the data

NOTE:- The benfit of Xml is, we can transfer the data across Network in Language Independent, platform Independent and Browser Independent.

*) DataBase as a Backend :-

→ with FileManagement System we got some drawbacks, so we entered into DataBase management System.

→ with DataBases we have the following advantages

① DataBases are Simple to work with it.

② we have a common query Language (SQL).

③ The data stored in a DataBase is secured

④ A DataBase can store large amount of data

⑤ we can manage the Integrity blw the data in a database

.
.
.
.
etc.....

05/02/2013

Q:- Why we need to go for object Relational Mapping (ORM)?

→ Sun micro Systems given JDBC technology for java applications to talk with databases.

→ Eventhough JDBC is a good technology for connecting with databases. But in a realtime environment, the technology has produced some problems (or) drawbacks.

① while creating JDBC persistence logic, we keep the DataBase structure in the mind and we write the JDBC application code. After the application is compleered, if the database structure is modified then wee need to make the changes in the entire application code, when ever the JDBC code is added. It makes burden on a programmer. (db ⟶ code)

② In applications of real time, the data will be in objects & the data will be transfered with in the application, also in the form of objects but when storing it in database the JDBC technology cannot transfer objects blw application & a database. So an object is converted to a text & then the text will be transfered to the database Here the application code is increased for converting an Object into a text or Text into an object & also the burden on programmer will be increased.

(Objects ⟶ Text)

③ JDBC technology uses SQL operations & the SQL based persistence Logic is a data base dependent (JDBC persistence Logic DataBase dependent).

④ In Jdbc technology, we have all checked Exceptions, whenever we add JDBC code in our application, we need to add either try & catch blocks or throws. It increases the burden on programers (checked Exceptions)

⑤ In JDBC technology some repirative statements are require to perform any operation on database. This repirative statement is called Boiler-plate code. Bcoz of this code a burden on a programmer will be increased. (Boiler-plate code)

⑥ In JDBC, a programmer has to takecare about connection management explicitly. It means the programer has to handle the connection opening & connection closing with a DataBase. if it is not handle properly then at a later point of time, an application can run out of Connections (Connection management)

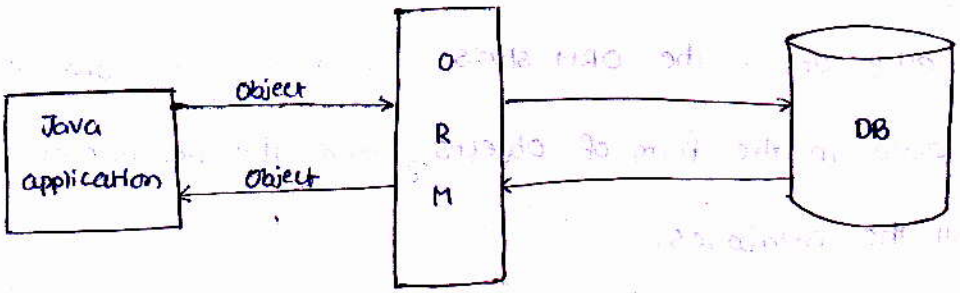
→ In order to overcome the above drawbacks of JDBC technology the third party vendors started providing a special kind of slw ORM software or tool.

→ while using ORM slw, it internally uses again JDBC technology for Connecting with database. It means, if ORM slws, we are indirectly using JDBC to talk with DataBases.

→ The major benfits of ORM slw's includes Data transfer in objects & the persistence logic independent of data bases.

Q:- What is ORM?

Ans:- ORM is a new kind of persistence s/w, which acts like a bridge b/w Java application & Database and transfers to or form data in the form of objects

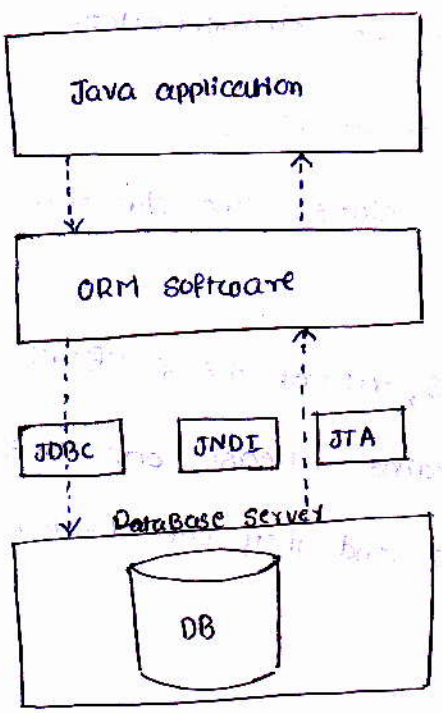


06/02/2013

→ ORM softwares internally uses JDBC, JNDI and JTA (Java Transaction API) for communicating a Java application with a DataBase in order to Transfer Objects b/w them.

→ an ORM s/w uses JDBC Technology to perform DataBase operations by connecting with it

→ an ORM s/w uses JNDI when Connection Pooling technique is applied for obtaining DataBase connections. and JTA is used while managing the DataBase transactions



The following are the some of the ORM s/w's given by different vendors

→ Hibernate ——→ JBoss (Soft tree)
→ iBATIS ——→ Apache
→ TopLink ——→ Oracle Corp
→ OJB (Object Java Bean) ——→ Apache
etc.→ JDO (Java Data object) ——→ Adobe

→ In Real time applications, the two ORM slw's prefered Hibernate, iBATIS.

→ Hibernate is the most ORM Populer slw when compared with the other ORM slw's

*** → The primary functionality of all the ORM slw's is same. It means the Data transfer will be done in the form of objects and the persistence Logic is independent of all the Databases.

Q:- What is Hibernate ?

Ans:- Hibernate is an openSource frame work, also called as an ORM tool. and It acts as a middle layer between a Java appn and DataBase & provides Persistence and query service in the form of Objects.

In other terminalogy, we call Hibernate as an abstraction Layer on top of JDBC technology.

Hibernate can be called as a tool and also as a framework. Bcz Hibernate follows the commonalities of frameworks.

Frame works follows the following commonalities

① Frame works are non-installable slw's. The slw of a Framework will be in the form of a set of jar files.

② Frame works are implementations, but not the specifications.

③ Every Frame work application contains atleast one configuration file

④ Frame works contains jar files and they are divided as main and dependent jar files.

Benfits / Advantages of Hibernate :-

① Hibernate has an Exception Translator, which converts checked Exceptions of JDBC into unchecked Exceptions of Hibernate. So, Hibernate has all unchecked Exceptions. So, The burden on a hibernate programmer has a reduced, because

a programmer is no need to handle the Exceptions explicitly.

② Hibernate has its own query Language called Hibernate Query Language (HQL) with this HQL, Hibernate queries becomes Database independent.

③ Hibernate has caching mechanism. It reduces the no.of round trips between a Java application and a DataBase. So, Hibernate increases the performance of an application.

④ Hibernate has versioning and Timestamp feature. with this feature, we can come to know how many no.of times a data is modified and also on what date at what time the data is last modified.

⑤ Hibernate Supports Inheritence and polymorphism.

⑥ with Hibernate we can manage the data stored across multiple tables, by applying Relationships. (assossiations)

⑦ Hibernate is a light-weight framework, because Hibernate uses POJO classes Objects for DataTransfer b/w an application and a Database.

etc.....

07/02/2013

POJO class :-

→ A Java class, which doesnot extend a base class or doesnot implement an interface, given by any technology or any framework of Java then such type of Java class is called a POJO class.

→ For example,

```
① public class Test
  {
    //methods
  }
```

→ here Test is a POJO class. A Java class by default extends java.lang.Object. but Object class is a core API of Java. So, A POJO class can extend on core API class of Java.

② public class MyException extends Exception

```
{
    //methods
}
```

→ here MyException class is a pojo class. Because the Base class Exception belongs to CoreAPI of Java.

③ class Demo

```
{
}
```

class Test extends Demo

```
{
}
```

→ here Test is a pojo class. Because the base class Demo is also a POJO class

④ public class Test implements java.io.Serializable

```
{
}
```

→ here Test is a POJO class. Because it is implementing Serializable(I) and this interface is belongs to CoreAPI of Java.

⑤ public class Test extends javax.servlet.HttpServlet

```
{
}
```

→ here Test is not a POJO class. Because it is extending HttpServlet and HttpServlet is a class of ServletAPI / Servlet Technology.

⑥ public class Test implements java.rmi.Remote

```
{
}
```

→ here Test is not a Pojo class. bcz it is implementing Remote(I) and this Remote(I) belongs to Rmi technology.

→ In Java, a light-weight framework is also called as non-invasive or non-intrusive framework.

→ An Heavy-weight framework is called invasive or intrusive framework.

Q:- What is a JavaBean?

Ans:- A JavaBean also a Java class, which is independent of any technology or a framework.

→ to say that a java class as a JavaBean the following two conditions must be satisfied

① A Java class must be "public"

② A Java class must contain a public default constructor.

→ Every JavaBean class is also a POJO class. But ~~every~~ a POJO class may or may not be a JavaBean class.

Eg:-
```
public class Demo
{
    public Demo()
    {
    }
    //methods
}
```

→ Here Demo is a JavaBean class and also a POJO class.

Eg:-
```
public class Demo
{
    public Demo(int a, int b)
    {
    }
    //methods
}
```

→ Here Demo is not a JavaBean class, Demo is a POJO class. Because there is no public default constructor in that class.

mapping in Hibernate :-

→ mapping is a mechanism of informing an ORM tool (Hibernate) about what POJO class object is need to be stored in what table of a Database

→ mapping is a process of writing mappings between a POJO class name to a table name and POJO class variables names to a table columns name.

→ mapping is a metadata, but not the data. Through mapping metadata, Hibernate stores or reads the data using a database.

→ Every ORM tool need mapping information, inorder to store the state of an object in a DataBase.

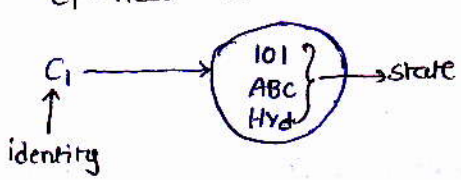
→ An Object has '3' elements called Identity, state, behaviour.

→ In ORM, Storing an Object is nothing but storing the state of an Object, but not the identity & behaviour.

→ For Eg:-

```
public class Customer
{
   private int CustomerId;
   private String customerName;
   private String customerAddress;
    → setters & getters
}
```

```
Customer c1 = new Customer();
```

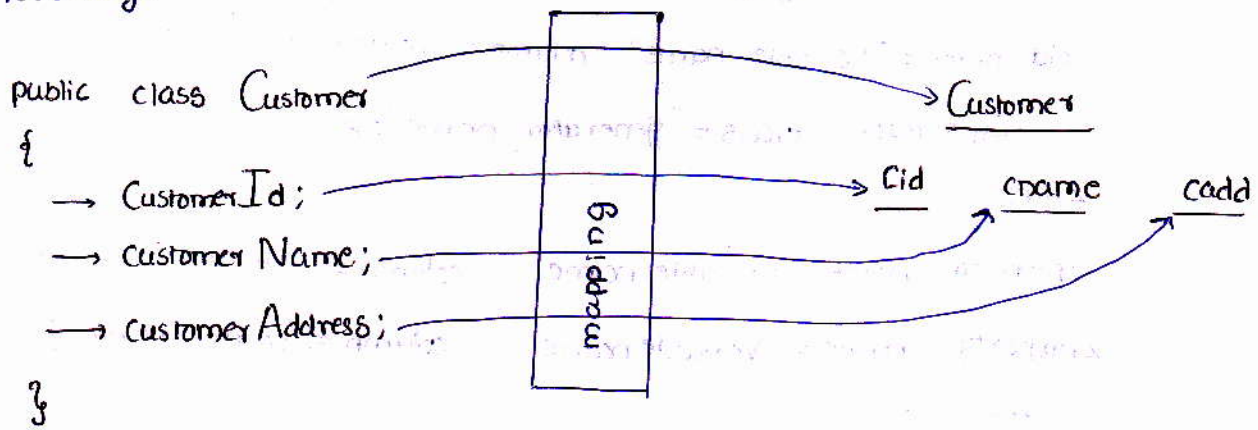



```
setXxx()
getXxx()   → behaviour
```

mapping is mandatory for an ORM tool, because an application can contain multiple POJO classes and DataBase can contain multiple tables,

So an ORM tool can't understand what class object is need to be stored in what table. To give this information, a programer must Construct / create mapping.

→ A mapping between a Customer class to Customer table coill be like the following.

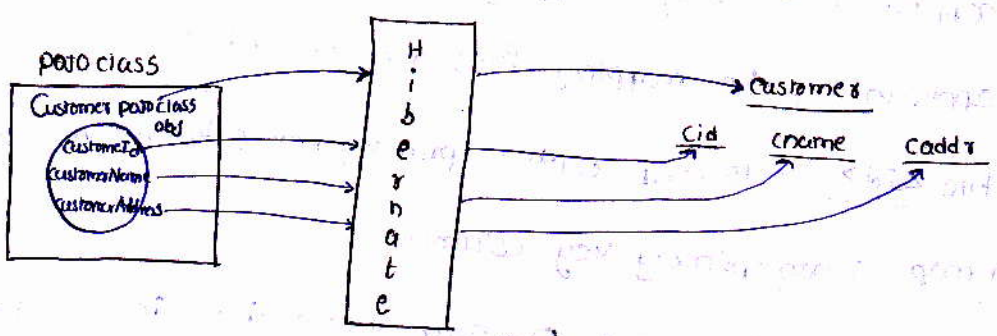


→ in Hibernate, mapping can be done in 2 ways.

① By creating an xml file

② By using Annotations

Pojo class object data coill be stored in Hibernate ORM tool. Hibernate store that data in DB.



→ In the above table Customer pojo class object is need to be stored in Customer table in Database. That pojo class object state will be stored in Customer table columns in DataBase.

→ In hibernate, a mapping file is identified with <anyname>.hbm.xml

→ Actually the extention required for an xml file is ".xml" but hibernate creates us suggested to put the extension of a mapping file as ".hbm.xml", to recognize it easyly in an application.

→ for in a mapping xml file a Java class and properties of a class are mapped to a DataBase table and Columns of a DataBase table.

08/02/2013

→ The Syntax of writing a mapping file in Hibernate is

<anyname>.hbm.xml

```
<!DOCTYPE ..... >
<hibernate-mapping>
    <class name="fully qualified java classname" table="database table name">
        <id name="variable name" column="primary key column name">
            <generator class="generator name"/>
        </id>
        <property name="variable name" column="column name"/>
        <property name="variable name" column="column name"/>
        .....
        .....
    </class>
</hibernate-mapping>
```

Imp Statements :-

① while constructing a mapping file in Hibernate, it is possible to write multiple Java classes mapping in a single mapping file.

② in Realtime applications, the mapping files are created on module basis

③ in a mapping file <Id> is to map with a primary key column and <property> is a tag to map a non-primary key column.

④ If there are multiple primary keys in a DataBase table then instead of <id> tag we use <Composite-id>

Configuration in Hibernate :-

→ Configuration in Hibernate is nothing but passing 3 types of information to the Hibernate.

① Connection properties
② Hibernate properties
③ Mapping files

→ Configuration in Hibernate can be done in only one way i.e. by constructing an xml file.

→ A Configuration file requires extension as '.xml'. but hibernate developers suggested to put the extension as ".cfg.xml", To recognize a Configuration of Hibernate in a project easyly.

→ we can't replace configuration from xml with annotation. It means in hibernate annotations are used only for avoiding mapping xml, but not configuration xml.

NOTE:- *) Hibernate is not a replacement of JDBC. Hibernate is actually introduced as an aulternative for EJB2 style entity beans.

*) Working with Hibernate is nothing but working with JDBC technology indirectly.

→ Hibernate mappings files are write in 2 ways

① Through xml file

② Through Annotations

→ Hibernate configuration files are write in only .xml file. ~~we~~ it is introduced hibernate 3.x also configuration files are write in properties files. This properties files used in hibernate 2.x In hibernate 3.x onwards we are using .xml file only.

→ mapping files are used in Realtime the below.

one mapping file contains one per each module in Realtime. A module can contains no.of Java files. i.e. in one mapping file we can write no.of Java files

Eg:-
```
<hibernate-mapping>
    <class name=....>
        <id.... />
        <property .... />
        <property .... />
    </class>
    <class name=....>
        :
    </class>
    <class name=....>
        :
    </class>
</hibernate-mapping>
```

→ In JDBC code we are using connection properties of database. If we change one DataBase to another DataBase we must open that JDBC code and modifying & last recompile. If we use Hibernate only If we change oracle DB to mySQL DB we open only configuration file & we no need to open Java code

first open connection operations are completed close the connect

11/02/2013

<any name>.cfg.xml

```
<!DOCTYPE ---------->
<hibernate-configuration>
  <session-factory>
    <!-- Connection Properties -->
    <property name="hibernate.connection.driver_class">fully qualified driver class name</property>
    <property name="hibernate.connection.url">driver url</property>
    <property name="hibernate.connection.username">username</property>
    <property name="hibernate.connection.password">password</property>

    <!-- hibernate properties -->
    <property name="hibernate.dialect">fully qualified dialect classname</property>
    <property name="hibernate.show_sql">true</property>

    <!-- mapping files -->
    <mapping resource="mapping file name"/>

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

Q:- How many Configuration files are required in an Hibernate application?

Ans:- It depends on the no.of DataBases used in an application.

→ The ratio between the Configuration files and DataBases is 1:1

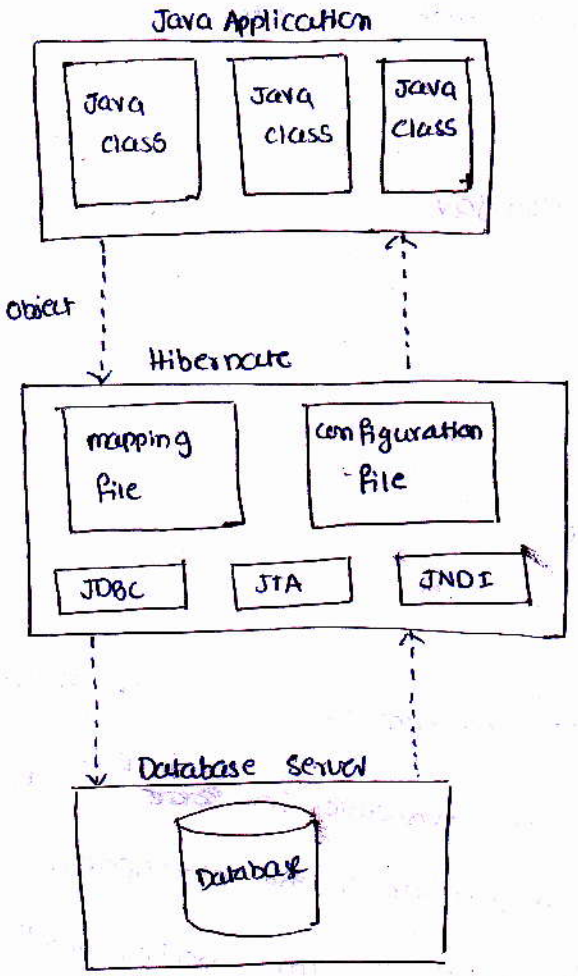
→ while working with Hibernate, we must ~~configure~~ create one configuration file for each DataBase and it doesnot matter weather the Databases are running in the Single Db server (or) in multiple DataBase servers.

→ In JDBC programmer has to write the SQL operations in Java code manually. It is the responsibility of programmer in JDBC.

But in Hibernate we no need to write SQL commands in Java code manually. It is the responsibility of hibernate. So, we use Dialect class in Configuration file. This Dialect class will read internal code of hibernate and generating database specific sql commands for each operation of hibernate.

→ ~~show~~ That generating SQL commands if we see then we use show_sql property. This default value is 'false'. if show_sql value is true then we see the SQL command. otherwise we can't see.

Hibernate Architecture :-



Hibernate Software :

Author : Gavin king

Type : open source framework

Version : 4.x (current version) (compatible with Jdk1.6+)
3.x (compatible with Jdk1.5+)
2.x (j2sdk1.3+)

download : www.sourceforge.net/projects/hibernate/files/hibernate3

Tutorial : www.tutorialspoint.com/hibernate
www.roseindia.net
www.java4s.com

Books : Hibernate in Action
Pro Hibernate 3 (Apress publication)

→ Because Hibernate is a framework, the slw is nothing but a set of jar files.

→ working with framework slws is nothing but using the predefined classes given in the jar files, to develop an application.

→ For a frame work, we have a main jar file and dependent jar files. similierly for Hibernate hibernate3.jar is the main jar file and the remaining jar files are dependent jar files of it.

→ The following are the list of jar files setting in to the CLASSPATH for executing a Java application with hibernate.

1. Hibernate3.jar
2. antlr-version.jar
3. commons-collections-version.jar
4. dom4j-version.jar
5. javassist-version.jar
6. jta-version.jar
7. slf4j-api-version.jar
8. hibernate-jpa-version.jar

12/02/2012

Understanding main and dependent jar files :-

→ If a Java application uses a class available in a jarfile then at the time of compilation we need to set that jar file in the classpath.

→ If a class of one jarfile depends on a class on another jarfile then we need to set the dependent jarfile also in the CLASSPATH

Eg:-

```
public class A
{
    void m1()
    {
        //code
    }
}
```

```
>javac A.java
>jar cvf a.jar A.class
```

```
public class B
{
    A aobj=new A();
    void m2()
    {
        aobj.m1();
    }
}
```

```
>javac B.java ↵
    Exception occurse
>set classpath=a.jar;.; ↵
>javac B.java ↵
>jar cvf b.jar B.class ↵
```

```
public class Main
{
    p s v m(String args[])
    {
        B bobj=new B();
        bobj.m2();
    }
}
```

```
>javac main.java ↵
    Exception occurse
>set classpath=b.jar;. ↵
>javac Main.java ↵
    Compiled successfully
```

```
>java Main ↵
    Exception occurse
>set classpath=%classpath%;a.jar; ↵
>java Main ↵
```

→ In the above example code, b.jar is the main jar file and a.jar is the dependent jar file for main.

→ while compiling the main class b.jar is set in the CLASSPATH and while executing the Main class along with b.jar, we also need a.jar in the CLASSPATH

Steps to write a client application in hibernate:-

Step-I:-

→ create an object of Configuration class

→ Configuration class is said to be the BootStraping class of hibernate.

→ In a Java application, hibernate environment begins by creating an object of Configuration class.

→ Configuration is a class of org.hibernate.cfg package

→ After creating an Object of Configuration class, we need to call configure() method by passing configuration file as a parameter.

→ In this step, along with configuration file mapping files are also loaded and internally an Xml parser is used and the data is read from both Configuration and mapping files and stored in Some Variables of Configuration class.

```
Configuration conf = new Configuration();
conf.configure("hibernate.cfg.xml");
```

Step-II:-

→ Built an high level object of hibernate called SessionFactory.

→ SessionFactory is an interface of org.hibernate

→ The implementation class of SessionFactory interface is SessionFactoryImpl.

→ Since hibernate is a framework, both interface and its implimentation class are provided by hibernate only.

→ when a SessionFactory Object is build then all Configuration and mapping data will be stored into SessionFactory object.

→ SessionFactory Object is called a heavy weight Object.

→ This is the only one heavy weight Object created in hibernate.

Syntax:-

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

Step-III:- Open a Session of Hibernate.

→ when a Session is opened then internally a Connection with the DataBase will be established.

→ Session is an interface of org.hibernate package. its implementation classname is SessionImpl.

→ in Hibernate sessions are produced by SessionFactory. it means SessionFactory is a factory is producing Sessions(Objects)

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

Step-IV:- Begin a Transaction with a DataBase

→ in hibernate all DataBase operations must be done within a Transaction, except "select" operation

→ A Transaction can be started using session Object.

→ Transaction is an Interface of org.hibernate package. Its implementation class is TransactionImpl.

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

Step-V:- perform operations on DataBase using methods of Session

Step-VI:- Commit a Transaction after complition of operations.

→ in Hibernate, all operations must be done with in a Transaction.

→ If any operation is failed then hibernate automatically rollbacks the transaction.

→
```
tx.commit();
```

Step-VII :-

→ Close a Session with a DataBase

→ when a Session is closed then internally a Connection with the DataBase will be closed.

```
Session.close();
```

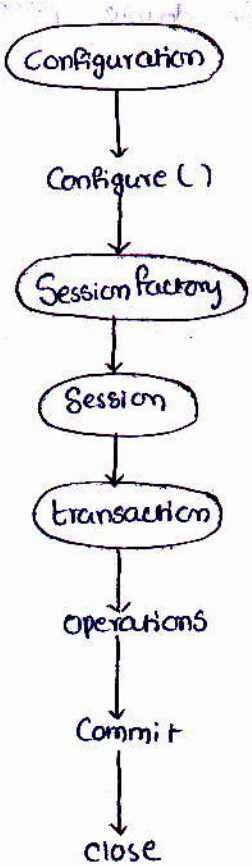
Step-VIII :-

→ Close SessionFactory of hibernate

```
factory.close();
```

While storing it search for getter methods for reading it search for setter methods.

Flow:-



The class should not have default constructor then it stores the value but not read the data. So, default constructor is manditory.

The following Example is for Storing a POJO class object called product in a database using Hibernate.

HbApp1

→ product.java → To store the data in POJO class object

product.hbm.xml

hibernate.cfg.xml

ProductInsert.java

```
// Product.java (POJO)
public class Product
{
    private int productId;
    private String productName;
    private double price;

    // default Constructor
    public Product()
    {
    }
    // Parameterized Constructor
    public Product(int productId, String productName, double price)
    {
        this.productId = productId;
        this.productName = productName;
        this.price = price;
    }
    // setters and getters
    public void setProductId(int productId)
    {
        this.productId = productId;
    }
    public int getProductId()
    {
        return productId;
    }
    public void setProductName(String productName)
    {
        this.productName = productName;
    }
    public String getProductName()
    {
        return productName;
    }
    public void setPrice(double price)
    {
        this.price = price;
    }
```

```
public double getPrice()
 {
    return price;
 }
}
```

→ In a POJO class of Hibernate application while storing an object in a DataBase, Hibernate checks for getter methods in a POJO class and while reading an object from a DataBase, Hibernate checks for a default constructor and also setter methods in a POJO class. So, to perform either a store or read operations, in a POJO class of Hibernate we should include a public default constructor and also setter and getter methods of properties

→ If we donot create any constructor in the Java class then during compilation a public default constructor will be automatically added for that java class.

```
<!-- product.hbm.xml -->
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
    <class name="Product" table="product-table">
        <id name="productId" column="pid">
            <generator class="assigned"/>
        </id>
        <property name="productName" column="pname"/>
        <property name="price" column="price"/>
    </class>
</hibernate-mapping>
```

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

<hibernate-configuration>
    <session-factory>
        <!-- Connection properties -->
        <property name="hibernate.connection.driver_class">oracle.jdbc.OracleDriver</property>
        <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:orcl</property>
```

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

<!-- hibernate properties -->
<property name="hibernate.dialect">org.hibernate.dialect.OracleDialect</property>
<property name="hibernate.show_sql">true</property>
<!-- mapping file properties -->
<mapping resource="product.hbm.xml"/>
</session-factory>
</hibernate-configuration>
```

```
// ProductInsert.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public ProductInsert
{
public static void main(String[] args)
{
// Step-I
Configuration conf = new Configuration();
conf.configure("hibernate.cfg.xml");

// Step-II
SessionFactory factory = conf.buildSessionFactory()

// Step-III
Session session = factory.openSession();
// Create a Product object
Product p=new Product();
// Set the values
p.setProductId(111);
p.setProductName("sony");
p.setPrice(7000);
// Step-IV
Transaction tx = session.beginTransaction()
// Step-V
```

```
        session.save(p);
        // Step-VI
        tx.commit();
        // Step-VII
        session.close();
        System.out.println("A Product object is stored in Database");
        // Step-VIII
        factory.close();
    }
}
```

→ Product.java is a Pojo class, so it can be compiled by without any CLASSPATH setting.

```
HibernateApp1> javac Product.java
```

→ ProductInsert.java used HibernateAPI. so, to compile this Java file, we need to set main jar file (Hibernate3.jar) of Hibernate in the CLASSPATH

```
HibernateApp1> set classpath = D:\hibernate-3.6.5 Finaldistribution\hibernate3.jar;.
HibernateApp1> Javac ProductInsert.java  successfully compiled
```

→ To execute ProductInsert application, we need to set main Jar file and dependent jar files & ojdbc14.jar in the CLASSPATH.

```
HibernateApp1> set classpath = D:\hibernate-Final distribution-3.6.5\hibernate3.jar; D:\hibernate
Finaldistribution-3.6.5\lib\required\antlr-2.7.6.jar; D:\Final distribution-3.6.5\lib\required\
commons-collections-3.1.jar; D:\H-d\required\lib\dom4j-1.6.1.jar; D:\H-d\r\lib\javassist-3.12.0
GA.jar; D:\H-d\required\lib\jta-1.1.jar; D:\H-d\lib\required\slf4j-api-1.6.1jar; D:\H-d\
lib\JPA\hibernate-jpa-2.0-api-1.0.0.Final.jar; D:\Oracle\ora92\jdbc\lib\ojdbc14.jar;.
```

→ Before we run this ProductInsert client application, we need to create a table called Product_table in the DataBase.

```
Create table product_table (pid number(5) primary key, pname varchar(15), price
                                                        number(9,2));
Table Created
```

→ HibernateApp1> java ProductInsert

↳ This insert query is created by internally Dialect class. if we see this command show_sql is recommended

Hibernate : insert into Product_table (pname, price, pid) values (?, ?, ?)

↳ PreparedStatement

A Product Object is stored in DataBase.

NOTE:- Hibernate internally uses Prepared Statement of Jdbc, to perform DataBase operations.

NOTE:-

① While creating a mapping file in Hibernate, if a POJO class variable name and Database table column name are matched then column attribute is optional.

② If POJO class name and Table name are matched then table attribute is optional.

hibernate.hbm2ddl.auto property :- 15/02/2013

→ This is an Hibernate property, used for performing DDL operations on a database by hibernate.

→ while working with hibernate, hibernate has the ability to automatically execute/perform DDL operations, if necessary

→ The 4 values of this property are

① validate
② create
③ update
④ create-drop

→ The default value of this property is "validate".

(i) if the value of this property is validate then hibernate only checks for the table and for columns in the table.

(ii) if either table (or) columns does not exist then hibernate throws an Exception.

→ if hibernate.hbm2ddl.auto property is "create"

(i) then hibernate checks for the table in Database. if exist then it will be dropped and then a new table will be created by hibernate.

(ii) if table doesn't exist then directly a new table is created.

(iii) the Create value is suitable when executing an application for first time. while executing for next time, we will loose the previous data added to the table.

→ if hibernate.hbm2ddl.auto property is update then hibernate checks for the table and for columns. if table exist then it checks for columns. if a column doesnot exist then it will be added to the table.

(i) Incase of update value, hibernate doesnot drops the existing table and doesnot drop the existing columns. If a table doesnot exist then hibernate creates a new table in the database.

(ii) Comparatively update is the best value then the remaining values.

→ if hibernate.hbm2ddl.auto property is create-drop then

(i) hibernate creates the table when an application execution starts, and drops the table when SessionFactory is closed explicitly in an Application.

(ii) if SessionFactory is not closed explicitly, i.e. if we donot write factory.close() then hibernate will not drop the table from the DataBase.

→ This hibernate.hbm2ddl.auto property can be added in a hibernate configuration file like the following

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

Q:- Why we prefere to use Configuration Filename as "hibernate.cfg.xml" ?

Ans:- If a Configuration filename is hibernate.cfg.xml then it is not mandatory to pass the filename as a parameter for Configure() method.

→ By default Configure() loads hibernate.cfg.xml file

→ If the Configuration Filename is some other name then ~~we have to~~ It must be passed as parameter for Configure() method.

→ If filename is ~~then~~ hibernate.cfg.xml then conf.configure(); ✓
conf.configure("hibernate.cfg.xml"); ✓

→ If filename is sathya.cfg.xml then conf.configure(); ✗
conf.configure("sathya.cfg.xml"); ✓

Q:- What is the difference b/w hibernate properties file and hibernate configuration file ?

Ans:- In hibernate properties file, we can write Connection properties and hibernate properties. But mapping files can't be added to a properties file. But if it is a configuration xml file then we can add hibernate properties, connection properties and also mapping files.

→ Hibernate configuration xml is introduced in hibernate 3.x
in hibernate 2.x, instead of xml file a properties file is used.

→ If properties file is used then the filename should be "hibernate.properties" and in a client application it will be loaded automatically, whenever an Object of Configuration class is created.

```
Configuration conf = new Configuration();
```

It means we no need to call configure() method

→ Incase of properties files, hibernate mapping files are added to the Configuration object by calling a method conf.add(-)

```
Conf.addFile("product.hbm.xml");
```

Q:- If hibernate properties file and a Configuration xml file, both are created then which is considered by hibernate ?

Ans:- If configure() method is not called then hibernate uses properties file and if configure() method is called then Configuration xml file.

Q:- can we create a hibernate application, by without creating hibernate properties file or configuration xml file or not ?

Ans:- Yes,
we need to add the connection properties, hibernate properties and mapping files directly to the Configuration object in the source code.

→ The draw back of creating an hibernate application without a properties file

or a Configuration xml file is, if any changes are required in the properties then we need to make the changes in Source code, then we need to recompile the Source code.

If it is a serverside application then we need to reload the application into the server, and sometimes we need to restart the server.

for Eg:-

```
Configuration conf = new Configuration();
conf.setProperty("hibernate.connection.driver_class", "oracle.jdbc.OracleDriver");
conf.setProperty(_, _) url
                 (      ) username
                 (      ) password
                 (      ) → dialect
                 (      ) → show_sql
                 (      ) → hbm2ddl.auto
conf.addFile("product.hbm.xml");
SessionFactory factory = conf.buildSessionFactory();
```

16/02/2012

(*) load operation (Select Operation) :-

→ load operation is nothing but reading an object from a DataBase into a Java app using Hibernate.

→ In order to perform a load operation / select operation a Java programmer has to call any one of the following two methods given by Session Interface

① get (Class clazz, Serializable id)

② load (Class clazz, Serializable id)

→ While performing Select operation using Hibernate, Hibernate need a Class object of a Pojo class and Id for identifying one Object among multiple objects saved in a DataBase

→ While loading, Hibernate creates internally an Object of a pojo class for storing the selected data from a DataBase. For creating an Object of the pojo class, hibernate needs Class Object of the pojo class.

→ Hibernate internally calls newInstance() for creating an Object of a pojo class, using Class object.

→ When a newInstance() is called then it will check for public default Constructor in a pojo class. If exist then an Object of the pojo class is created. otherwise an Exception will be raised. So, while creating a pojo class in hibernate application, a public default constructor is manditory in the class. Other coise hibernate fails to load / select the objects from the DataBase

→ While loading / selecting an Object from a DataBase then hibernate performs the following ~~methods~~ operations internally

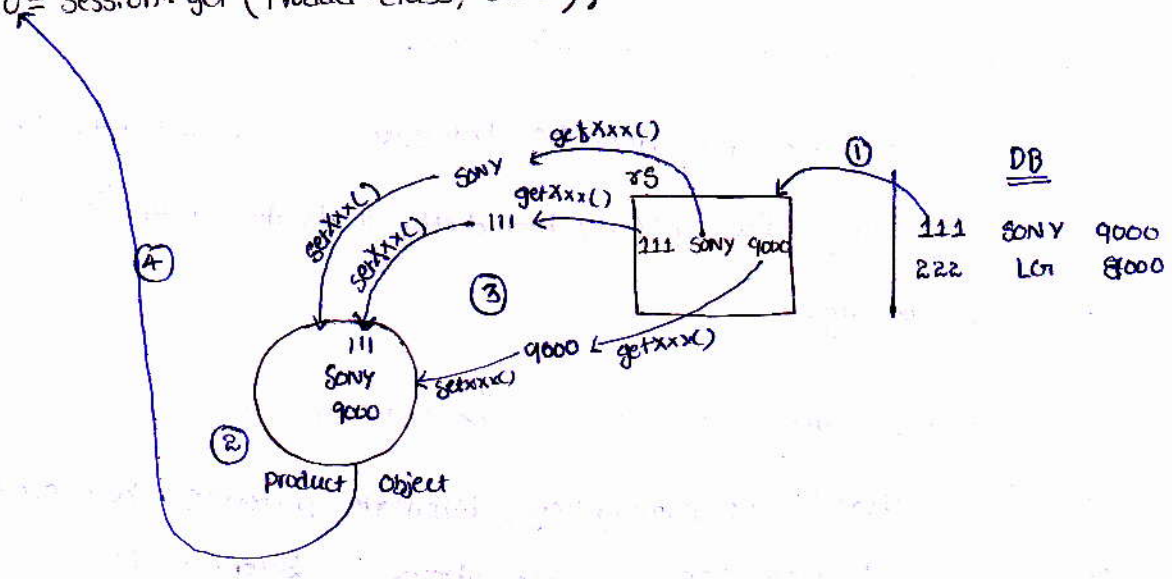
① Hibernate first reads data from a DataBase and stores it in a ResultSet object.

② Hibernate creates a pojo class object by calling newInstance() method

③ Hibernate reads the reco values from ResultSet, by calling getXxx() methods and then stores the values into ~~pojo class object setXxx()~~ pojo class Object by calling setXxx() method of pojo class.

④ Finally, Hibernate returns the POJO class object to Java appn.

Eg:- Object o = session.get(Product.class, 111);



→ When get(-,-) method is used for loading an object from database. Hibernate 1st verifies weather the given "id" is exist in the DataBase or not. If exist then immidietly reads it from the DataBase and prepares internally POJO class object (Product class obj) and finally returns that POJO class object back to our Java appn

→ If the given id is not exist in the DataBase then hibernate skip the internal process and finally returns null value back to the Java appn

→ incase of load(-,-) method, while loading an object, hibernate checks for given id and it is exist in the DataBase then Hibernate prepares a pojo class object with empty data and returns that object back to a Java program.

Here the object returned by hibernate is not a real object. It means the object does not contain the data of a DataBase. so, we call it as a "proxy" object.

→ When a Java program starts accessing that object then internally hibernate will reads the data from DataBase and puts it into the POJO class object

This mechanism in hibernate is called "Lazy loading".

→ if the given Id does not exist in the DataBase then hibernate throws ObjectNotFoundException, incase of load method.

Q:- What is the difference b/w a load(-,-) and get(-,-)

Ans:- ① If the given Id does not exist in the DataBase then load(-,-) method throws an Exception, but get(-,-) returns null.

② load(-,-) loads the data from the DataBase ondemand (at the time of accessing). It is Lazyloading. But get(-,-) immidietly loads the data from a DataBase. so, It is early loading.

Q:- IS it manditory to have a primerykey in a DataBase table, while working with in Hibernate?

Ans:- NO, If there is no primary key, hibernate performs save operation without Errors, but in load operation hibernate throws an Exception if the given id is found for more than once.

NOTE:- In a mapping file, If the table doesnot have a primary key then any colemn can be mapped as "id". It means "id" in a mappingfile is mandatory.

// ProductSelect.java

```
import org.hibernate.*;
import org.hibernate.cfg.*;
public class ProductSelect
{
  p s v m(String[] args)
  {
    Configuration cof=new Configuration();
    Conf.configure();
  SessionFactory factory = conf.buildSessionFactory();
  Session session = factory.openSession();
  Object o= session.get(Product.class, 333);
    if(o==null)
    {
      S.o.p("A Product with id 333 is not found");
    }
  else
    {
      Product p=(Product)o;
    S.o.p("A Product with id 333 is found");
    S.o.p("product id ="+p.getProductId());
    S.o.p("Product name ="+p.getProductName());
    S.o.p("product price ="+p.getPrice());
    }
    Session.close();
    factory.close()
  } //main(-)
} //class
```

```
>javac ProductSelect.java
>java ProductSelect
Hibernate: select product0_.pid ....
```

A product with id 333 is found

Product id = 333
Product name = LG
Product price = 6000.0

Q:- How to Identified weather an Object is early (or) Lazy loaded from the Databases?

Ans:- In Early loading, Hibernate generates a Select operation, immidietly when an Object is Select (or) loaded. It means even though we are not accessing (reading the data) that Object in the code but it is loaded from (selected from) the DataBase.

In Lazy loading, Hibernate generates a Select operation, only when accessing that Object in the code. otherwise a select operation is not generated by the hibernate.

*** Session - level cache (level 1 cache)

→ When a session is opened in Hibernate then internally a (buffer) Cache is opened along with the Session.

→ During the session of hibernate, the operations done on the objects will be stored in the cache (buffer) after Session. It means if we perform an Insert (or) Update (or) delete (or) Select then the result of this operations will be stored by hibernate internally in the cache of that Session.

→ with the help of the cache hibernate reduces the round trips between a Java appn and a DataBase. So that the performance of an Application will be increased.

→ For example, if we load an Object from a DataBase then first hibernate verifies weather this Object is there in the Session cache or not. If yes, then it reads the object from cache, instead of from DataBase. So, the trips between Java appn and database will be reduced.

→ In hibernate, for every Session opened (or) generated by SessionFactory contains its own Cache.

→ As a hibernate programmer, we dont have any settings, either to enable (or) create the cache

or disable (remove) the cache.

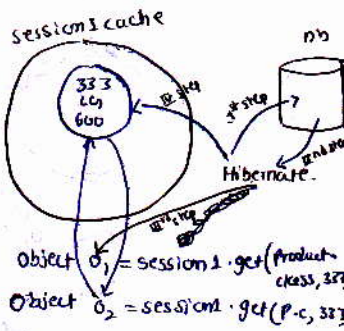
→ This session level cache will be automatically created when a session is opened and automatically closed when a session is closed.

19/02/2013

→ As a programmer, if you want to remove an object from a Session Cache, by calling evict() method, then we need to pass an object as a parameter for the evict() method.

→ while calling clear() method for clearing objects from a session cache, no parameter is required for the method, because it removes all objects from cache, not a perticular object.

⊛ The following code is for loading same object from the DataBase for twise, whose product ID is "333".

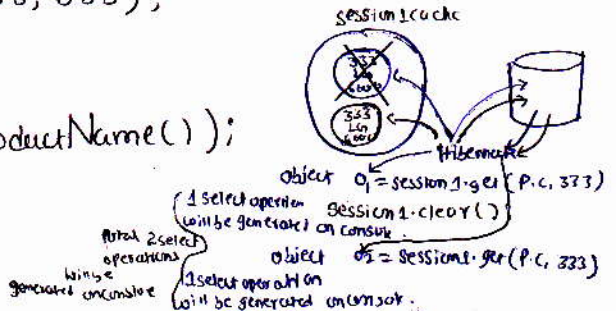



```
Object o1 = session1.get(Product.class, 333);
Product P1 = (Product) o1;
S.o.p("Product Name=" + P1.getProductName());
S.o.p("----------");
Object o2 = session1.get(Product.class, 333);
Product P2 = (Product) o2;
S.o.p("Price=" + P2.getPrice());
```

In the above code, an object is loaded within the session for 2 times, but it is loaded for once from the DataBase and next time it is loaded from the Session cache. so, only one select operation will be displayed on console by hibernate.

→ In the middle, if we add session1.clear() then an object will be loaded from the DataBase only for 2nd time. so, that two select operations will be showed on the console by hibernate.

```
Object o1 = session1.get(Product.class, 333);
Product P1 = (Product) o1;
S.o.p("Product Name=" + P1.getProductName());
S.o.p("----------");
*** session1.clear();
```

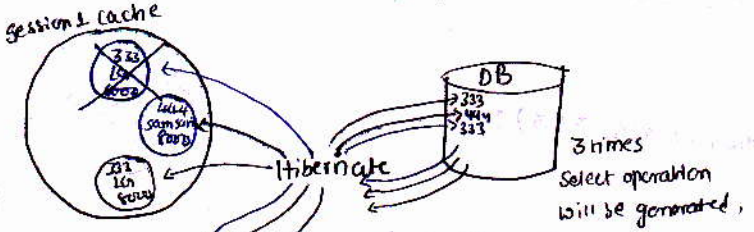


```
Object   o2 = session1.get(Product.class, 333);
Product  P2 = (Product) o2;
S.o.p("price =" + P2.getPrice());
```

→ In the middle if we want to remove any perticuler object we use session.evit( )method. in this evict( ) we pass one object as parameter. That perticuler object will be removed from the session cache.

Eg:

```
Object   o1 = session.get(Product.class, 333);
Product  P1 = (Product) o1;
S.o.P("Product name =" + P1.getProductName());
Object   o2 = session.get(Product.class, 444);
Product  P2 = (Product) o2;
Object   o3 = session.get(Product.class, 555);
S.o.P("price =" + P2.getPrice());
Product  P1 = (Product) o1;
Product  P2 = (Product) o2;
```




```
Object o1 = session1.get(Product.class, 333);
Object o2 = session1.get(Product.class, 444)
session1.evict(P1);
Object o3 = session1.get(Product.class, 333);
```

```
Object   o1 = session1.get(Product.class, 333);
Product  P1 = (Product) o1;
S.o.P("Product name =" + P1.getProductName());
S.o.P("----------")
Object   o2 = session1.get(Product.class, 444)
Product  P2 = (Product) o2;
S.o.P("price =" + P2.getPrice());
S.o.P("----------");
session1.evict(P1);
Object   o3 = session1.get(Product.class, 333)
Product  P3 = (Product) o3;
S.o.P("price =" + P3.getPrice());
```

Update Operation :-

→ In hibernate, an object which is already stored in a DataBase can be updated in two approaches.

1) we can Update an Object (means new), by without loading it from database.

2) we can update an object by loading it from database.

Approach - I :-

→ If we want to update an Object, by without loading it from a DataBase then the following steps are need to be follow.

(i) create a new Object of POJO class.

(ii) set the previous values to the properties, which we donot want to update.

(iii) set new values to the properties which we want to update.

(iv) Call update( ) of a Session with in a Transaction

Eg:- If we want to update a price of a product whose "id" is 333 then the following code is required.

```
product p=new Product();
p.setProductId(333);
p.setProductName("LG");
p.setPrice(4000);
Transaction tx = session.beginTransaction();
session.update(p);
tx.commit();
```

→ when the above code is executed then in the DataBase the price of the product is updated to "4000", but the product name becomes the same name as "LG".

Product_table

Pid	Pname	price
333	LG	~~6000~~ → old 4000 → new

→ The drawback of this approach is, we need to set both the values which we donot want to update and the value which we want to update. It is a burden on the programmer and also, if a property is not set then its default value will be updated in the DataBase.

Eg:-
```
Product p = new Product();
p.setProductId(333);
p.setPrice(8000);
Transaction tx = session.beginTransaction();
session.update(p);
tx.commit();
```

Product_table

Pid	Pname	price
333	null	~~6000~~ → old 8000 → new

```
Product p = new Product();
p.setProductId(333);
p.setProductName("SAMSUNG");
Transaction tx = session.beginTransaction();
Session.update(p);
tx.commit();
```

Product_table

Pid	Pname	price
333	~~LG~~ old SAMSUNG new	0 → default value.

Approach-II:-

→ In this approach, 1st we need to load an Object, which we want to update from the DataBase and then we can set the new values to the properties which we want to update, with in a Transaction.

→ In this approach, we no need to set previous values to the properties which we donot want to update.

→ In this approach, we no need to call update() method of a session

Eg:- If we want to update the price of a product Id "333" then the following code is required.

```
Object O = session.get(Product.class, 333);
Product p = (Product)O;
Transaction tx = Session.beginTransaction();
p.setPrice(6000);
tx.commit();
```

in the table

Product_table

Pid	Pname	price
333	LG	~~6000~~ → old 8000 → new

→ when updating an object while loading it from a database, the modifications done on the data of an object will be stored 1st in cache of a session. later, when a Transaction is committed then the changes are transfered the chache memory to the DataBase. This operation in hibernate is called as "flushing"

→ Hibernate performs flushing, only when if changes existing between the state of an object in cache and the database.

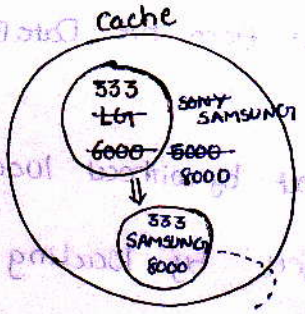
→ If there are no changes in the state of an Object stored in cache and stored in Database then hibernate doesnot apply any flush operation when a Transaction is Committed.

→ Flushing is a mechanism of transfering the state changes from a cache memory Object to a DataBase.

The following client application of Java is for loading and updating an object, whose productId is 333.

```
// ProductUpdate.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public class ProductUpdate
{
   p s v m(String[] args)
   {
      Configuration conf = new Configuration();
      conf.configure();
   SessionFactory factory = conf.buildSessionFactory();
   Session session = factory.openSession();
   Object o = session.get(Product.class, 333);
                                  ↳ Primitive data type (or) wrapper type serializes Id
   Product p = (Product)o;
   Transaction tx = session.beginTransaction();
      p.setPrice(5000);
      p.setPrice(8000);
```

```
p.setProductName("SONY");
p.setProductName("SAMSUNG");
tx.commit();
session.close();
factory.close();
}
}
```



flushing

Product_table

Pid	PNAME	price
~~333~~	~~LOT~~	~~6000~~
	SAMSUNG	8000

*) Delete Operation:-

→ We can Delete an Object from the DataBase using Hibernate in the following two ways.

① We can delete an Object by without loading it from the DataBase.

② We can delete an Object by loading it from the DataBase.

Approach - 1 :-

→ In this approach, we can create an Object and we can assign the Id to be deleted from the DataBase and then we can call delete() method to delete an Object from the DataBase.

→ While deleting an Object in this approach we no need to assign any other values an Object except Id. Because Hibernate deletes an Object from database by checking for the Id.

→ For example, If we want to delete a product whose id is 333 from the DataBase, by without loading it from the DataBase then the following code is required.

```
//ProductDelete.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public class ProductDelete
{
  p s v m (String[] args)
  {
Configuration conf=new Configuration();
  conf.Configure();
SessionFactory factory = conf.buildSessionFactory();
  Session session1 = factory.openSession();
  Product p=new Product();
   p.setProductId (333);
Transaction tx = session1.beginTransaction();
  Session1.delete (p);
```

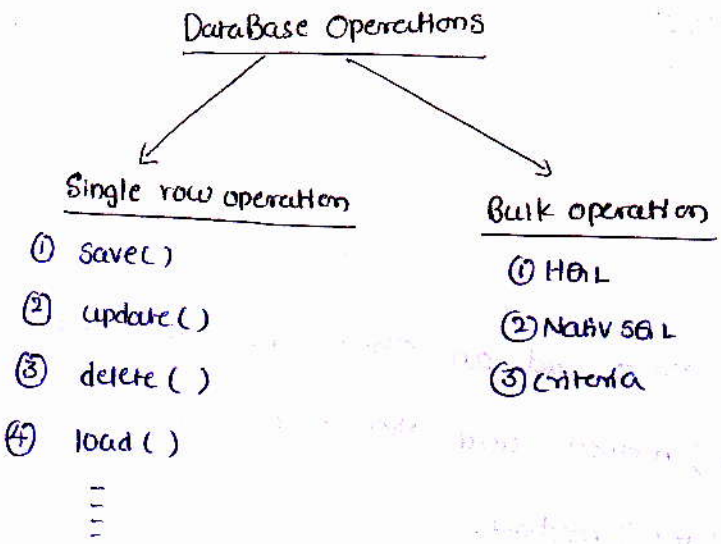
```
        tx.commit();
        Session1.close();
        factory.close();
    }
}
```

Approach-2:-

→ In this approach, we need to load an Object from the DataBase by calling get(-,-) or load(-,-) method and then we need to delete the loaded Object by calling delete() method.

→ when compared with this approach 2, in approach 1, it is enough to call delete(-) method in approach 1. So, Comparatively approach 1 is better than approach-2.

```
Object O = session.get(Product.class, 333);
Transaction tx = session.beginTransaction();
    Session.delete(O);
tx.commit();
```



Versioning an Object :-

→ In Hibernate, we have a feature that hibernate stores version of an object in a DataBase table column, to identify how many no.of times an object is updated, since from its creation.

→ When an object is newly inserted into the DataBase then it version number is inserted as '0'. when its data is updated then Hibernate automatically increments its version by '1'. and stores that version number in DataBase.

→ By reading the version of an object, we can find that how many no.of times the object is updated so far..

→ To get Versioning feature for a POJO class object then the following two changes are required in an application.

① in POJO class, we need to create an Integer variable and we need to generate appropriate setters and getter methods.

② In mapping file, we need to Configure a tag called <version> with Version variable and Version column used in a POJO class and DataBase table respectively.

→ This <version> tag must be configured immediately after <id> tag in a mapping file.

→ To apply this versioning feature of hibernate to the product objects then we need to make the changes in Product.java and product.hbm.xml like the following

```
// Product.java
Public class Product
{
   Private int productId;
   private String ProductName;
   private double price;
** private int v;
   ⟶ setters
   ⟶ getters
}
```

```
<!-- product.hbm.xml -->
<hibernate-mapping>
  <class name="Product" table="product_table">
    <id name="productId" column="pid">
      <generator class="assigned" />
    </id>
**<version name="v" column="ver_id" />
  <property name="productName" column="pname"
            length="10" />
  <property name="price" column="price" />
  </class>
</hibernate-mapping>
```

→ when a new product object is inserted in a DataBase then its version number will be saved as "0". If a product is loaded and updated then its version will be incremented by "1" automatically by hibernate.

21/02/2013

product_table

Pid	pname	price	ver_id	
111	SONY	~~5000~~ 9000	0 1	→ when a new product is inserted its version number is 0 → whenever a product is updated its version number will be incremented by 1

Timestamp feature of Hibernate :-

→ in Hibernate, it is possible to know, on what date at what time an Object is updated using the Timestamp feature of Hibernate.

→ Hibernate ~~auto~~ automatically ~~*~~ stores (or) records the System date and time into the DataBase, whenever an Object is inserted (or) whenever an object is updated.

→ with this Timestamp feature of Hibernate, it is only possible to know when an Object is updated in the DataBase. but it is not possible to know how many no.of times an Object is updated.

→ It is possible to know how many no.of times ~~the~~ an Object is updated using Versioning feature of Hibernate. but it is not possible to apply both

Versioning and Timestamp at a time in an appn.

→ To apply Timestamp features of Hibernate then the following changes are required in an Application.

① in POJO class, we need to create a property of type Timestamp and generate setter and getter methods.

② in mapping file of Hibernate, configure <timestamp> immediately after <id> tag.

→ In POJO class, while creating a property of Timestamp we need to import java.sql.*; package, because Timestamp is a class of java.sql package.

Example:-

→ To apply this Timestamp feature for product example, we need to make the changes in product.java and product.hbm.xml like the following

```
// Product.java (POJO)
import java.sql.*;
public class Product
{
  private int productId;
  private String productName;
  private double price;
  private Timestamp ts;
  → setters
  → getters
}
```

```
<!-- Product.hbm.xml -->
<hibernate-mapping>
  <class name="Product" table="product_table">
    <id name="productId" column="pid">
      <generator class="assigned" />
    </id>
    <timestamp name="ts" column="tstamp" />
    <property name="productName" column="pname" length="10" />
```

```
<property name="price" column="price" />
</class>
</hibernate-mapping>
```

→ if Hibernate is creating a DataBase table then Hibernate uses the Datatype of Timestamp column as Date. If it is Datetype then it can store only on what date an Object is updated. but it can't store on what time an Object is updated.

→ If we create DataBase table manually then we can use Datatype of a Timestamp column as timestamp. so that hibernate stores both Date and time in the DataBase whenever an object is created or updated.

```
SQL> create table product_table( pid number(5) primary key, pname varchar2(10),
        price number(9,2), tstamp timestamp);
Table created.
```

→ If we run ProductInsert Client application then Hibernate stores a Product object and also the system date and time like the following in DataBase.

```
SQL> select * from product_table;
```

pid	pname	price	Tstamp
111	SONY	8000	21-Feb-13 08.00.40.35900 AM

Steps to develop an Hibernate application Using MyEclipse 8.x.

Step-I :- Create a table called Employee in the DataBase like the following

```
SQL> create table employee (empno number(5) primary key, ename varchar2(20),
        Sal number(8), Deptno number(3)) ;

Table created.
```

Step-II :- Start MyEclipse IDE

Start → programs → MyEclipse → MyEclipse 8.X stable → Enter

workspace [C:\work1] → OK

Step-III :- Click on File menu → New → Javaproject → projectname : [App2]

→ Finish

Step-IV :- By default a PackageExplorer view is displayed at leftside. In this view "bin" folder is invisible. so change to Navigator

↳ stores .class file

(in src folder Java source program will saved)

Click on Windoco menu → showview → Navigator

Step-V :- Right click on project name → MyEclipse → Add Hibernate capabilities

→ ⊙ Hibernate 3.2 → next → next → enter the following details.

DB Driver : [ ]

Connect URL : [jdbc:oracle:thin:@localhost:1521:orcl]

Driver class : [oracle.jdbc.OracleDriver]

Username : [scott]

Password : [tiger]

Dialect : [Oracle 9/10g] → next → deselect ☐ create session Factory class

→ Finish

Step-VI :- Click on Windoco menu → Open perspective → MyEclipse DataBase Explorer

→ at leftside in a DB Browser view, click on ▼ ViewMenu button → New

→ enter the following Details.

Driver Template : Oracle (Thin Driver)

Driver name : MyDriver → alias name of classname.

Connection URL : jdbc:oracle:thin:@localhost:1521:orcl

Username : scott

Password : tiger

Driver JARS

D:\oracle\ora92\jdbc\lib\ojdbc14.jar Add JARS

Driver classname : oracle.jdbc.driver.OracleDriver

TestDriver → next → Finish

Step-VII :- in DB Browser view at left side, Right click on MyDriver → open a connection

→ Enter password : tiger → ok

Step-VIII :- Expand MyDriver → Expand Connected to MyDriver → Expand SCOTT

→ Expand TABLE → select Employee table → Right click on Table name

and select Hibernate Reverse Engineering → do the following

Java src folder : /App2/src Browse → ☑ Create POJO DB Table mapping information

⊙
○
☑

→ ☑ Java Data Object (POJO <> DB Table) → next → ID Generator: assigned

☐ create abstract class

→ next → Finish

Step-IX :- Right click on project name → new → class → name : StudentInsert (client appn name)

→ Finish

```
import org.hibernate.*;
import org.hibernate.cfg.*;
public class StudentInsert
 {
   public static void main(String[] args)
   {
```

```
Configuration conf = new Configuration();
conf.configure();
SessionFactory factory = conf.buildSessionFactory();
Session session = factory.openSession();
Employee e = new Employee();
e.setEmpno (1121);
e.setEname ("Manoj");
e.setSal (6000);
e.setDeptno (10);
Transaction tx = session.beginTransaction();
session.save(e);
tx.commit();
System.out.println("Employee saved in DataBase");
session.close();
factory.close();
}
}
```

Step-I:- Right click on project name → properties → Java Build path (leftside) → Libraries (rightside)

→ Add External JARs Button → select ojdbc14.jar → open → ok

Step-II:- Right click on StudentInsert.java → Run as → Java application

22-02-2013

Different States of an Object in hibernate : (life cycle of an object)

→ A POJO class object of Hibernate is going to have the following 3 states in an Application.

① Transient state
② Persistent state
③ Detached state

① Transient State :-

→ When a new object is created for a POJO class then that object will be in a Transient state.

→ If we create a POJO class object by assigning "null" value then also we say that an object is in Transient State.

→ A Transient state object is an object which is not yet assossiated with a Session of Hibernate.

→ A Transient state object is not representing a row of an underline DataBase

→ If we make any changes on the Data of a Transient object then this changes are not effected on the DataBase.

Eg:-
```
Product p=new Product();
p.setProductId(111);
p.setProductName("nani");
p.setProductPrice(3000);
```

→ here "p" is in Transient state.

```
Product p=null;
```

→ here "p" is in Transient state.

② Persistent State :-

→ when a Save operation is done on a Transient state object (or) when a load operation is done on a Transient state object then the Object will be entered into persistent state.

→ When an Object is entered into persistent State then it is assossiated with Session of Hibernate.

→ If any modifications are done on an Object in a persistent state then the changes are effected on the underlaying DataBase also.

Eg(1):-

```
Product p=new Product();
p.setProductId(111);
p.setProductName("ABC");
p.setPrice(9000);
```

→ here 'p' is in Transient state

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

→ here 'p' is in Persistent state

Eg(2):-

```
Product p=null;
```

→ here 'p' is in Transient state

```
Object O=session.get(Product.class,111);
P=(Product)O;
```

→ here "P" is in Persistent state.

(3) Detached State:-

→ When a session is closed or when a session is cleared or when an Object is evict from the session then an Object is Converted from persistent state into Detached State.

→ When an Object looses its attachment with a session then it will be entered into Detached state.

→ When an Object is in Detached state and if any modifications are done on that Object then the changes are not effected on the DataBase.

→ The commanality in Transient state and Detached state is, in both cases the changes are not effected on a DataBase.

→ Detached State is a state when an Object is comming out of a session and Transient State is a state before entering into a session.

Eg:-(1)

```
Product p=new Product();
p.setProductId(111);
p.setProductName("ABC");
p.setPrice(9000);
```

→ here 'p' is in Transient state

Eg:-(2)

```
Product p=null;
```

→ here "p" is in transient state

```
Object O=session.get(Product.class,111);
P=(Product)O;
```

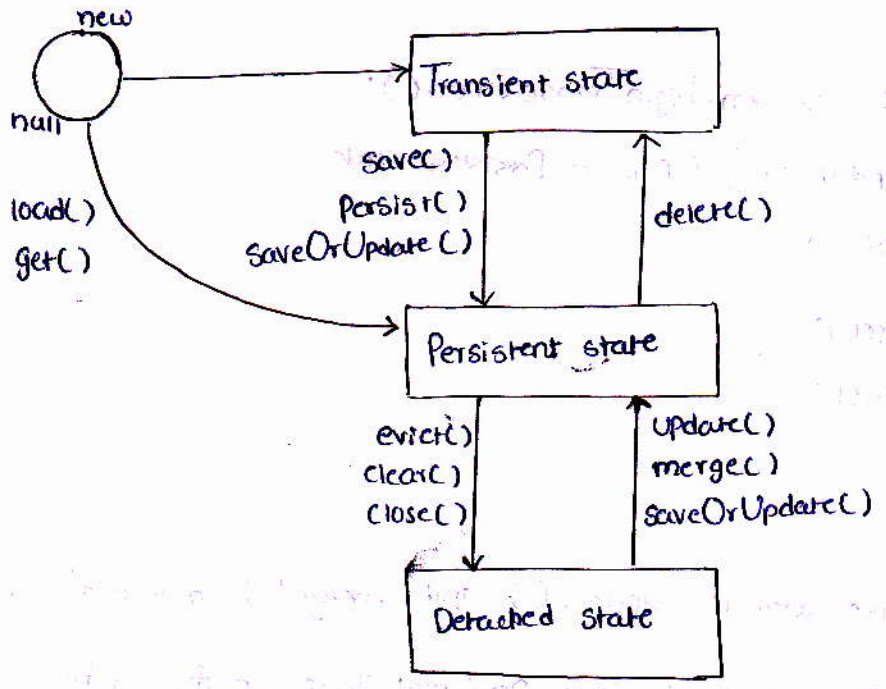
→ here "p" is in persistent state

```
session.clear();
```

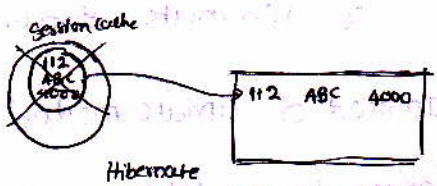
→ here "p" is in Detached state.

```
Transaction tx = session.beginTransaction();
    Session.save(p);
→ here 'p' is in persistent state
    tx.commit();
  Session.close();
→ here 'p' is in detached state.
```

23-02-2015



NOTE:- Transient State is an Object which is not yet attached to a session of hibernate and Persistent State is an Object attached to Session of hibernate and Detached State is an Object which is came out of a Session.




```
                    //StatesClient.java
                    import org.hibernate.*;
                    import org.hibernate.cfg.*;
                    Class StatesClient
                    {
                      p s v m(String args[])
                      {
SessionFactory      factory = new Configuration().configure().buildSessionFactory();
    Session         session = factory.openSession();
    Product         p=new Product();    //p is in transient state
                    P.setProductId(112);
                    P.setProductName("ABC");
```

```
        p.setPrice (4000);

Transaction   tx = session.beginTransaction();
        session.save(P); // P is in Persistent state
        tx.commit();

      Session.clear();   // P is in Detatche state
      P.setPrice (3000);

Transaction   tx2 = session.beginTranscaction();
        session.update(P); // P is in Persistent state
        tx2.commit();

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

*** Q:- What is the difference between update() and merge() method of hibernate?

Ans:- While Converting a detached Object into persistent state, in the cache of the session if already an Object is saved with the same Id (identifier) value then update() method fails to convert an Object from detached state into persistent.

So, Hibernate throws org.hibernate.NonUniqueObject Exception.

→ Instead of update method if we call merge() then hibernate only copies the state changes (values) of detached state Object into an Object in the cache with the same Identifier value (Id value) and then the changes are updated to the DataBase when the Transaction is Committed.

→ While Converting a detached Object into persistent then it is always better to call merge(-) instead of update(-) method.

```
// UpdateMerge.java
import org.hibernate.*;
import org.hibernate.cfg.*;
class UpdateMerge
{
   p s v m(String args[])
   {
SessionFactory factory=new Configuration().configure().buildSessionFactory();
   Session session1= factory.openSession();
      Product P1=null; // P1 ⟶ transient
Object O= session1.get(Product.class, 111);
      P1=(Product)O; // P1 ⟶ persistent
   session1.close();
P1.setPrice(10000); // P1 ⟶ detached
```

This modifications are not done in the DataBase in detached state object. modification are one in the only persistent state

```
Session session2 = factory.openSession();
   Product P2=null; // P2 ⟶ transient
Object O2 = session2.get(Product.class, 111);
   P2 = (Product)O2; // P2 ⟶ persistent

Transaction tx = session2.beginTransaction();
   // session2.update(P1); ⟶ throws Exception
   session2.merge(P1); // P1 ⟶ persistent
    tx.commit();
   session2.close();
   factory.close();
   }
}
```

Composite-id :-

→ If a DataBase table has morethan one primarykey column then while writing mapping file of hibernate, instead of using <id> tag, we need to use <composite-id> tag.

→ While writing the mapping file of hibernate, it is the mandatory that either <id> tag or <composite-id> tag must be configured

→ When a table has a Single primary key (or) no primary keys then we need to Configure <id> tag in mapping file. If morethan one primary keys in table then <composite-id> tag is required.

→ Syntax:-

```
<composite-id>
    <key-property  name="Variable name"  column="primarykey colname"/>
    <key-property  name="Variable name"  column="primarykey colname"/>
</composite-id>
```

25-02-2013

→ While Saving an Object into the DataBase with CompositId, we need to create a just POJO class object, and we assign properties for the object and then we save it in DataBase. It means we dont have any new step while saving an Object.

→ When loading an Object from a DataBase, we need to pass Composite-Id for hibernate to load it from a DataBase. It means we need to Create an object of POJO class, then we need to assign Primarykey properties for that Object and then we need to Pass that Object as a parameter for either get (or) load methods to load an Object from a DataBase.

→ At the time of loading an Object, to pass a POJO class object as a Second parameter for either get or load methods, the Object must be a Serialized object.

→ To make a POJO class Object as a Serialized object, we implement the class from java.io.Serializable(Interface)

Example :-

→ In the following Example, we are going to Store and load book class objects in a DataBase. Here bookid & isbnNumber are taken as Primarykeys. i.e. Composite id

→ At the time of saving an Object in DataBase, we donot have any changes but at the time of loading a Book, we are passing bookid & IsbnNumber as a book object for get(or) load(--) methods.

CompositeApproach
- Book.java
- book.hbm.xml
- hibernate.cfg.xml
- BookInsert.java
- BookSelect.java
- *.class

```
//Book.java
public class Book implements java.io.Serializable
{
    private int bookId;
    private String bookName;
    private double price;
    private int isbnNumber;

    Setters & getters
}
```

```
//Book.hbm.xml
<!--DOCTYPE ----->
<hibernate-mapping>
  <class name="Book" table="book_info">
    <composite-id>
      <key-property name="bookId" column="bid"/>
      <key-property name="isbnNumber" column="isbn"/>
    </composite-id>
    <property name="bookName" column="bname" length="10"/>
    <property name="price"/>
  </class>
</hibernate-mapping>
```

NOTE:- While creating a mapping file of hibernate, either <id> or <composite-id> tag is mandatory.

```
//BookInsert.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public class BookInsert
{
psvm(String[] args)
{
SessionFactory factory = new Configuration().configure().buildSessionFactory();
Session session = factory.openSession();
Book b=new Book();
b.setBookId(1);
b.setBookName("java");
b.setPrice(300.0);
b.setIsbnNumber(10111);
Transaction tx = session.beginTransaction();
session.save(b);
tx.commit();
session.close();
factory.close();
}
}
```

```
//BookSelect.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public class BookSelect
{
psvm(String[] args)
{
SessionFactory factory = new Configuration().configure().buildSessionFactory();
Session session = factory.openSession();
```

```
Book b=new Book();
b.setBookId(1);
b.setIsbnNumber(10111);

Book b1=(Book)session.get(Book.class, b);
                          ↳ here 'b' is Serializable type. b/c Book pojo class implemented by Serializable. There is a two primarykeys. This Serializable object internally converts primitive types serialized data
S.o.p("BookName=" +b1.getBookName());
S.o.p(" price=" + b1.getPrice());
session.close();
factory.close();
}
}
```

→ Before executing this example, the following table is required in the DataBase.

SQL> Create table book_info (bid number(5) ~~primary key~~, bname varchar2(10), price number(4,2), isbn number(5), primary key(bid, isbn));

```
>javac *.java
>java BookInsert
Hibernate: insert into book_info (bname, price, bid, isbn) values (?, ?, ?, ?)
>java BookSelect
Hibernate: select book0_.bid .....
Bookname = java
price = 300.0
```

Generators in Hibernate

→ Before storing an object in a DataBase, hibernate verifies weather an "id value for that object is set or not.

→ In Hibernate Id value for an object can be generated by either a programmer or by hibernate. This information will be given to the hibernate using <generator> tag as a subtag of <id> tag.

→ In Hibernate, the framework as already provided some predefined primary key generation algorithms. and existing algorithms are not suitable for an application then Hibernate has given a procedure for creating user-defined primary key generator algorithms.

→ a list of predefined generators given by Hibernate are

1. assigned
2. increment
3. sequence
4. Identity
5. hilo
6. native
7. foreign
8. uuid.hex

etc.....

26/02/2013

① assigned :-

→ The default generator class in Hibernate is "assigned"

→ If generator class is assigned then hibernate is expecting the id value for an object from a programmer, before an object is going to be saved in the DataBase.

→ Incase of assigned generator, it is not mandatory to add generator as the sub tag for <id> tag.

→ This assigned generator is DataBase Independent. It means it works (assigned) on any DataBase.

For Eg:- <id name="ProductId" column="pid">
<generator class="assigned" />
</id>

(or)

<id name="productId" column="pid" />

② increment:-

→ If generator class is added as "increment" then hibernate uses a formula called max(Id)+1. and generates an "id" value and then assigneds this Id value for the new object, before it is going to save an Object in DataBase.

→ If a programmer is set the value for the <id> property of an object then hibernate simply ignores the "id" value set by the programmer.

→ If a DataBase table doesnot contain any rows then Hibernate assumes max(Id) as "0"

→ This increment generator is a DataBase independent generator.

Eg:- <id name="productId" column="pid">
<generator class="increment" />
</id>

→ At the time of Save operation, Hibernate is going to generate a Select and Insert commands

Hibernate: select max(pid) from product
Hibernate: insert into product(pname, price, pid) values (?,?,?)

③ sequence:-

→ If generator class is sequence then hibernate selects next value of a Sequence from the DataBase and assigns that value as an "id" for the Object and then that object will be saved in the DataBase.

→ while Configuring this generator class Sequence, we need to pass a parameter called "sequence" with the sequence name created in DataBase as a value.

→ If a sequence name is not passed as a parameter then hibernate creates itsown sequence called Hibernate_Sequence and selects next value of the sequence and assigns it as an "id" before an Object is going to save in a DataBase.

→ This sequence generator is a DataBase dependent. for Example, it works for Oracle only but it does not work for mysql.

→ Eg1:-

```
<id name="productId" column="pid">
    <generator class="sequence" />
</id>
```

while save an Object (Product Obj) in a DataBase, Hibernate is going to generate the following SQL Commands.

```
Hibernate: select hibernate_sequence.nextVal from dual
Hibernate: insert into product (pname, price, pid) values of (?,?,?)
```

→ Eg2:-

```
SQL> create sequence my_sequence increment by 5;

<id name="productId" column="pid">
    <generator class="sequence">
        <param name="sequence">my_sequence</param>
    </generator>
</id>
```

while saveing an Object (product obj) in a DataBase, Hibernate is going to generate the following SQL Commands

```
Hibernate: select my_sequence.nextval from dual
Hibernate: insert into product (pname, price, pid) values of (?,?,?)
```

④ hilo :-

→ This generator class uses a special formula ~~for~~ generating Id for an Object, before it is going to store in a DataBase.

→ when this generator is executing for 1st time then the Id value generated is '1'

→ when this 'hilo' generator is executed for 2nd time onwards then it uses a formula for generating the 'id' value called " max_lo * column + column "

→ while configuring this hilo generator in a mapping file, along with generator we need to configure 3 parameters

① table
② column
③ max_lo

→ If we donot configure the above 3 parameters along with " hilo " generator then hibernate takes default values for the parameters as

table ⟶ hibernate_unique_key
column ⟶ next_hi
max_lo ⟶ 32767

→ when first time hilo generator is executed then after generating the "id", hibernate stores a value "1" in next_hi column of hibernate_unique_key table.

→ when hilo generator is executed for 2nd time then 1st it calculates the next id by using the formula and then increments the value of next_hi column by "1". i.e. nex_hi column value becomes '2'.

→ If we configure the above '3' parameters then hibernate considers programmer assigned values, instead of the default values.

delete from product;
commit;
drop table product;
drop table hibernate_unique_key;
commit;

for Eg:-

```
<id name="productId" column="pid">
    <generator class="hilo">
        <param name="table">my_table</param>
        <param name="column">next</param>
        <param name="max_lo">10</param>
    </generator>
</id>
```

→ when a product object is saved using this "hilo" generator for 1st time then productId will be assigned as "1". After generating the "Id", hibernate stores a value "1" in next column of my_table.

→ After executing same operation, we will get the following output into the DataBase.

SQL> Select * from product_table;

pid	pname	price
1	sony	8000

SQL> Select * from my_table;

next
1

→ If another product is saved in DataBase then "hilo" generator generates the "id" value as "11" (max_lo * next + next i.e. 10 * 1 + 1 = 11). After generating the id, hibernate increments the value of next column by 1

→ At DataBase we will get the following output.

SQL> Select * from product_table;

pid	pname	price
1	sony	8000
11	Lg	6000

SQL> select * from my_table;

next
2

→ This hilo generator is Independent of DataBases.

→ while configuring this "hilo" generator, we can also configure some parameters instead of all the three in xml file.

⑤ Identity :-

→ This Identity generator is used for representing autoincrement column of a DataBase table, in a mapping file of hibernate.

→ autoincrement column is not supported by all DataBases servers. So, this Identity is a DataBase dependent generator.

forEg:- oracle doesn't support autoincrement column, but mySQL supports autoincrement column. So, Identity generator works for mySQL, but not for oracle.

Eg:-
```
<id name="studentId" column="sno">
    <generator class="identity" />
</id>
```

⑥ native :-

→ This generator is not a new generator class. It is equal to any one of the following 3 generators. ① sequence
② Identity
③ hilo

→ Hibernate 1st checks sequence is supported by the DataBase or not. If yes then native is equal to sequence.
If not then hibernate checks for Identity. If yes then native is equal to Identity
If not then finally hibernate uses "hilo".

Eg:-
```
<id name="productId" column="pid">
    <generator class="native" />
</id>
```

⑦ foreign :-

→ This generator class is only useful when applying one-one relationship blw the objects. This foreign generator is used to copy the id value of one object to Id value of another object.

→ This foreign generator is independent of DataBases.

uuid.hex:

→ This generator class is used for a primarykey of "String type" in a DataBase table. UUid stands for Universal Unique id.

→ UUid is an alogrithm it generates a Unique String of 32 hexadecimal digits, by based on the following 4 values.

① Ipaddress of the System.

② Startup time of JVM

③ System time

④ Counter value in JVM

→ for String type of primary key columns, either coe can use generator class as this UUID.hex (or) assigned.

→ The remaining generator classes can be used only for primarykeys of number type.

→ In most of the cases, primarykeys in tables are taken as number type only. So, this uuid.hex is not a frequently used generator class.

Creating a Custom generator class :-

28/02/2013

→ Hibernate framework has provided some predefined generator class is @generation for id's for an objects. If the existing generator classes are not suitable for our application then we can also create an User-defined generator class we call such generator class as a "custom generator".

→ Creating a Custom generator in an Hibernate appn involves two steps.

Step-I:- We need to create a generator class by implementing it from an interface org.hibernate.id.IdentifierGenerator

Step-II:- Apply the generator class for an <id> tag in a mapping file using <generator> tag

→ IdentifierGenerator is an interface provided a Single Abstract method called generate(). so, we need to implement generate() with the logic, to generate an Unique Identifier for an Object.

Eg:-
```
public class MyGenerator implements IdentifierGenerator
{
    public Serializable generate(SessionImplementor si, Object o)
    {
        //logic
    }
}
```

→ In a mapping file of Hibernate this Generator class can be configured for an <id> tag like the following

```
<id name="productId" column="pid">
    <generator class="MyGenerator" />
</id>
```

→ The following Generator class is used for generating a product-Id by using the logic a Random number from 0-10 multiply with length of the product name.

```
// MyGenerator.java
import org.hibernate.id.*;          ——→ IdentifierGenerator
import org.hibernate.engine.*;      ——→ SessionImplementor
import java.util.*;                 ——→ Random
import java.io.*;                   ——→ Serializable
```

```
public class MyGenerator implements IdentifiesGenerator
{
public Serializable generate (SessionImplementator si, Object o)
{
Random r = new Random();
int k = r.nextInt(10); // It contains 0-10 numbers line by line serialized
Product p = (Product)o; // it is typecast into product pojo class
String str = p.getProductName(); // Eg: SONY → 4 letters  1*4=4
int length = str.length(); // SAMSUNG → 7 letters. length=7
Integer i = k*length; // i=1*7=7  =2*7=14
return i;
}
}
```

wrapperclass

↳ it contains serialized numbers.

Wrapper classes shows null value but premitives shows '0' value So, Enduser don't confuse to see the result. it is '0' value. so we have to take wrapper class then it contains 'null' value.

Servlet with Hibernate Integration :-

→ In Servlet to a DataBase communication, a Servlet can use JDBC code to talk with a DataBase.

→ In place of JDBC code, if we use Hibernate then we call this consept as Servlet with Hibernate Integration.

→ In Servlet with Hibernate Integration, we have the following 3 approaches for developing a Servlet with Hibernate persistence logic

→ Approach 1 :-

→ we can insert the steps for creating Hibernate client application in Service method of a Servlet.

→ If Service(-,-) method contains building SessionFactory then when the no.of

Clients requests are increasing then the no.of SessionFactories are also increased at the Server side. SessionFactory Contains no.of sessions. For Each & Every request more sessionFactories will be created. one sessionFactory Contains no.of sessions. so, it is heavy weight obj for the one sessionFactory. If more sessionFactories will created then the server side problem will there. it means less got

→ If no.of SessionFactories are increased then burden on a Server is also increased Because SessionFactory of Hibernate is a Heavyweight Object.

→ This approach is not a good approach for integrating a Servlet with Hibernate.

Eg:-

```
public class MyServlet extends GenericServlet
{
   public void service(request, response) throws SE, IOE
   {
   // Configuration Object
   // configure()
   // buildSessionFactory
   // openSession
   // beginTransaction
   // commit
   // close
   }
}
```

Approach-2:-

→ instead of defining the SessionFactory creation in Service(-,-) method, we can devide building SessionFactory into init(), session logic into Service() and closing the sessionFactory into destroy().

The Advantage of this approach is, SessionFactory of Hibernate is generated for once, not for every request on a Servlet Object. Bcz init() of a Servlet will be called for once per a Servlet Object.

→ If an Application Contains multiple Servlets and if this Approach-2 is applied then a SessionFactory is seperately generated for Each Servlet for once. It means, if an application Contains 4 servlets then

SessionFactory of hibernate is generated for four times.

→ This Approach-2 is suitable, if an Application contains Single Servlet to Integrate with Hibernate.

→ Eg:-

```
public class MyServlet extends GenericServlet
{
   public void init() throws SE
   {
      //Configuration object
      // Configure()
      // buildSessionFactory
   }
public void service(request, response) throws SE, IOE
   {
      //openSession
      // beginTransaction
      // commit
      // close session
   }
public void destroy()
{
   //close factory()
  }
}
```

01/03/2013

Approach-3:-

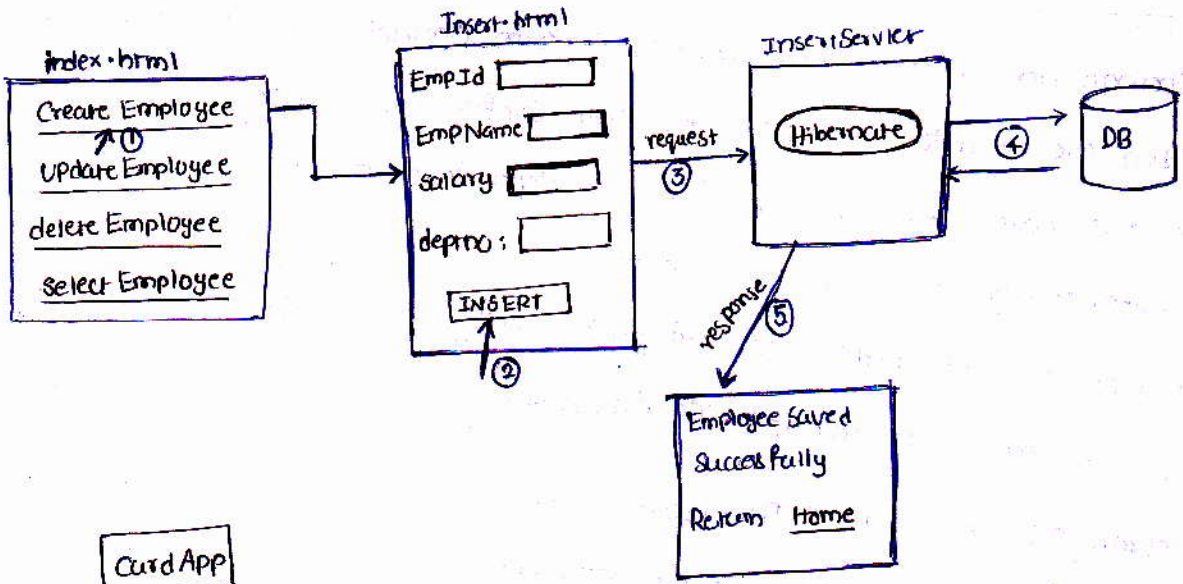
→ when multiple Servlets are integrating with a Hibernate in a webappn then it is always better to make SessionFactory of Hibernate as Singleton and share it with all Servlets of the webappn.

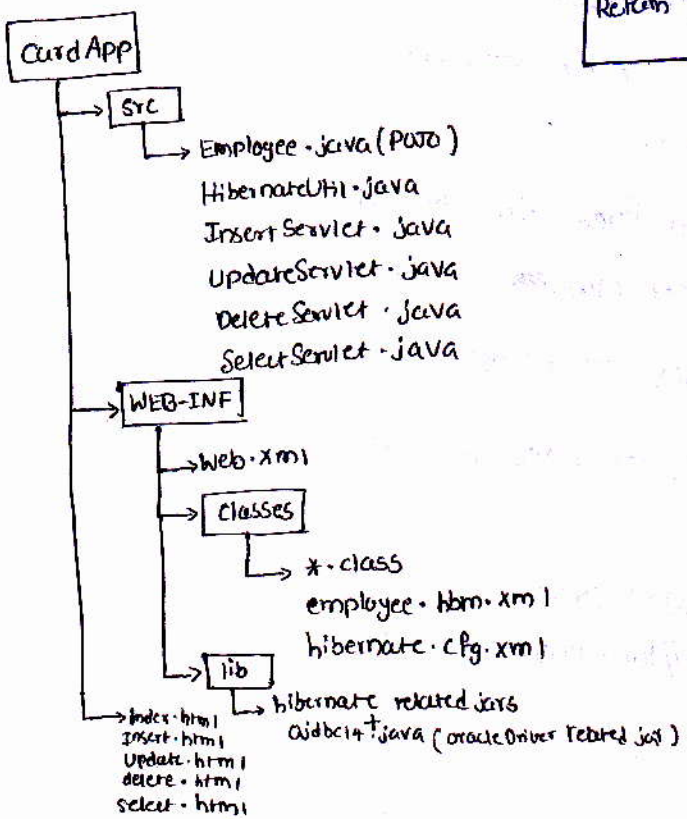
→ In order to make SessionFactory of Hibernate as Singleton, we need to create a separate class with a StaticFactory method for building (or) generating a SessionFactory for only once.

→ The following is a Utility class for making a SessionFactory of Hibernate as Singleton.

```
import org.hibernate.*;
import org.hibernate.cfg.*;
public class HibernateUtil
{
  private static SessionFactory factory;
  public static SessionFactory gerSessionFactory()
  {
     if(factory==null)
     {
factory=new Configuration().configure().buildSessionFactory();
     }
  return factory;
  }
}
```

→ The following example is to perform CURD operations on Employee objects using Servlet with Hibernate Integration.





→ When Integrating a Webapplication with hibernate, we need to copy POJO class, mapping file, Configuration file of Hibernate into classes folder and the main and dependent jar files of Hibernate into lib folder

→ To compile the .java files of the above application, we need to set the following 2 jar files in the CLASSPATH.

① Servlet-api.jar ② Hibernate3.jar

→ Deploye the application Root Directory (CURDApp) into <Tomcat_home>\webapps folder.

→ Start the Tomcat Server and send the following request from a Brocoser.

http://localhost:2013/CurdApp

02/03/2013 hibernate3.6.0/project/src.

*) Connecting with Multiple databases :-

→ From an Hibernate application, if we want to perform operations on Multiple DataBases at a time then we need to construct multiple configuration files.

→ In hibernate the no.of configuration files are dependending on the no.of Databases Connecting Simultaneously from an application.

→ In a client appn of Hibernate, to load each configuration file, we need a SessionFactory. It means for multiple configuration files we need multiple factories

→ To perform operations on multiple DataBases, we need a session with each DataBase and Transaction for each session. so, we need multiple sessions and multiple Transactions with in a client appn.

→ when creating multiple configuration files for connecting with multiple DataBases, The connection properties and Dialect classes in configuration files will be different and the remaining properties are almost same.

→ For Connecting a DataBase of mysql, we need to configure the following connection properties and Dialect class.

Driverclass : com.mysql.jdbc.Driver
Url : jdbc:mysql://localhost:3306/test
Username: root1
Password : root1

dialect : org.hibernate.dialect.MySQLDialect

→ For connecting with postgreSQL DataBase, the following are the connection properties and the Dialect class.

Driver class : org.postgresql.Driver

username : postgres

password : Postgres

Url : jdbc:postgresql://localhost:5432/postgres

dialect : org.hibernate.dialect.PostgreSQLDialect

NOTE:- ① If connection properties (or) Dialect class are unknown for a DataBase then we can get the Information from "hibernate.properties" file that is Distributed along with Hibernate s/w.

② The Hibernate.properties file is available at D:\hibernate-distribution-3.6.5 Final\project\etc folder

→ Apart from Hibernate jar files, we need to set mySQL and postgreSQL Driver related jar files in the CLASSPATH, while connecting with mySQL & postgreSQL at a time.

→ com.mysql.jdbc_5.1.5.jar

→ postgresql-jdbc3.jar

*) TO open mySQL prompt then do the following

Start → programs → mySQL → MySQL server 5.0 → MySQL Command Line

→ Enter password : root1

mysql> connect test
(test → DataBase name)

mysql> select * from student;

*) TO open Commandline client of postgresql, do the following

Start → programs → postgresql 8.3 → psql to postgres →

Enter password : postgres

postgres=# select * from student;

MultipleDBTest

↳ Student.java
Student.hbm.xml
hibernate_mysql.cfg.xml
hibernate_Postgresql.cfg.xml
InsertClient.java

hibernate with ConnectionPooling

→ While obtain a Connection with a DataBase using a DriverManager class then DriverManager class Directly opens a Connection with a DataBase Server. and it is a non reusable Connection

→ If a non-reusable Connection is opened then after DataBase operations are compleeted then it must be closed. Otherwise it will be still in OpenState and that Connection is a non-reusable for another Client.

→ If each time a Connection is opened and then closed then this opening and closeing makes burden on a DataBase server. so, the performance of the DataBase server will be decreased.

→ In order to overcome the above problems, we got a technique called Connection pooling.

→ In Connection pooling technique a set of Connections are opened with a DataBase Server and stored them in a pool. The Connections in the pool are reused for multiple ~~times~~ Clients.

→ With the help of this reusability, the burden on a DataBase Server will be reduced, sothat the performance of the DataBase server will be increased.

→ A Connection pool will be Created and monitered by the Server administrator at ServerSide in realtime apps.

03/03/2013

Q:- What is the diff b/w a pool and a cache?

Ans:- (a) In a pool all objects are equal objects. It means 1st object in the pool is exactly same as a last object in the pool. we also call this type of objects as stateless objects. But a cache is a group of stateful objects because in a cache no two objects are equal.

(b) In a pool, when it is busy then a client waits until one of the multiple objs in pool is free. But when a cache is busy then a client waits unHil a particular object is free. It means waiting time in a pool is lessthan waiting time in a cache.

→ By default every hibernate application uses ConnectionPooling Technique internally while talking with DataBases. The built-in Connection pooling in hibernate is also called JDBC connection pooling.

→ In hibernate the connection pooling is devided into the following 3 catagories

1) Built-in JDBC connection pool
2) Third party connection pools
3) Server side connection pool.

1) Built-in Connection pool:-

→ By default, hibernate creates internally a Connection pool when an application execution is started

→ By default, the built-in connection pool of hibernate contains minimum capacity as '1' and maximum capacity as '8'.

→ As a programmer we can change the maximum capacity of the Built-in Connection pool by setting (or) Configuring hibernate.connection.pool_size property in hibernate Configuration file.

for Eg:-

<property name="hibernate.connection.pool_size">15</property>

② Third party Connection pool:-

→ Hibernate is recomended to use the built-in JDBC Connection pool, only for Testing the applications of Hibernate. but it cannot be used for production mode applications (Real time apps).

→ While creating Realtime applications with Hibernate and without a server environment then Hibernate is recomended to use any on of the following two third party Opensource Connection pools, which are distributed along with hibernate s/w.

① C3P0 connection pool

② proxool connection pool

① C3P0 connection pool :-

→ C3P0 is an open source third party connection pool, it is distributed along with hibernate in the form of a jar file C3P0-version.jar.

→ To activate this C3P0 connection pool, we need the following two changes.

① add C3P0 related properties to the configuration file of Hibernate.
(hibernate.cfg.xml)

② add C3P0-version.jar in the CLASSPATH.

→ For C3P0 Connection pooling, we need to add the following 3 properties in the Configuration file.

```
<property name="hibernate.connection.provider_class">org.hibernate.connection.
                                        C3P0ConnectionProvider </property>

<property name="hibernate.c3p0.min_size">2 </property>
<property name="hibernate.c3p0.max_size">10 </property>
```

→ C3P0 jar file is available at the following location.

<hibernate_home>\lib\optional\c3p0\c3p0-version.jar.

② proxool Connection pool :-

→ This proxool connection pool is also an opensource third party connectionpool, it is distributed along with hibernate in the form of a jar file
proxool-version.jar

→ when this proxool connection pool is activated then automatically the builtin JdbcConnection pool will be deactivated.

→ To activate this proxool Connectionpool, the following are the 3 changes

required.

① Create a seperate xml file with the connection properties and maxsize of the pool. This xml filename can be <anyname>.xml

② Configure the separate xml file created and along with it the provider class and alias name. in hibernate configuration file.

③ ~~At~~ While executing, Set proxool-version.jar in the CLASSPATH

→ While creating a separate xml file for a proxool it doesn't required any DTD for the xml and the root element of the xml can be any name.

→ In proxool connection pool minimum size of the pool is '1' and we can't set minsize explicitly. But we can set maxsize of the pool.

```
<!-- proxool.xml -->
<something>
  <proxool>
    <alias> abcd </alias>
    <driver-url> jdbc:oracle:thin:@localhost:1521:orcl </driver-url>
    <driver-class> oracle.jdbc.OracleDriver </driver-class>
    <driver-properties>
      <property name="user" value="scott" />
      <property name="password" value="tiger" />
    </driver-properties>
    <max-connections-count> 5 </max-connection-count>
  </proxool>
</something>
```

→ In hibernate configuration file we need to remove connection properties and in place of them we need to add proxool properties.

```
<!-- proxool properties -->
<property name="hibernate.proxool.xml"> proxool.xml </property>
<property name="hibernate.proxool.pool_alias"> abcd </property>
<property name="hibernate.connection.provider_class"> org.hibernate.Connection.ProxoolConnectionProvider </property>
```

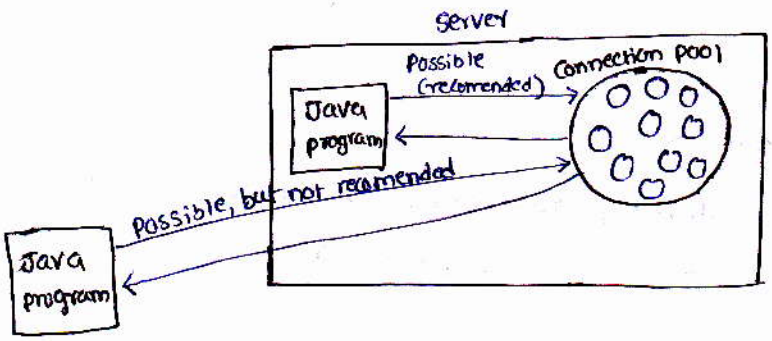
04/03/2012

3) Server-side connection pool :-

→ In hibernate, when an application is running at Serverside, it is recomended to use a Serverside Connection pool configured by the Server administrator.

→ Actually if a connection pool is configured at server-side then a Java application running either with in the Server or an application running at out of the server can access that Connection pool.

→ If a Java application running at out of the server then to access a connection from pool, the server must be started. If a Java application is running inside the server then it is recomended to use a Serverside connection pool.



→

→ At Serverside, an Administrator configures a Connection pool and also a DataSource.

→ A DataSource acts like a medicator between a Java program and a Connection pool

→ A Java program will ask a DataSource to obtain (get) a Connection from a pool.

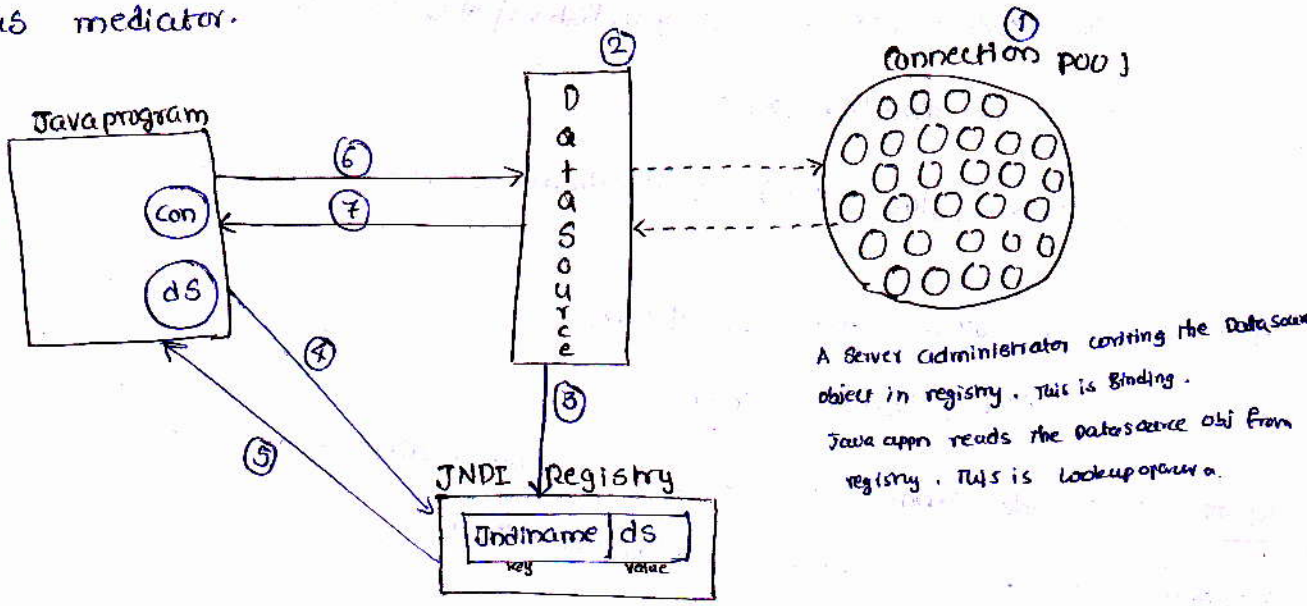
→ In the middle a DataSource gets (obtains) the connection from pool, then creates proxy connection and then that proxy connection will be send to Java program.

→ A proxy connection is also called as a logical Connection and it is not a real connection, but acting like a real connection in our Java program

→ When a Java program closes the Javaprogram then internally the real connection (or) (pool connection) sent back to the pool.

→ A Server administator stores DataSource object in Jndi registry of the Server. A Jndi registry is a common area, to share the objects between programers and administators.

→ A Java program first connects with registry and takes DataSource object and then obtains a connection from a pool through DataSource object as mediator.



W.r.to Diagram

① A Server administrator Configure a connectionpool in a server for a DataBase.

② A Server administator Configure DataSource as a mediator for a Connection pool.

③ An administator Registered (stored) DataSource object in JNDI registry.
This is also called "Binding".

④&⑤ A Javaprogram uses Jndi properties and Connects with registry and obtains DataSource object, by using its Jndiname (key). This is also called "Lookup operation".

⑥&⑦ A Javaprogram asks for a Connection by calling getConnection() method on DataSource object. and finally obtains a Connection from the pool.

→ In Case of hibernate, hibernate internally obtains a DataSource from registry and then obtains a Connection from the pool through DataSource Object. For this, we need to remove Connection properties and inplace of them we need to attach Jndi properties in hibernate.cfg.xml.

① hibernate.connection.datasource ⟶ JndiName (key)

② hibernate.jndi.class } depends upon the server like Glassfish Server (or) Weblogic server.

③ hibernate.jndi.url

05/03/2013

Configuring a Connection Pool in GlassfishV3 Server :-

Step-I :- Copy ojdbc14.jar in C:\glassfishv3\glassfish\domains\domain1\lib\ext folder.

Step-II :- Start the glassfish server

C:\glassfishv3\glassfish\bin> asadmin start-domain domain1

Step-III :- Open the browser and type the url

http://localhost:4848

Step-IV :- At left side expand Resources ⟶ expand JDBC ⟶ select Connection pools ⟶

click [New] button ⟶ enter the following details

Name : [poolone] ← any name

Resource Type : [javax.sql.DataSource]

Database vendor : [Oracle] ⟶ click [Next] button

Step-V :- Enter the following 3 additional properties

Password	tiger
URL	jdbc:oracle:thin:@localhost:1521:orcl
User	scott

click on [finish] button

Step-VI :- At left side select JDBC Resources ⟶ New ⟶ Enter JNDI Name : [jndi1]

⟶ Pool Name [Poolone] ⟶ OK

NOTE :- To Verify weather a Connection pool is successfully created or not, click on Poolname ⟶ [Ping]

The following example is to obtain a connection from a Connectpool of GlassFish Server for storing a Student object in the DataBase using Hibernate.

JndiTest
- Student.java
- Student.hbm.xml
- hibernate.cfg.xml
- Client.java

For this example we need to set the following 3 jarfiles of GlassFish server, apart from all the jarfiles of hibernate in the CLASSPATH.

① appserv-rt.jar (c:\glassfishv3\glassfish\lib)
② javaee.jar (c:\glassfishv3\glassfish\lib)
③ imqjmsra.jar (c:\glassfishv3\glassfish\lib\install\application\jmsra)

for this example refer page NO: ⑧ an application ④ of the Handout.

NOTE:- When executing the above example, if we get an exception with message ~~slf4j-api~~-1.6.0 (or later) is incompatible then set Slf4j-api-jar in classpath by removing Slf4j-api-1.6.0.jar from the CLASSPATH. (we can doconload Slf4j-api.jar from hibernate-3.4)

Configuring a Connection pool using weblogic 10.x:

Step-I:- Create a Server domain (if no domain exist)

Start → programs → Oracle Weblogic → weblogic Server → TOOLS →
Configuration wizard

Step-II:- ⊙ create a new weblogic domain → Next → Next → Enter

Domain name: hibernate_domain → next →
User name: weblogicadmin
Password: weblogic10
Conform password: weblogic10
→ next → next → next → create → Done

Step-III:- Start the weblogic server

C:\Oracle\Middleware\user_Projects\domains\hibernate_domain\StartWeblogic.bat

Step-IV:- open the Browser and Type the following Url in address bar

http://localhost:7001/console

Username : Weblogicadmin

password : Weblogic10

Login

Step-V:- Expand Services at leftside ⟶ expand JDBC ⟶ select DataSources ⟶

New ⟶ Enter the following (poolname)

Name: poolone

JNDI Name: oraclejndi

Database Type: Oracle

Database Drives: Oracle's Driver (Thin) for Service Connections ⟶ next ⟶ next ⟶ Enter the following Connection properties

Database Name: Orcl

HostName : localhost

port : 1521

DataBase username: Scott

Pass word : Tiger

Confirm password: Tiger ⟶ Next

Step-VI:- Click on TestConfiguration ⟶ Next ⟶ ☑ Adminserver ⟶ Finish

NOTE:- if you want to set Capacity for a Connection pool then click on poolone (Poolname) ⟶ select connectpool tab ⟶ Enter the Capacity Initial Capcity: 1, maximum Capacity: 10 ⟶ save

In the previous example, if we want to use Connection pool of WebLogic Server then in hibernate.cfg.xml file, we need to add the following Jndi properies

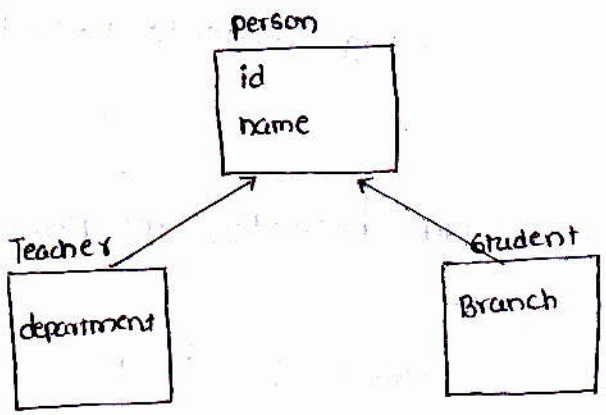
```
<property name="hibernate.connection.datasource">oraclejndi</property>
<property name="hibernate.jndi.class">weblogic.jndi.WLInitialContextFactory</property>
<property name="hibernate.jndi.url">t3://localhost:7001</property>
```

In CLASSPATH we add the following one additional jar file called WebLogic.jar (C:\oracle\middleware\Wlserver_10.3\server\lib).

Inheritance mapping in hibernate

→ While creating applications in Java, if more than one class is going to have some common attributes then instead of repeating the common attributes individually in each class, we apply inheritance mechanism by separating generalized attributes and specialized attributes

→ For example, we have two classes called Teacher and Student with common attributes as id and name. and special attributes as department in Teacher and branch in Student. Then we can convert this application into inheritance by creating a parent class (or) Base class has "person like the following



→ While persisting the inherited objects i.e. a Teacher (or) a Student object then hibernate persists the inherited properties also in the DataBase. It means both generalized and specialized attributes in the DataBase. So, we say that hibernate supports inheritance.

→ To inform the Hibernate about the generalized and specialized properties of an Object, we need to add some additional Configuration in a mapping file (hbm file) This is called inheritance mapping in hibernate.

→ Inheritance mapping in hibernate is also called Hibernate hierarchies.

→ Hibernate has provided the following 3 strategies to represent the inherited data in DataBase.

① Table per class hierarchy
② Table per subclass hierarchy.
③ Table per Concreate class hierarchy.

① Table per class hierarchy :-

→ In this type of strategy, hibernate uses a single Table in DataBase for representing the entire objects of a class hierarchy of a Java application.

→ In other words, if Table per class hierarchy, the objects of all derivid classes are stored in a single table DataBase.

→ In this Type of hierarchy, a DataBase table contains columns for generalized attributes, columns for specialized attributes and an additional column called Discriminator column.

→ A Discriminator column stores a Descriminator value, which is used to identify one Derived class object among multiple Derived classes easyly in a DataBase.

→ While working with this Table per class hierarchy, the Descriminator column is mandatory in a DataBase table.

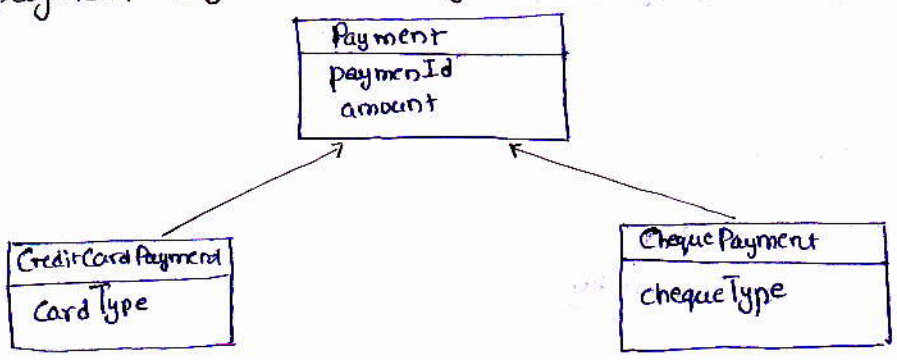
→ Table per class hierarchy can't be implemented by without using Descriminator Column

→ While Creating a Single table, the columns related to Specialized attributes (Derived classes attributes) should not be applied with not null constraint.

→ A Descriminator column stores a Descriminator Value and it is used to identify one Derived class object among multiple Derived classes objects stored in a Single table.

Eg:- ① We have a Base class called payment, with two derived classes creditCardPayment and cheque payment.

② we are applying TablePerclass hirarchy for storing both credit card and cheque payment objects in a Single table of DataBase.



primary key ↓ | discriminator column ↓

Payments

pid	amount	ccType	chType	pType

→ In hibernate mapping file, we need to use <subclass> tag for mapping subclasses of the hirarchy.

→ We have two approaches for mapping the subclasses.

① We can configure the Base class and the subclasses in a Single Mapping file.

② We can Separate Baseclass and each subclass configuration into a Different mapping file.

in the above two approaches, The 1st approache is preferable, Bcz it reduces the no.of xml files (mapping files). Apart from this, in a Configuration file we no need to configure mapping tag for multiple times.

1st approach:-

```
<!-- Payment.hbm.xml -->

<hibernate-mapping>
    <class name="Payment" table="payments">
        <id name="paymentId" column="pid" />
***     <discriminator column="ptype" type="java.lang.String" />
        <property name="amount" />
****    <subclass name="CreditCardPayment" discriminator-value="credit">
            <property name="cardType" column="cctype" />
        </subclass>
        <subclass name="ChequePayment" discriminator-value="cheque">
            <property name="chequeType" column="chtype" />
        </subclass>
    </class>
</h-m>
```

IInd approach:-

Payment.hbm.xml

```
<h-m>
    <class name="Payment" table="payments">
        <id name="paymentId" column="pid" />
```

```
<discriminator column="pmode" type="java.lang.String" />
<property name="amount" />
</class>
</h-m>
```

Credit.hbm.xml

```
<h-m>
<subclass name="CreditCardPayment" extends="Payment" discriminator-value="credit">
<property name="cardType" column="cctype" />
</subclass>
</h-m>
```

Cheque.hbm.xml

```
<h-m>
<subclass name="ChequePayment" extends="Payment" discriminator-value="cheque">
<property name="chequeType" column="chtype" />
</subclass>
</h-m>
```

TablePerClass

- Payment.java
- CreditCardPayment.java
- ChequePayment.java
- Payment.hbm.xml
- hibernate.cfg.xml
- InsertClient.java
- *.class

for this example refer page no: (9) an Application (5) of handout.

After executing this table per class client application then the following output is generated in DataBase.

SQL> Select * from payments;

PAYID	Amount	PDATE	CCTYPE	CHTYPE	PMODE
1	5000	07-MAR-13	VISA		CR
2	8000	07-MAR-13		ORDER	CH

② Table per Subclass Hirarchy :-

→ In this type of hirarchy, hibernate represents the inheritance data in separate tables, but not in a Single table.

→ In this type of inheritance mapping, hibernate uses a separate table for storing Base class attributes and a separate table for each derived class for storing the Derived class attributes.

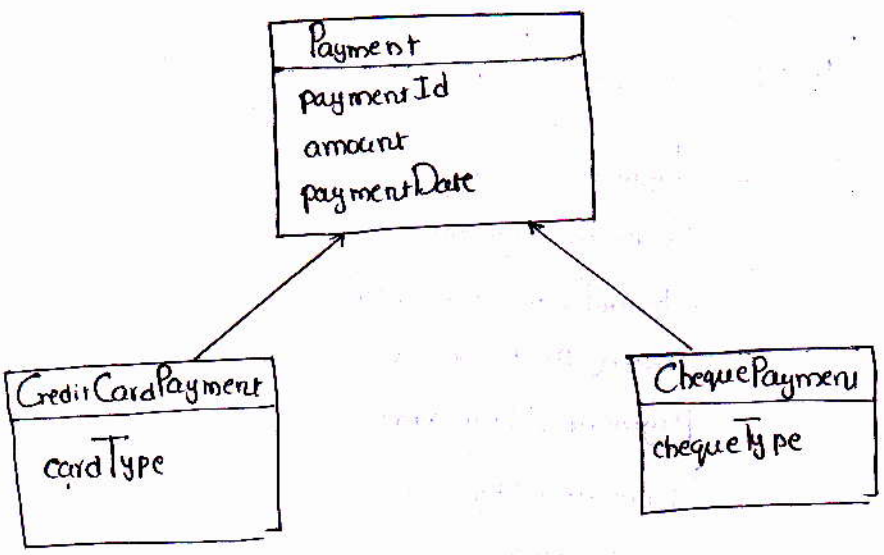
→ In order to maintain the relationship between the data stored in Base class table and Derived class tables, hibernate uses Foreign key relationship.

→ In this Table per subclass hirarchy, the use of descriminater Column is optional.

→ In a hibernate mapping file, we need to use a tag <joined-subclass> for Configuring Table per subclass hirarchy.

→ Example :-

We have a Baseclass called Payment with two derived classes CreditCardPayment and ChequePayment. We are applying TablePerSubclass hirarchy for representing this data in a DataBase by hibernate.



Primarykey
↓
payid — payments — amount — pdate

Foreignkey
↓
pid — Credit-table — cctype

Foreignkey
↓
pid — cheque-table — chtype

payment.hbm.xml

```
<hibernate-mapping>
   <class  name="Payment"   table="payments">
      <id  name="paymentId"  column="payid" />
      <property  name="amount" />
      <property  name="paymentDate"  column="pdate" />
      <joined-subclass  name="CreditCardPayment"  table="Credit_table">
         <key  column="pid" />
         <property  name="cardType"  column="cctype" />
      </joined-subclass>
      <joined-subclass  name="ChequePayment"  table="Cheque_table">
         <key  column="P_id" />
         <property  name="chequeType"  column="chtype" />
      </joined-subclass>
   </class>
</hibernate-mapping>
```

another table in this inheritance use this reg

**** <joined-subclass ... > → inform to Hibernate this indicates Foreign key column

Hibernate will automatically copies the primary key value into Foreign key.

TablePerSubclass

- Payment.java
- CreditCardPayment.java
- ChequePayment.java
- InsertClient.java
- Payment2.hbm.xml
- hibernate.cfg.xml
- *.class

For this example refer pageno: 10 appn (6) of the Hundred.

Create the following tables at DataBase:-

```
SQL> create table payments (payid number(5) primary key, amount number(9,2),
                            pdate date);

SQL> create table credit_table (pid number(5), cctype varchar2(10));

SQL> Create table cheque_table (P_id number(6), chtype varchar2(10));
```

```
SQL> Select * from payments;
SQL> select * from credit_table;
SQL> select * from cheque_table;
```

SQL> select * from payments;

payID	Amount	pdate
1	5000	08-mar-2013
2	8000	08-mar-2013

select * from Credit-table;

PID	CCTYPE
1	VISA

select * from cheque_table

PID	CHTYPE
2	ORDER

③ Table Per Concreate class hirarchy :-

→ In this type of Inheritance hirarchy, hibernate uses a separate table for storing the generalized and specialized attributes of a derived class.

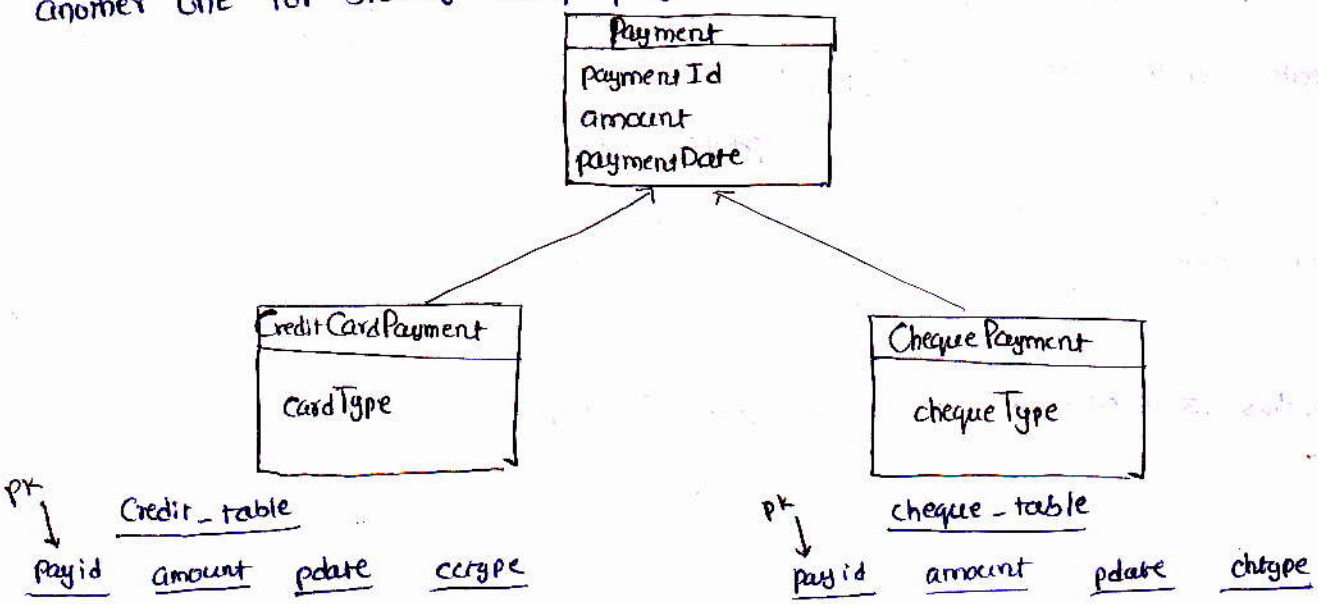
→ In this type of Inheritance mapping, hibernate uses a separate table in DataBase for each Derived class of the hirarchy.

→ In this Type of Inheritance mapping also, descriminator column is optional.

→ In a mapping file of Hibernate, we need to configure <union-subclass> tag for Configuring TablePer Concreate class hirarchy.

<subclass>	⟶ Table per class
<joined-subclass>	⟶ Table per subclass
<union-subclass>	⟶ Table per Concreate class

Example:- we have a Base class called payment with two derived classes CreditCardPayment and ChequePayment. In TablePerConcreate class hirarchy, hibernate uses two tables in DataBase, one for storing CreditCardPayment objects another one for storing ChequePayment objects.



payment3.hbm.xml

```
<h-m>
    <class  name = "Payment" >
     <id   name = "paymentId"   column = "payid" />
          <property   name = "amount" />
          <property   name = "paymentDate"   column = "pdate" />
****<union-subclass   name = "CreditCardPayment"   table = "credit_table" >
          <property   name = "cardType"   column = "cctype" />
    </union-subclass >
***<union-subclass   name = "ChequePayment"   table = "cheque_table" >
          <property   name = "chequeType"   column = "chtype" />
    </union-subclass >
  </class>
</h-m>
```

TablePerConcreateClass

- Payment.java
- CreditCardPayment.java
- ChequePayment.java
- InsertClient.java
- payment3.hbm.xml
- hibernate.cfg.xml
- *.class

For this example refer page: (11) an appn (7).

```
SQL> create table credit_table (payid number(5) primary key, amount number(9,2),
        pdate date, cctype Varchar2(10));
SQL> create table cheque_table (payid number(5) primary key, amount number(9,2),
        pdate date, chtype Varchar2(10));
```

```
SQL> Select * from credit_table;
```

payid	Amount	PDATE	CCTYPE
1	5000	05-MAR-13	VISA

```
SQL> select * from cheque_table;
```

Pay Id	Amount	PDATE	chtype
2	9000	05-MAR-13	ORDER

In this 3 inheirtance better to choose Table per Concreate class

HQL
(Hibernate Query Language)

09/03/2013

Bulk Operations in Hibernate :-

→ While working with Hibernate, operations on objects in hibernate can be done in two ways. that is 1) Single row operations and 2) Bulk operations.

→ While performing Single row operation, the operation will be effected on a Single row of the DataBase.

→ Incase of BulkOperations, the operations are effected on multiple rows at a time on DataBase

→ Hibernate Framework has provided the following 3 techniques for performing Bulk operations.

1. Hibernate Query Language
2. Criteria API
3. Native SQL

HQL :-

→ An Object oriented form of SQL is called HQL.

→ Object Oriented form is nothing but a query which is constructed by replacing Tablename with pojo class name and columns names with Pojo class Variable names.

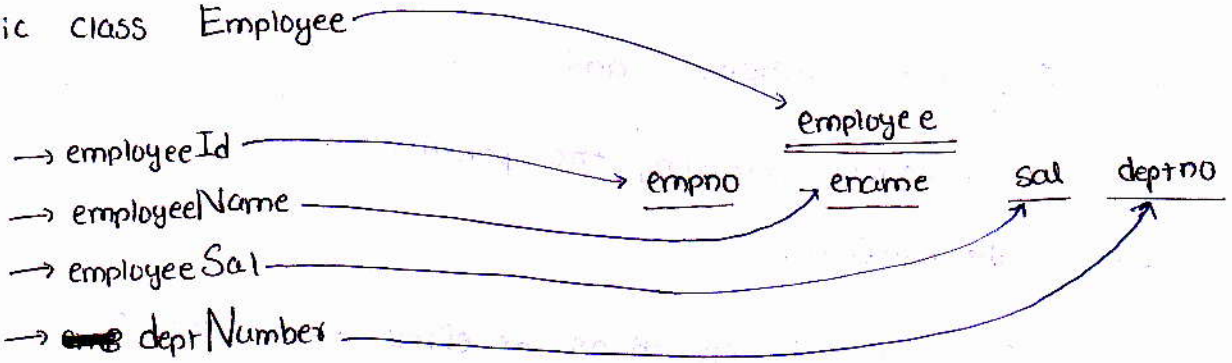
→ The big difference blw writing SQL and HQL queries is, SQL queries are DataBase dependent, but HQL queries are DataBase Independent.

→ We can perform both Select and non-select operations on a DataBase using HQL.

Select Operation in HQL:-

→ Constructing a select operation in HQL is almost equal to constructing Select operation in SQL.

→ public class Employee
{
→ employeeId
→ employeeName
→ employeeSal
→ deptNumber



Eg①:-

sql → select * from employee

hql → select e.* from Employee e

(or)

from Employee e

→ While using above HQL query, we can select all employees from the DataBase at a time. In hibernate terminology, if we load compleet details of a row then it is called loading Full object from the DataBase.

→ while loading Full object, we can directly start an HQL query with "from" keyword.

Eg②:-

sql → select ename, sal from employee

hql → select e.employeeName, e.employeeSal from Employee e

→ by using the above HQL query, while loading the partial details of employee from the DataBase. It is called loading partial objects.

→ while loading partial objects, we can't start an HQL query with "from" keyword.

Eg③:-

sql → select * from employee where deptno = 20

hql → select e.* from Employee e where e.deptNumber = 20

(or)

from Employee e where e.deptNumber = 20

→ By using the above HQL query, the employees working in 20th department are selected. In the above query, Full Objects are loaded from the DataBase, so we started HQL query with "from" keyword.

Eg(4):-

sql ⟶ select ename, sal from employee where deptno = ?

hql ⟶ select e.employeeName, e.employeeSal from Employee e where e.deptNumber = ?

```
Select e.employeeName, e.employeeSal from Employee e where e.deptNumber = :p1
```

→ by using the above HQL query, we are loading partial employee objects, by depending on deptNumber given at runtime

→ in an HQL query, to put a runtime value, we can insert either a positional parameter (or) a Named parameter into the query.

Positional parameter (?)

Named parameter (:<name>)

→ For executing HQL queries, we need a Query reference

→ to get a Query reference, we need to call createQuery( ) of Session Interface

→ Session Interface in Hibernate is the main runtime interface b/w a programmer and Hibernate. Because all the methods required for a programmer to talk with hibernate are given by Session Interface of hibernate.

Syntax:-

```
Query qry = session.createQuery("hql command");
```

After constructing a Query reference, we need to call one of the following two methods for executing that query.

① list( ) ⟶ for select operation

② executeUpdate( ) ⟶ for non-select operation

→ Query is an interface of org.hibernate pkg and its reference points to its Implimentation class object.

→ The implimentation class of Query Interface is QueryImpl

→ There are 3 ways of executing a select operation of HQL.

1st way :-

→ In this way we are selecting Full objects from DataBase

→ While selecting Full objects from DataBase by using HQL query, internally hibernate performs the following.

① Hibernate reads the rows from a table and stores them in a ResultSet object.

② Hibernate reads each row of ResultSet and stores the data in a POJO class object

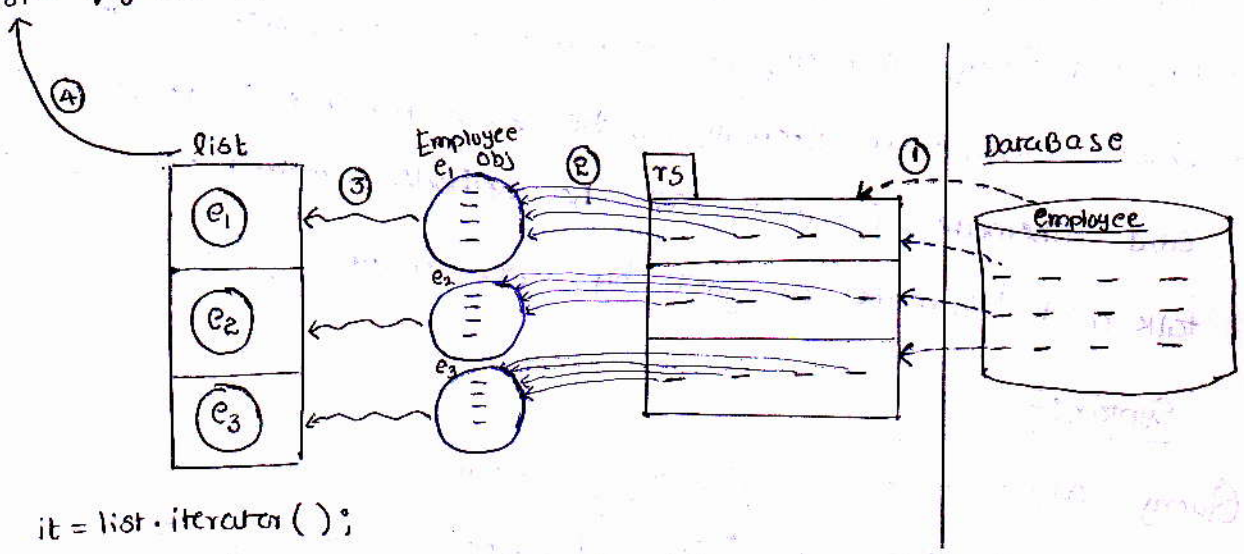
③ Hibernate stores all POJO objects in a Collection of type ArrayList(java.util.List)

④ Finally hibernate returns the Collection reference back to our Java program.

→ After obtaining a refernce of List Collection then we need to Iterate that collection for reading one by one object from the Collection.

example :-

```
Query qry = session.createQuery("from Employee e");
List list = qry.list();
```




```
Iterator it = list.iterator();
While(it.hasNext())
 {
   Object o = it.next();
   Employee e = (Employee) o;
   S.o.p(e.getXxx());
 }
```

Way-2:-

→ In this way, we are reading (or) loading partial objects from the DataBase using HQL query.

→ While reading partial objects, Hibernate internally performs the following

① Hibernate reads the Data from a DataBase table and stores it in a ResultSet object

② Hibernate stores each row data of ResultSet into an Object[]

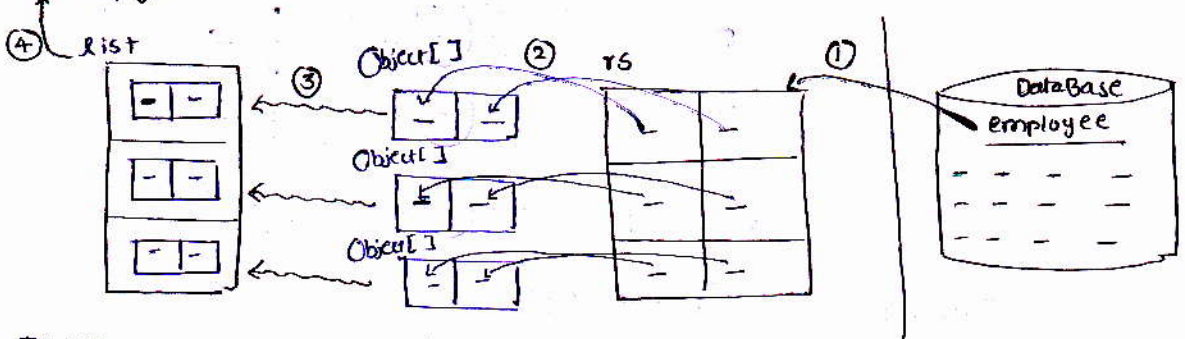
③ Hibernate stores Object[] into java.util.List collection

④ Finally Hibernate returns the list collection back to our Java program.

→ While iterating the collection, we need to typecast each element of the collection into Object[]

Example:-

```
Query qry = session.createQuery("select e.employeeName, e.employeeSal from Employee e");
List list = qry.list();
```




```
Iterator it = list.iterator();
While(it.hasNext())
{
Object[] row = (Object[]) it.next();
S.o.p(row[0] + " " + row[1]);
}
```

Way 3:-

→ In this way, we are reading a single property of an Object (or) objects from the DB using HQL query.

→ While executing this type of query then internally hibernate performs the following

① Hibernate reads the data from table and stores in ResultSet.

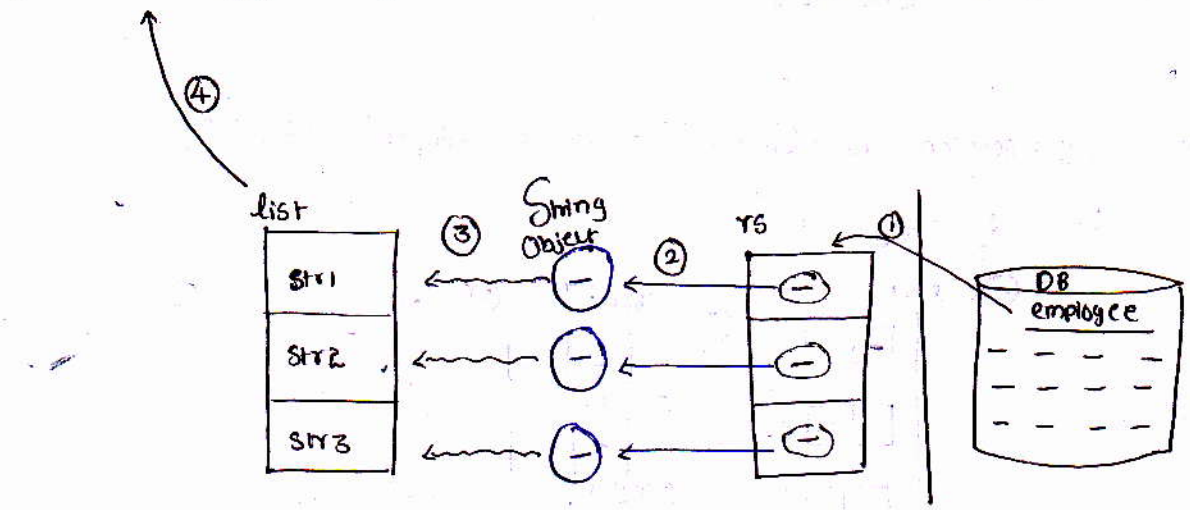
② Hibernate finds the property type using ResultSet metadata and stores the value of each raco of ResultSet into that Particuler type of Object.

③ Hibernate stores the Objects into a collection of type java.util.List

④ Finally, Hibernate returns the collection back to our java program.

→ While iterating the collection, we typecast each element of the collection into that perticuler Object type.

```
Query qry = session.createQuery("select e.employeeName from Employee e");
List list = qry.list();
```




```
Iterator it=list.iterator();
While(it.hasNext())
 {
String s1=(String)it.next();
  S.o.p(s1);
 }
```

HqlSelect
↳ Employee.java
employee.hbm.xml
hibernate.cfg.xml
SelectClient.java
x.class

refer page no (11) appn no: (8) of handoout.

Non-Select operations using HQL :-

11/03/2013

→ Using HQL, we can perform both select and non-select operations on a DataBase

→ The way of constructing a non-select operation is again similler to an SQL non-select operation. But we replace a Tablename with a POJO class name and columns names with POJO class variables names.

→ For executing a non-select operation of HQL, we need to call executeUpdate()

→ While executing a non-select operation, we need to put a Transaction because hibernate needs a Transaction for non-select operations.

for example, if we want to perform an Update operation for Updating the salary of employees, the following code is required

```
Query qry = session.createQuery("update Employee e set e.employeeSal=800
                                 where e.deptNumber = 20");
Transaction tx = session.beginTransaction();
int k = qry.executeUpdate();
tx.commit();
S.O.P(k+" rows updated");
```

HQL Insert :-

→ HQL Insert Operation is not a straight forward operation. It means by using HQL, we cannot perform direct insert operation into a DataBase Table.

→ in HQL Insert operation is nothing but copying one object data into another object.

→ While Constructing an Insert operation in HQL, we use insert & select queries combinely.

→ While Constructing an Insert operation of HQL, the attributes of destination Object and the attributes of Source Object, must be individually Specified in the Query.

no.of variables are equal to the source class and destination class other wise error will come.

For example :-

```
Query qry = session.createQuery("insert into TestEmployee(empId, empName, empSal,
                                                          deptno)
                                 Select e.employeeId, e.employeeName, e.employeeSal,
                                 e.deptNumber from Employee e");

Transaction tx = session.beginTransaction();
int k = qry.executeUpdate();   // k → no.of objects inserted (from one table to another table is inserted nothing but copied)
tx.commit();
```

→ While using the above query objects of Employee pojoclass are copied to TestEmployeePojo class.

→ In HQL insert operation is nothing but copying the rows of one table into another table of DataBase.

refer pageno (12) appn (9) of the Handout.

HqlInsert
- Employee.java
- TestEmployee.java
- employee.hbm.xml
- testemployee.hbm.xml
- hibernate.cfg.xml
- InsertClient.java
- *.class

```
SQL> Select count(*), sum(sal), max(sal) from employee
HQL> select count(*), sum(e.employeeSal), max(e.employeeSal) from Employee e.
```

Criteria API

→ Criteria API in Hibernate is used for loading (selecting) entities (objects) from the DataBase

→ Using this CriteriaAPI, only Select operation can be done. It can't perform any insert, Update (or) delete operations (non-select) on DataBase.

→ In CriteriaAPI, hibernate internally uses reflection mechanism for reading metadata of a pojoclass. so, we need to pass Class object of a pojo class for Hibernate.

→ While creating Criteria reference, we donot pass any query as a parameter. Instead of query, we pass Class object.

→ To obtain a Criteria refernce, we need to call ~~Create~~ createCriteria() of Session interface

```
Class clazz = Employee.class;
```

→ `Criteria crit = session.createCriteria(Class clazz);`

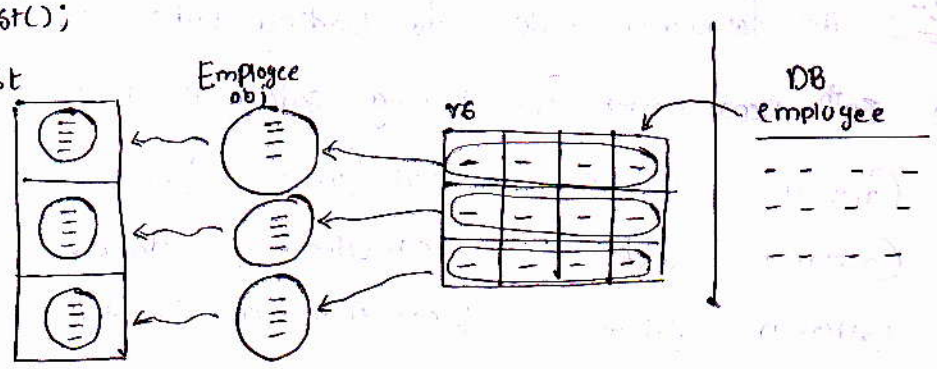
→ Criteria is prefered in hibernate, when entities (objects) are need to be loaded with multiple condition (Criterias)

→ After getting a reference of criteria, we need to call list() method, for executing the Select operation using Criteria.

```
List list = crit.list();
```

→ The following is a code using criteria, for loading Full objects (or) Full Object from the DataBase.

```
Criteria crit = session.createCriteria(Employee.class);
List list = crit.list();
```




```
Iterator it=list.iterator();
while (it.hasNext())
{
  Employee e=(Employee) it.Next();
  S.o.p(e.getXxx());
}
```

Adding a Condition to Criteria :-

→ While loading entities (objects) from a DataBase using criteria, if we want to apply a condition for loading objects then we need to create a Criterion object and then we need to add Criterion object to Criteria Object

→ Criterion is an interface and we can get its reference through Restrictions class.

→ Restrictions class is not a Derived class of Criterion interface. Restrictions class is a POJO class.

→ The Restrictions class contains all static methods and each static method returns a Criterion object.

→ The only Similarity b/w Criterion and Restrictions, both are from same package called org.hibernate.Criterion

Eg:-①:-

The following code is for loading entities of Employee POJO class, whose name contains a letter "S", using Criteria.

```
Criteria  crit = session.createCriteria(Employee.class);
Criterion condition = Restrictions.ilike("employeeName", "%S%");
          crit.add(Condition);
List      list = crit.list();
Iterator  it = list.iterator();
```

Eg:-②:- The following code is for loading entities of Employee POJO class, working in 30th Department and having salary as morethan 5000, using Criteria.

```
Criteria  crit = session.createCriteria(Employee.class);
Criterion condition1 = Restrictions.eq("DeptNumber", 30);
Criterion Condition2 = Restrictions.gt("employeeSal", 5000);
Criterion Condition3 = Restrictions.and(condition1, Condition2);
          Crit.add(Condition3);
List      list = crit.list();
```

To put more than one condition, first we need to make the two conditions as single condition by calling and(-)(or) or(--) methods and then the final condition we need to add for Criteria.

Eg③:- The following code is for loading entities of Employee POJO class, by applying the conditions as department number 30, salary should be greater than 5000 and employee name should contain "S" letter

```
Criteria   crit = session.createCriteria(Employee.class);
Criterion  condition1 = Restrictions.eq("deptNumber", 30);
Criterion  condition2 = Restrictions.gt("employeeSal", 5000);
Criterion  condition3 = Restrictions.like("employeeName", "%S%");

Criterion  condition4 = Restrictions.and(condition1, condition2);
Criterion  condition5 = Restrictions.and(condition4, condition3);

crit.add(condition5);
List  list = crit.list();
.....
.....
.....
```

Adding Sorting Order to Criteria :-

→ While loading entities (objects) from a DataBase using Criteria, it is possible to apply some sorting order for loading the objects.

→ If we want to apply a sorting order then we need to add Order object to the criteria by calling addOrder() method.

→ We can obtain an Object of Order class by calling static method of Order class only. Order is a class having two static methods asc(-) and desc(-) methods. This Order class is from org.hibernate.Criterion package.

→ The following code is for loading entities of Employee POJO class, in Descending Order of their salaries.

```
Criteria  crit = session.createCriteria(Employee.class);
Order     order = Order.desc("employeeSal");
          crit.addOrder(order);
List      list = crit.list();
          ------
          ------
```

Loading partial Objects using Criteria :-

→ While loading objects from a DataBase using Criteria, it is possible for loading particular properties of objects (partial objects), instead of complete objects (full objects) from a DataBase. This is called loading partial objects.

→ For loading partial objects, we need to create a Projection object for each property.

→ We can obtain a projection object by calling static methods of Projections class.

→ Projections class is a factory for producing Projection objects.

NOTE:- Similarly Restrictions class is a factory for producing criterion objects.

→ Loading partial objects using Criteria involves the following 3 steps

① Create a Projection Object for each property.

② Add Projection objects to ProjectionList

③ setProjectList to Criteria.

→ Finally we need to call list() for executing the Criteria.

→ At the time of Iterating that collection, we need to type cast the result into an Object[].

→ The following example code is for Loading two properties of Employee objects from the DataBase using Criteria.

```
Criteria crit = session.createCriteria(Employee.class);
//Step-I
Projection p1 = Projections.property("employeeName");
Projection p2 = Projections.property("employeeSal");
//Step-II
ProjectionList plist = Projections.projectionList();
  plist.add(P1);
  plist.add(P2);
//Step-III
crit.setProjection(plist);

List list = crit.list();
Iterator it = list.iterator();
While(it.hasNext())
{
  Object row[] = (Object[]) it.next();
  S.o.p(row[0]+" "+row[1]);
}
```

⊛ Loading a Single property of an Object using Criteria :-

→ While loading a Single property, we no need of using ProjectionList

→ Incase of loading a Single property, we can directly add a projectionObject to Criteria.

→ The following code is for loading a Single property called employeeName from DataBase using Criteria.

```
Criteria crit = session.createCriteria(Employee.class);
Projection p = Projections.property("employeeName");
  crit.setProjection(P);
  List list = crit.list();
Iterator it = list.iterator();
  While(it.hasNext())
  {
```

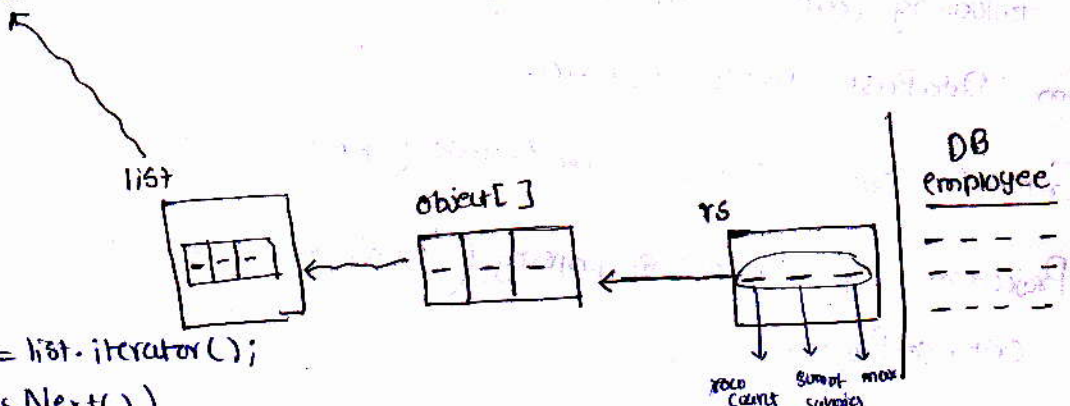
```
String str = (String) it.next();
    S.o.p(str);
}
```

→ In hibernate, Projections concept is also used, when we want to find any aggregate functions result using criteria.

→ Finding aggregate results is nothing but like Sum of Salaries, count of Objects (or) max of salaries etc...

The following sample code is for finding rowCount, sum & maximum of Salaries of employees using Projections of criteria.

```
Criteria crit = session.createCriteria(Employee.class);
//1
Projection P1 = Projections.rowCount();
Projection P2 = Projections.sum("employeeSal");
Projection P3 = Projections.max("employeeSal");
//2
ProjectionList plist = Projections.projectionList();
 plist.add(P1);
 plist.add(P2);
 plist.add(P3);
//3
 crit.setProjection(plist);
 List list = crit.list();
```




```
Iterator it = list.iterator();
 if(it.hasNext())
 {
   Object row[] = (Object[]) it.next();
   S.o.p(row[0] + " " + row[1] + " " + row[2]);
 }
```

Pagination methods in Criteria :-

→ While loading entities of a POJO class using Criteria, it is possible to load the objects of a perticuler range, instead of loading all objects from a DataBase.

→ Reading some part of data from a huze amount of data is called applying pagination technique.

The following are the two methods used for pagination in criteria

1) setFirstResult(15) ⟶ it starts from 16th record onwards to read th data
2) setMaxResults(5) ⟶ 16th record to 20th records toread bcz Max result 5

For Eg:-

```
Criteria crit = session.createCriteria(Employee.class);
    crit.setFirstResult(5);          reads the details of Employees for 6,7 &8 records
    crit.setMaxResults(3);
  List list= crit.list();
  Iterator it = list.iterator();
   While(it.hasNext())
    {
   Employee e =(Employee)it.next();
     S.o.p(e.getEmployeeId()+" "+e.getEmployeeName()+" "+e.getEmployeeSal()+"
                                                    +e.getDeptNumber());
    }
```

GitTest
- Employee.java
- employee.hbm.xml
- hibernate.cfg.xml
- CritClient.java
- *.class

refer page NO: (15) appn (10)

Q:-
Ans: choose HQL when the Queries are static, non-select operations & choose Criteria when the Queries are dynamic Queries, select operations on dynamic

NativeSQL

Q:- When to choose HQL and When to choose Criteria in Hibernate?

Ans:- When we want to perform a Select operation and when we want to apply multiple conditions i.e. we want to make the Query as dynamic then it is recomended to use Criteria.

→ When we want to perform non-select operations (or) when we want to execute static Queries then it is recommended to use HQL.

13/08/2013

→ NativeSQL is a consept of Hibernate, through which, we can execute direct SQL operations on a DataBase.

→ with NativeSQL, we can perform both select and non-select operations on a DataBase

→ In NativeSQL, we directly type SQL Queries, so NativeSQL is a DataBase dependent technique.

→ The following of the two major reasons for using NativeSQL in an Hibernate application.

① While migrating an existing JDBC application into an Hibernate appn, we use NativeSQL. Because a JDBC application contains SQL commands and NativeSQL also allows SQL commands.

② In order to call a "procedure" of a DataBase from an Hibernate application, we need NativeSQL.

→ For executing SQL Queries on a DataBase from an Hibernate application, we need a reference of org.hibernate.SQLQuery interface.

→ we can obtain a reference of SQL Query Interface, by calling createSQLQuery() method of Session(Interface).

Syntax:-

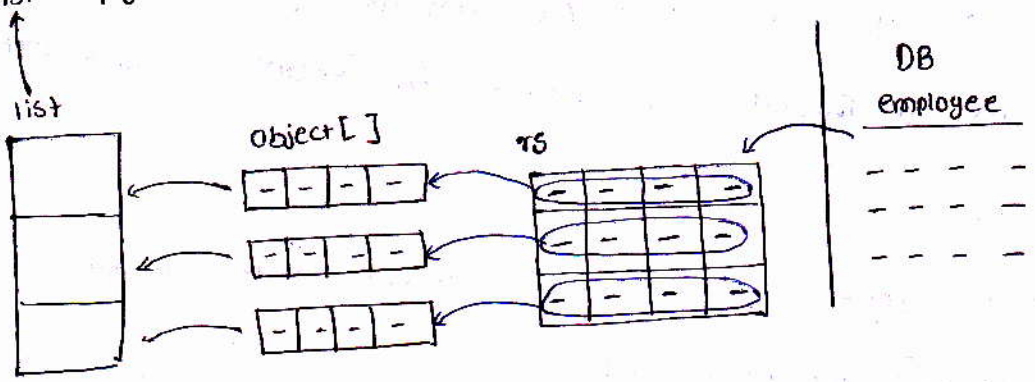
```
SQLQuery  qry = session.createSQLQuery("sql command");
```

→ While selecting a Full Objects (or) a Full Object from a DataBase using NativeSQL, internally Hibernate converts each row of ResultSet into an Object[], but not into a pojo class object. because in NativeSQL, we type directly SQL Command and it doesnot contain a pojo class name in it.

For example:-

```
SQLQuery  qry = session.createSQLQuery("select * from employee");

List  list = qry.list();
```




```
Iterator  it = list.iterator();
 while (it.hasNext())
  {
Object  row[] = (Object[]) it.next();
S.o.p(row[0]+" "+row[1]+" "+row[2]+" "+row[3]);
  }
```

→ While executing a Native SQL query for loading a Full Object (or) Objects from a DataBase, then we can inform the hibernate that Convert each row of ResultSet into a pojo class object, instead of an Object[]

→ To inform the Hibernate that Convert internally each row into a pojo class object, we call addEntity() method. To this method we pass class object of a pojo class as a parameter.

For example:-

```
SQLQuery  qry = session.createSQLQuery("select * from employee");
```

```
qry.addEntity(Employee.class);
List list = qry.list();
Iterator it = list.iterator();
while(it.hasNext())
{
  Employee e = (Employee)it.next();
  S.o.p(e.getXxx());
}
```

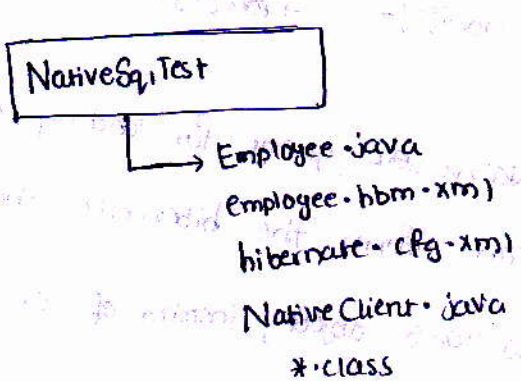
Executing a non-select operation :-

→ We call executeUpdate() for executing a non-select operation on a DataBase

→ While executing a non-select operation, we begin a Transaction. Because, in Hibernate a Transaction is required for executing non-select operations.

Forexample:-

```
SQLQuery qry = session.createSQLQuery("update employee set sal=? where empno=?");
qry.setParameter(0, 8000);
qry.setParameter(1, 7788);
Transaction tx = session.beginTransaction();
int k = qry.executeUpdate();
tx.commit();
```



refer page(16) appn (11) of handout.

Q:- What is the difference b/w HQL and NativeSQL?

Ans:- ① In case of HQL, we use HQL queries, but in NativeSQL we use SQL queries.

② HQL Queries are DataBase Independent, but SQL Queries are DataBase dependent.

③ Through HQL, we can't call a procedure or a function of a DataBase, using NativeSQL we can call a procedure or a function of a DataBase

Calling a procedure or a Function :-

→ In Hibernate, we have a limitation for calling a procedure or a function

→ The limitation is, a procedure can be called from Hibernate, if "it does not return any values". It means a procedure should not contain either out (or) inout parameters.

→ We can call a Function of a DataBase from Hibernate, only "if that function is returning a Cursor."

→ While Working with either JDBC or hibernate, it is not possible to create a Procedure or a function in DataBase from a Javaprogram. It is only possible to call a procedure or a function from a Javaprogram.

14/03/2012

→ To call a procedure or a function when it is returning any no. of values or any Type of values then we can use JDBC code in Hibernate for calling such type of procedures (or) functions

Calling a procedure (or) a function from Hibernate can be done in the following 2 ways.

① Using Hibernate without a JDBC code explicitly.

② Using Hibernate with JDBC code explicitly.

Calling a Procedure from Hibernate :-

→ If u want to call a procedure from Hibernate by without using JDBC code then that procedure should not return any values.

→ we can call a procedure by constructing a reference of SQLQuery like the following

```
SQLQuery qry = session.createSQLQuery("{call proname(params)}");
```

The following procedure in oracle takes two inparameters and updates the given salary for an employee.

```
SQL> ed file1
     Create or replace procedure testpro(eno in number, salary in number)
     is
      begin
       update employee set sal = salary where empno = eno;
    end;
```

→ compiling of procedure

```
SQL> @X1
This line to execute ← 7 /
     Procedure created
```

The following Hibernate application is for calling the above procedure.

Procedure Call

→ hibernate.cfg.xml → mapping resource tag is removed from this file.
client.java
*.class

```
// Client.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public class Client {
    psvm(String[] args)
    {
SessionFactory factory = new Configuration().configure().buildSessionFactory();
 Session session = factory.openSession();
SQLQuery qry = session.createSQLQuery("{call testpro(?,?)}");
```

```
        qry.setInteger(0,7788);
        qry.setInteger(1,8800);
    Transaction tx = session.beginTransaction();
        qry.executeUpdate();
        tx.commit();
        session.close();
        factory.close();
        S.O.P("procedure executed");
    }
}
```

Calling a Function from Hibernate :-

→ To call a Function from Hibernate by without using JDBC code then the following 3 conditions should be satisfied

① a Function of DataBase must return a cursor

② The cursor must return (or) must contain one or more rows.

③ That function must call using Named Queries concept.

Named Query :-

→ While Developing an Hibernate application, if same HQL Query (or) SQL Query is required for multiple times then instead of retyping the same Query again and again, we can use Named Queries Concept of Hibernate.

→ Named Queries are two types

① Named HQL query

② Name Native SQL query.

→ In order to reuse a query for multiple times in an Hibernate application then we need to configure that query in mapping file, by putting some name for the query.

→ To configure HQL query in a mapping file then the tag is <query>.

Similarly, to configure SQLQuery the tag is <sql-query>

NOTE:- The <query> or <sql-query> tag should be added at outside of <class> tag.

For example :-

Employee.hbm.xml

```
<h-m>
  <class name="Employee" table="employee">
    .....
    ....
    ...
  </class>
  <query name="q1">
    from Employee e where e.deptNumber = ? ⟶ HQL query
  </query>
  <sql-query name="q2">
    Select * from employee where sal > ? ⟶ SQL query.
  </sql-query>
</h-m>
```

→ To execute either a Named HQL query (or) a Named SQL query, we need to Obtain Query reference by calling getNamedQuery() method of session Interface.

```
Query qry1 = session.getNamedQuery("q1");   └→ HQL query
Query qry2 = session.getNamedQuery("q2");   └→ SQL query.

qry1.setParameter(0,20);   └→ deptnumber
List list = qry1.list();
  └→ employee pojoclass object bcz Full object

qry2.setParameter(0,2000);   └→ salary
List list = qry2.list();
  └→ Object[] objects bcz it is SQL query.
```

The following function is going to return a cursor with a set of Employees

```
SQL> ed x2
     create or replace function emp_fun
     return sys_refcursor is
      c1 sys_refcursor;
     begin
      open c1 for select * from employee;
      return c1;
     end;
SQL> @x2
     /
     Function created
```

drop function emp_fun;

→ For calling the above function from Hibernate, we need to Configure a NamedNativ SQL query in a mapping file. like the following

```
<sql-query name="q1" callable="true">
 <return class="Employee" alias="e" />
  {?= call emp_info()}
</sql-query>
```

Function
- employee.hbm.xml
- hibernate.cfg.xml
- Employee.java
- client.java

refer page no. (17) in Handout appn (12)

In the above, <return> tag is used for informing the hibernate to convert each row of ResultSet into POJO class object.

→ If we donot Configure <return> tag then the Hibernate Converts each row of ResultSet into an Object[].

→ In a client application of hibernate the following code is required for calling the function, which is Configured in the mapping file

```
Query qry = session.getNamedQuery("q1");
List list = qry.list();
```

→ when we call list() method then the following steps are executed.

① Hibernate calls the function and obtains a cursor object from it.

② Hibernate stores the cursor data into a ResultSet object of jdbc

③ Hibernate stores the data of each row of Resultset into POJO class object.

④ Hibernate stores the POJO class objects into a collection of type List.

and finally hibernate returns the Collection object back to a Java program.

Calling a procedure / Function in Hibernate using Jdbc code :-

→ Session interface of hibernate has provided a method doWork(), which is used to execute the JDBC code written in an Hibernate applications.

→ In Hibernate, to write JDBC code then we need to implement org.hibernate.jdbc.Work interface

→ Work interface has provided an Abstract method called execute() and we should provide implementation for execute() method by writing JDBC code.

→ For example :-

```
public class MyWork implements Work
{
    public void execute(Connection con) throws Exception
    {
        //Write jdbc code
    }
}
```

→ While calling procedures and functions from hibernate using jdbc code then there are no restrictions when calling the procedures and functions.

→ in execute() method of Work interface, we use CallableStatement of jdbc for calling a procedure (or) a function.

The following procedure is for calculating bonous by depending on salary of an Employee. This procedure takes empno has input then returns bonous as output.

```
SQL> ed temp2          (temp2 → File name)
Create or replace procedure bonous_pro(eno in number, bonus out number)
is
  empsal number(9,2);   (empsal → to create variable)
begin
  Select sal into empsal from employee where empno = eno;
```

```
if empsal between 0 and 2000 then
    bonus := 500;
elsif empsal between 2001 and 5000 then
    bonus := 1000;
else
    bonus := 1500;
end if;
end;
```

```
SQL> @temp2
/
procedure created
```

ProcedureCall
- hibernate.cfg.xml
- Client.java / .class

```
// Client.java
import org.hibernate.*;
import org.hibernate.cfg.*;
import org.hibernate.jdbc.*;
import java.sql.*;
import java.util.*;

public class Client
{
    p s v m(String args[])
    {
    SessionFactory factory = new Configuration().configure().buildSessionFactory();
     Session session = factory.openSession();
    Scanner s = new Scanner(System.in); → to give input for runtime dynamically
     S.o.p("enter empno");
```

in anonymous class we inside use eno in anonymous class so, we define final

```
    final int eno = s.nextInt();
    session.doWork(new Work()
                    {
                    public void execute(Connection con) throws SQLException
                    {
CallableStatement cstmt = con.prepareCall("{call bonus_pro(?,?)}");
```

→ jdbc code throws SQLException so, we write this

```
//register Out parameter
cstmt.registerOutParameter(2, Types.INTEGER);
    //set in parameter
    cstmt.setInt(1, eno);
                  to ser(or)write eno
    //call the Procedure
    cstmt.execute();
  int bonus = cstmt.getInt(2);
                  to read the bonous
   S.o.p("Bonus=" +bonus);
  }//end of execute()
 }//end of Work implementation
); //end of doWork()
   session.close();
   factory.close();
 }
}
```

17/03/2013

Pagination with Servlet and Hibernate Integration :-

→ When there is a huze amount of data to be displayed on a brocoser, as part of a response for a given request then, if all the data is displayed at a time in the brocoser then an enduser looses consentration, while reading the huze amount of data at once, shown on a brocoser.

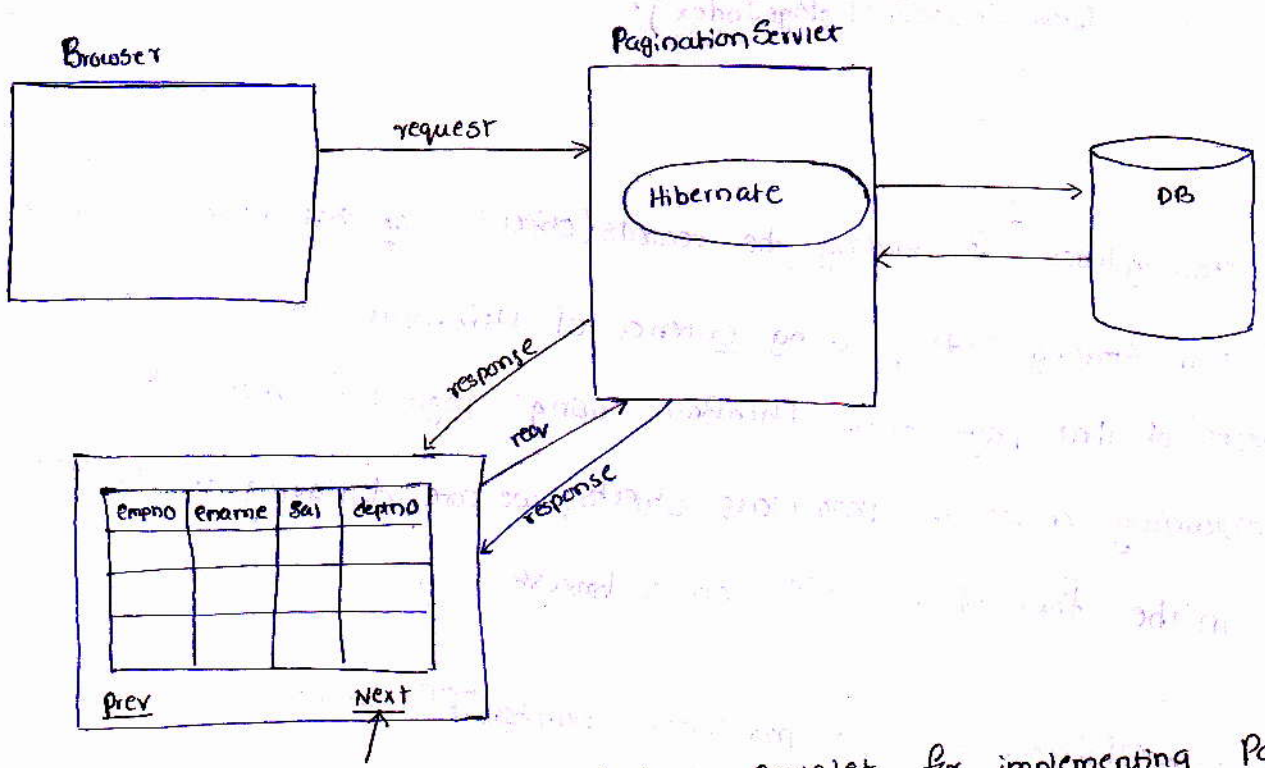
→ Inorder to overcome the above problem, we need devided the huze amount of data into multiple pages and we need to show page by page of data on a brocoser. This mechanism is called "pagination".

→ For example, if gmail, all the mails of Inbox are not displayed at a time on the browser. The mails are displayed in a page by page. It means pagination mechanism is applied for displaying the mails of an Inbox.

→ In most of web applications used in a realtime, (or) applied with pagination

The following example is a Servlet with Hibernate Integration, to display 3 Employee Objects details at once in a page, by deviding the Employees plages into multiple pages using pagination mechanism.

Flow:



The following logics are required in a servelet for implementing Pagination mechanism.

logic 1:-

→ In a servlet, we need to check weather a request is made for which page of response data.

→ In a Servlet, we are reading the value of a request parameter called pageIndex, to find the page requested by the client.

→ we are deciding page index as one, if a client is not passing pageIndex parameter.

→ If passed pageindex then we are taking the value of pageindex parameter as a pageIndex in our servlet.

```
String sPageIndex = request.getParameter("pageIndex");
if(sPageIndex == null)
{
    pageIndex = 1;
}
else
{
    pageIndex = Integer.parseInt(sPageIndex);
}
```

logic2:-

→ find the startingIndex for reading the records(objects) of that page from a DataBase.

→ After finding Starting Index, using Criteria of Hibernate we are reading the objects of that page from DataBase using pagination methods.

→ After obtaining a list of pojo class objects, we are displaying the employees details in the form of a table on a browser.

```
int si = numberOfRecordsPerPage * pageIndex - numberOfRecordsPerPage
Criteria crit = session.createCriteria(Employee.class);
crit.setFirstResult(si);
crit.setMaxResults(numberOfRecordsPerPage);
List list = crit.list();
Iterator it = list.iterator();
While(it.hasNext())
{
.....
}
```

logic 3:-

→ find totalnoofobjects (records) available in the DataBase table

→ in hibernate, while using Criteria, we need projections for finding an agreegate function result.

→ In this logic, we are storing the no.of objects available in DataBase in totalnoofrecords variable.

```
Criteria  crit = session.createCriteria(Employee.class);
Projection  p1 = Projections.rocoCount();
cri.setProjection(p1);
List  list = crit.list();
Iterator  it = list.iterator();
if(it.hasNext())
{
   Integer  i=(Integer)it.next();
   totalNumberOfRecords = i;
}
```

logic 4:-

→ Find no of pages required for displaying the total no of records

→ we devide the total no of records with no of records per page for finding the no of pages

→ If any additional records are exist then increment the no of pages by one.

```
noOfPages = totalNumberOfRecords / numberOfRecordsPerPage;
if( totalNumberOfRecords > (noOfPages * numberOfRecordsPerPage) )
{
   noOfPages++;
}
```

logic 5:-

→ ~~print~~ Prev and Next links hyperlinks on brocoser for PageNavigation.

→ pass a parameter pageIndex to a servlet cohen the link is linked

→ for previous link, pageIndex is currentPageIndex - 1 and for Next Link the page index is currentPageIndex + 1.

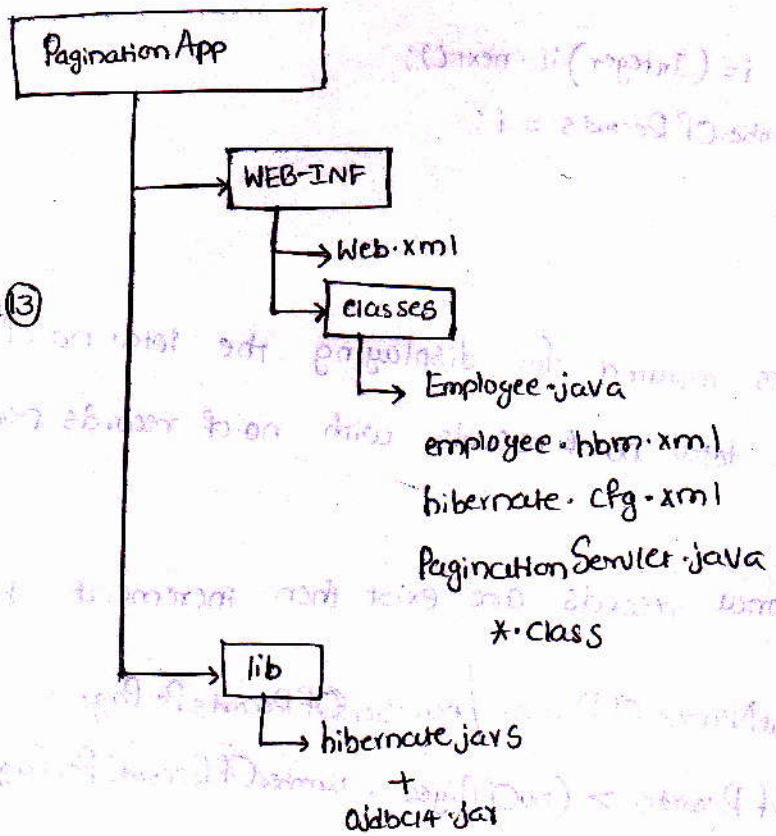
→ Previous link should not be displayed on FirstPage and Next Link should not be displayed on Lastpage.

```
if( pageIndex > 1 )
{
  pro.println("<a href=srv1?pageindex=" + (pageIndex - 1) + ">Prev</a>");
}
```

```
if(pageIndex < noOfPages)
{
pw.println("<a href=srv1?pageindex=" +(pageIndex+1)+ ">Next</a>");
}
```

Directory Structure :-

refer page No (18) application (3) of handout.



TO compile pagination Servlet, we need to set the following two jarfiles in the CLASSPATH

① servlet-api.jar

② hibernate3.jar

request:

http://localhost:2020/PaginationApp/ps

Relationships in Hibernate (Associations in Hibernate)

→ The data in a DataBase is stored in multiple tables and then some relation is assigned blw the Data, because of the following two reasons.

① In order to reduce ~~Data~~ Redundancy of the Data in DataBase tables

② To manage the operations on the data easyly.

for Example:- We have a table Hospital, where we want to store the Doctors and patients taken Oppointment for a doctor. so, the table is going to look like the following.

Hospital

Docid	DocName	specialization
101	ABC	MD
101	ABC	MD
-	-	-
-	-	-
-	-	-
102	XYZ	MS
102	XYZ	MS
-	-	-
-	-	-
-	-	-

Duplicate

Duplicated

pat-id	pat-name	Timeslot
1	X	9:00
2	Y	6:15
3	-	-
4	-	-
5	-	-
301	A	7:15
302	B	7:20
303	-	-
304	-	-
305	-	-

In the above table,

a doctor information is repeated for each patient information.

So, a Doctor information is duplicated in the table for multiple times.

So, it leads to DataRedundancy.

→ To solve the above DataRedundancy problem, we can devide the above table data into two tables. like the following

doctor_info

docid	docname	specialization
101	ABC	MD
102	XYZ	MS

patient_info

Pat_id	pat_name	timeslot
1	X	----
2	Y	----
301	A	----
302	B	----

In the above, the data is devided into two tables, there is no relation blw the data in two tables. So, the data stored in the DataBase is an Useless data.

→ So, when the data is devided across two tables then there should be some relation blw the data, to make it as useful data.

→ In the above, we have two options, to make the data as Usefull.

① We can copy a patient id into Doctor_info table.

② We can copy a doctor id into Patient_info table.

→ in the 1st case, for each patient id, again a Doctor information is repeated. it means still there is a DataRedundency.

doctor_info

docid	docname	specialization	pat_id
101	ABC	MD	1
101	ABC	MD	2
101	ABC	MD	3
=	=	=	4
=	=	=	5
=	=	=	6
=	=	=	7

duplicated.

→ To resolve this DataRedundancy problem, we need to copy the Doctor id into patient_info table. so that for each patient information, only a Doctor id is repeated. So, the Redundency problem is reduced.

Patient_info

Pat_id	Pat_name	timeslot	doc_id
1	X	....	101
2	Y	....	101
3	Z	....	101
301	A	....	102
302	B	....	102

finally the Redundency problem is reduced by copying a doctorid into a Patient information. So, According to the DataBase terminology, a referential Integrity is applied blw the data of two tables.

→ According to DataBase terminology, doctorid is copied to patient, so doc_info is a parent table (or) master table and Patient_info table is a child table (or) Detailed table.

in Hibernate

→ While working with hibernate for persisting this type of related data in DataBase, we create multiple POJO classes as Part of our application. and we assign a relationship b/w objects of the POJO classes. This Consept is called Association Mapping in Hibernate.

→ Hibernate supports the following 4 types of relationships b/w objects.

① one-to-many
② many-to-one
③ many-to-many
④ one-to-one

One-to-many (Uni-directional) relationship :-

→ To apply this one-to-many assossiation between the objects of two POJO classes in Hibernate, then the following changes are required

① while creating a parent POJO class, we need to add an additional property of type Collection. This Collection property is used to store many child objects in a parent object.

② we need to Configure the Collection Property in a parent class mapping. During this Configuration, we need to Configure the Foreign key and the relationship in the mapping file.

③ In a client application of hibernate, child objects are added to the parent like the following

ⓐ add a group of child objects to a collection object

ⓑ add collection object to the parent object

Example :-

→ We have two POJO classes in application called Vendor and Customer. The relationship between these two class objects is, one Vendor is associated to multiple customers. so the relationship is one-to-many.

→ In this Vendor and Customer relationship, Vendor class is a Parent class and Customer class is a child class.

→ The one-to-many relation between Vendor to Customer is going to use the following two tables in DataBase.

PK

VENDOR(Parent)

VID	VNAME

```
//Parent
Vendor.java
Public class Vendor
{
   private int vendorId;
   private String vendorName;
(1) private Set customers;
      setters & getters
}
```

Vendor.hbm.xml

```
<h-m>
   <class name="Vendor" table="vendor">
      <id ----/>
      <property ---/>
(2)   <set name="Customers">
         <key column="VENID" />
                  └→ FK
         <one-to-many class="Customer" />
                              └→ child class
   </set>
   </class>
</h-m>
```

PK FK

CUSTOMER (child)

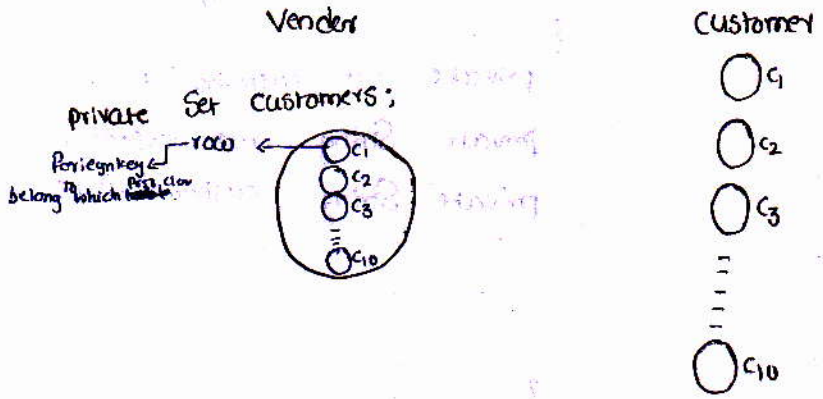
CUSTID	CUSTNAME	CUSTADDR	VENID

```
//child
Customer.java
public class Customer
{
   private int customerId;
   private String customerName;
   private String customerAddress;
      setters & getters
}
```

Customer.hbm.xml

```
<h-m>
<class name="Customer" table="Customer">
      <id ---- />
      <property ---/>
      <property ---/>
</class>
</h-m>
```

```
// Client.java
Public class Client
{
   P s v m (String[] args)
   {
      =
      =
③ a) Set set = new HashSet();
      set.add(c1);
      set.add(c2);
      set.add(c3);
   b) v.setCustomers(set);
      Transaction tx = session.beginTransaction();
        session.save(v);
        tx.commit();
   }
}
```



NOTE:- While working with relationships of hibernate, inorder to inform the hibernate that apply the operation done on one object on its relationship objects also, we need to add an attribute called cascade, while configuring the collection element in a mapping file.

for Eg:-

```
<set name="customers" cascade="all">
    <key column="venid" />
    <one-to-many class="Customer" />
</set>
```

If we donot add cascade attribute then the default value of cascade is "None". It means, by default hibernate doesn't transfer any operations on to the related objects.

→ In a DataBase, the vendor and customer tables can be created for this one-to-many relationship like the following.

```
SQL> Create table vendor (vid number(5) primary key, vname varchar2(20));
SQL> Create table customer (custid number(5) primary key, custname varchar2(10),
                            custaddr varchar2(10), venid number(5) references vendor(vid));
```

→ OneToMany

- vendor.java
- Customer.java
- Vendor.hbm.xml
- Customer.hbm.xml
- hibernate.cfg.xml
- InsertClient.java.
- *.class

```
//ParentPOJO
//Vendor.java
import java.util.*;
public class Vendor
{
  private int vendorId;
  private String vendorName;
  private Set customers;
    ==
    ==
}
```

```
//Customer.java
public class Customer
{
  private int customerId;
  private String customerName;
  private String customerAddress;
    ==
    ==
    ==
}
```

```
<!-- Vendor.hbm.xml -->
<h-m>
  <class name="vendor" table="vendor">
    <id name="vendorId" column="vid" />
    <property name="vendorName" column="vname" length="10" />
    <set name="customers" cascade="all">
      <key column="venid" />
      <one-to-many class="Customer" />
    </set>
  </class>
</h-m>
```

```
<!-- customer.hbm.xml -->
<h-m>
  <class name="Customer" table="Customer">
    <id name="customerId" column="custid" />
    <property name="customerName" column="custname" length="10" />
    <property name="CustomerAddress" column="custaddr" length="10" />
  </class>
</h-m>
<!-- hibernate.cfg.xml -->
        .
        .
        .
<mapping resource="vendor.hbm.xml" />
<mapping resource="Customer.hbm.xml" />
```

```
// InsertClient.java
import java.util.*;
impor org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
   p s v m (String[] args)
   {
  SessionFactory factory=neco
    Session      session = factory.openSession();

   //Parent Object
   Vendor  v=new Vendor();
   v.setVendorId (111);
   v.setVendorName ("IBM");
//child Object-1
Customer c1=neco Customer();
  c1.setCustomer Id (501);
  c1.set Customer Name ("INFY");
  c1.setCustomer Address ("HYD");
//Child Object-2
    Customer  c2=neco Customer()
  c1.set Customer Id (502);
  c1.set Customer Name ("TCS");
  c1.setCustomer Address ("HYD");
```

```
//child Object-3
Customer c3 = new Customer();
  c3.setCustomerId(503);
  c3.setCustomerName("VERIZON");
  c3.setCustomerAddress("US");
//add child Objects to java.util.Set
  Set s=new HashSet();
    s.add(c1);
    s.add(c2);
    s.add(c3);
//add java.util.set to Parent Object
  v.setCustomers(s);

Transaction tx = session.beginTransaction();
    session.save(v);
    tx.commit();
  Session.close();
  factory.close();
  }
}
```

```
SQL> Select * from Vendor;

VID   VNAME
111   IBM
```

```
SQL> select * from customer;

CUSTID   CUSTNAME   CUSTADDR   VENID
501      INFY       HYD        111
502      TCS        HYD        111
503      VERIZON    US         111
```

Select Operation in one-to-many :-

→ In one-to-many relationship, when a parent Object is loaded then hibernate automatically loads its child Objects are also from the DataBase.

→ While loading the child objects of a From parent Objet, we can inform the hibernate weather Lazy loading (or) early loading of child Objects is required through "lazy" attribute.

→ By default hibernate uses lazyloading, because the default value of "lazy" attribute is "true".

→ If we want to inform the hibernate that, apply earlyloading on child Objects then we should make the value of "lazy" attribute as "false"

→ In early loading, hibernate loads the child Objects along with the parent Object immediately. But in Lazyloading, first only parent Object is loaded and its child Objects are loaded on demand (when ever required)

```
<set name="customers" cascade="all" lazy="true/false">
                                          └default
        ||
        ||
</set>
```

lazy = "true" ⟶ lazyloading

lazy = "false" ⟶ earlyloading

```
//Step-1
Vendor  v=(Vendor) session.get(Vendor.class, 111);
//Step-2
Set  s= v.getCustomers();
//Step-3
Customer  c=(Customer) session.get(Customer.class, 503);
//Step-4
Transaction  tx = session.beginTransaction();
    s.remove(c);
    tx.commit();
```

→ When a child object is removed from a collection of child objects of a parent object then hibernate is going to remove the relationship between the parent object and the child object, by making Foreign key of that child object as "null"

→ After a relationship is removed blw a parent and a child object, hibernate is not going to remove that record from the DataBase.

→ If the above client application is executed then in Customer table of DataBase a Customer record with id "503" existing table with Foreign key as "null".

Customer

Custid	custname	custaddr	Venid
501	INFY	HYD	111
502	TCS	HYD	111
503	VERIZON	US	
504	CTS	HYD	111

→ Orphan record.

→ Eventhough the relationship is removed, but still the record is exist in the table. We call such record as a "Orphan record".

→ If we want to inform the Hibernate that remove orphan record from the DataBase table immediatly, whenever the relationship is removed blw the parent and child record then we need to modify a value of cascade attribute

as "all-delete-orphan".

```
<set name="customers" lazy="true" cascade="all-delete-orphan">
    ---
    ---
</set>
```

→ for cascade attribute, we have the following 6 values.

1) None (default)
2) all
3) all-delete-orphan
4) save
5) save-update
6) delete

→ Cascade attribute is used to inform the hibernate about which non-select operations should be transfered (applied (cascaded)) on relationship object of an Object

→ If "cascade = save" then Hibernate only cascades "insert" operation on child objects.

If "cascade = save-update" then Hibernate cascades "insert" and "update" operations on child Objects.

If "cascade = delete" then Hibernate cascades only "delete" operation on child Objects.

If "cascade = all" then Hibernate cascades insert, delete, update operations on child objects.

If "cascade = all-delete-orphan" then Hibernate cascades all operations on child objects and also deletes orphan records.

→ In one to many relation, if we want to remove the all child objects of a parent object then follow the below steps.

① load the parent object from the DataBase
② read all child objects of a parent object into collection
③ remove all objects from a collection with in a Transaction

```
//step-1
Vendor v = (Vendor) session.get(Vendor.class, 111);
//step-2
Set s = v.getCustomers();
//step-3
Transaction tx = session.beginTransaction();
s.removeAll();
tx.commit();
```

→ If you want to remove a parent object along its all child objects from the DataBase then we need to delete simply the parent Object. like the following

```
Vendor  v = (Vendor) session.get(Vendor.class, 111);
Transaction  tx = session.beginTransaction();
    session.delete(v);
    tx.commit();
```

Using java.util.List as a collection :-

→ When applying one-to-many relationship, for storing a group of child objects to a parent Object, in Parent class we need a collection of type either Set (java.util.Set) or List or Map.

→ If we use collection type as List then hibernate uses an additional column in child table for storing "indexes" of a child in the List into DataBase.

→ At the time of configuring <list> tag in a mapping file, we need to configure "listindex" also in the mapping file.

Eg: we have two POJO classes called Dealer and Product. we are applying one-to-many relation blw a Dealer and a Product (i.e. one dealer is assossicated with multiple Products.

→ In Product table of DataBase (child table), an additional column is required for storing the "list index"

→ The DataBase tables for this Example. are

dealer

PK	
did	dlname

product

PK			FK	index
pid	pname	price	dealid	li

OneToMany2

- Dealer.java
- Product.java
- Dealer.hbm.xml
- Product.hbm.xml
- hibernate.cfg.xml
- InsertClient.java
- *.class

// Parent

// Dealer.java

```
import java.util.*;
public class Dealer
{
   private int dealerId;
   private String dealerName;
   private List products;
     setter & getters
}
```

// child

// Product.java

```
public class Product
{
  private int productId;
  private String productName;
  private double price;
    setter & getters
}
```

Dealer.hbm.xml

```
<h-m>
 <class name="Dealer" table="dealer">
  <id name="dealerId" column="did" />
  <property name="dealerName" column="dlname" length="10" />
  (***) <list name="products" cascade="all">
        <key column="dealid" />
     (**) <list-index column="li" />
        <one-to-many class="Product" />
    </list>
  </class>
</h-m>
```

product.hbm.xml

// InsertClient.java

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

```
public class InsertClient
{
    p s v m (String[] args)
    {
SessionFactory factory = new Configuration().configure().buildSessionFactory();
Session session=

    //Parent Object
Dealer d=new Dealer();
d.setDealerId(101);
d.setDealerName("ABC");

//child Object-1
Product P1=new Product();
P1.setProductId(901);
P1.setProductName("X");
P1.setPrice(2000);

//child Object-2
Product P2=new Product();
P2.setProductId(902);
P2.setProductName("Y");
P2.setPrice(3000);

//child Object-3
Product P3=new Product
P3.setProductId(903)
P3.setProductName("Z
P3.setPrice(5000);

//Step-1 //add child Objects to java.util.List
List list=new ArrayList();
list.add(P1);
list.add(P2);
list.add(P3);

//step-2 //add java.util.List to parent
d.setProducts(list);

Transaction tx=session.beginTransaction();
session.save(d);
tx.commit();

Session.close();
factory.close();
}
}
```

```
select * from dealer;
```

DID	DNAME
101	ABC

```
select * from product;
```

PID	PNAME	PRICE	DEALID	LI
901	X	2000	101	0
902	Y	3000	101	1
903	Z	4000	101	2

<bag> tag :-

22/03/2013

→ When a collection is used of type java.util.List then we configure <List> tag in a mapping file. If we configure <list> tag then we also need to configure <list-index> tag inside it, for informing the hibernate about the column name in child table, in which the index of the child object is need to be stored

→ When a child object is stored in DataBase then hibernate also stores its index in a list along with that child object in the DataBase. But there is no use with the index of a child object stored in a DataBase.

→ In order to inform the hibernate that store only the child object, but not its index in a DataBase, we need to configure <bag> tag, instead of <list> tag in a hibernate mapping file.

→ For the above example, in dealer.hbm.xml file, instead of <list> tag we can configure <bag> tag like the following.

```
<bag  name="products"  cascade="all">
    <key  column="dealid" />
    <one-to-many  class="product" />
</bag>
```

NOTE:- By configuring the <bag> tag, we can avoid an additional column in the child table (list-index), used for storing index.

Q:- What is the difference b/w using a collection of type java.util.Set and java.util.List ?

Ans:- Incase of java.util.Set, it is not possible to read a perticuler child object of the parent, by without loading all child objects from the DataBase.

→(i) Incase of java.util.List, it is possible to read a perticuler child object for the parent by without loading all child objects

(ii) because, List allows index based accessing of an object.

(iii) The difference can be found, when Lazy loading is applied, but not in early loading.

For eg:- by using the following code, we can read only 4th child of the parent from the DataBase, by withloading all child objects from the DataBase.

*) Lazy = "true" ⟶ lazy loading

```
Dealer d = (Dealer) session.get(Dealer.class, 101);
List list = d.getProducts();
Object o = list.get(3);
Product p = (Product) o;
```

Using Collection type java.util.Map:-

→ While creating one-to-many relationship for storing a group of child objects in a parent Object, we use a collection type of property in a parent class. That collection type can be either a java.util.Set (or) java.util.List (or) java.util.Map.

→ If we use a Collection type as java.util.Map then in Mapping file of Hibernate, we should use <Map> tag

→ If Collection type is used as a Map then, in child table of DataBase an additional column is required for an Hibernate, for storing the key assigned for a child Object, while storing it in a Map.

→ If Collection type is java.util.Map then, while configuring the <Map> tag, we need to configure an additional tag also inside <map> tag called <map-key>

→
java.util.Set	⟶	<set>
java.util.List	⟶	<list> / <bag> / <idbag>
java.util.Map	⟶	<map>

Eg:- (*) We have two POJO classes called Dealer and Product & the Relationship is a one-to-many from a dealer to a product.

(*) In dealer POJO class, we are using "Map" type of collection for storing a group of products to a dealer.

(*) At the time of inserting a Dealer with Products in DataBase then hibernate also stores the key of each product used in a "Map" collection in the DataBase

One-to-many 3

- Dealer.java
- Product.java
- Dealer.hbm.xml
- product.hbm.xml
- hibernate.cfg.xml
- InsertClient.java
- *.class

// Dealer.java

```
import java.util.*;
public class Dealer
{
  private int dealerId;
  private String dealerName;
  private Map products;
    setters & getters
}
```

// Product.java

```
public class Product
{
  private int productId;
  private String productName;
  private double price;
    setters & getters
}
```

// dealer.hbm.xml

```
<!DOCTYPE ----- >
<h-m>
  <class name="Dealer" table="dealer">
    <id name="dealerId" column="did" />
    <property name="dealerName" column="dlname" length="10" />
    (***)<map name="products" cascade="all" >
        <key column="dealid" />
      (***)<map-key column="mkey" type="java.lang.String" length="10" />
        <one-to-many class="Product" />
    </map>
  </class>
</h-m>
```

//product.hbm.xml

//hibernate.cfg.xml

```
<mapping resource="product.hbm.xml" />
<mapping resource=
```

```
// InsertClient.java
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
  P S V m (String[] args)
  {
   SessionFactory factory=
   Session session=
  // Parent Object
 Dealer d= new Dealer();
  d.setDealerId (101);
  d.setDealerName ("ABC");
  // child object-1
Product P1=new Product();
 P1.setProductId (901);
 P1.setProductName ("X");
 P1.setPrice (2000);
// child object-2
Product P2=new Product();
 P2.setProductId (902);
 P2.setProductName ("Y");
 P2.setPrice (3000);
// child object-3
Product P3=new Product();
 P3.setProductId (903);
 P3.setProductName ("Z");
 P3.setPrice (4000);
//step-1 //add child objects to java.util.Map
  Map map=new HashMap();
   map.put ("prod1", P1);
   map.put ("prod2", P2);
   map.put ("prod3", P3);
  //step-2 // add java.util.Map to Parent Object
   d.setProducts(map);
Transaction tx=session.beginTransaction();
  Session.save(d);
   tx.commit();
  session.close();
  factory.close();
  }
}
```

Select * from dealer;

Did	Dname
101	ABC

Select * from product;

Pid	pname	price	Dealid	Mkey
901	X	2000	101	prod1
902	Y	3000	101	prod2
903	Z	4000	101	prod3

according to the hashcode Map data structure will be stored data in reverse order.

Q:- What is the diff b/w using java.util.List and java.util.Map?

Ans:- Incase of List, we can avoid the additional column in child table by Configuring <Bag> tag. But Incase of Map, it is not possible to avoid additional column in child table.

→ The Similarity b/w using a collection type as List and a collection type as Map is, it is possible to obtain a perticuler child object of the parent object, instead of loading all child objects of the parent object

→ Incase of collection type List, we use index and incase of collection type Map we use <map-key>

The following is to read a perticuler child of a parent Object:-

```
Dealer d = (Dealer) session.get(Dealer.class, 111);
Map map = d.getProducers();
Object o = map.get("prod2");
Product p = (Product) o;
S.o.p (p.getXxx());
```

Many-to-one

→ Many-to-one relationship can be applied from a child object to a parent object.

→ According to Many-to-one, a parent object is added to Many child objects.

→ To apply Many-to-one assossiation from a child object to its parent object, we need the following changes.

① in child pojo class, we need a reference, pointing to its parent class.

② in child mapping file, we need to configure <many-to-one> tag, to inform the hibernate about the relationship.

→ In hibernate, when working with this relationships, we need a collection in a parent pojo class for to-many relation and we need a reference of parent class in a child class when the relation is to-one.

Example:- We have two pojo classes Vendor and Customer & the relation of many-to-one is a Vendor object is added to many Customer objects.

→ The DataBase table of this relationship are

PK Vendor (Parent)

Vid	Vname

PK Customer (Child)

Custid	custname	Custaddr	FK Venid

```
//Parent
//Vendor.java
Public class Vendor
{
 private int VendorId;
 private String VendorName;
      ≡
}
```

```
//Child
//Customer.java
 public class Customer
 {
    private int customerId;
    private String customerName;
    private String customerAddress;
(****) private Vendor vendor;
      ≡
 }
```

```
//Vendor.hbm.xml
<h-m>
  <class name="Vendor" table="vendor">
    <id ---/>
    <property --/>
</h-m> </class>
```

//Customer.hbm.xml

```
<h-m>
  <class name="Customer" table="customer">
      <id --/>
      <property --/>
      <property --/>
      <many-to-one name="vendor" class="Vendor" column="venid" cascade="all"/>
  </class>
</h-m>
```

name = reference of Vendor; class = parent class name; column = Foreign key

→ in a client application of Hibernate a Vendor object is added to the Customer Objects by calling setVendor( ) method of Customer POJO class.

Eg:-
```
c1.setVendor(v);
c2.setVendor(v);
c3.setVendor(v);
```

→ At the time of saving a Customer object, hibernate 1st verifies weather its Parent Vendor Object already exist in the database (or) not. If exist then only child object will be saved in the DataBase.

If parent Object doesn't exist in the DataBase then hibernate 1st will save Parent Object and then after the child Object.

many-to-one

- vendor.java
- Customer.java
- Vendor.hbm.xml
- Customer.hbm.xml
- hibernate.cfg.xml
- InsertClient.java
- *.class

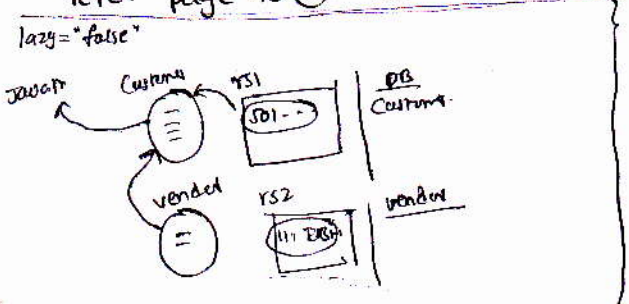
O/P:- select * from vendor;

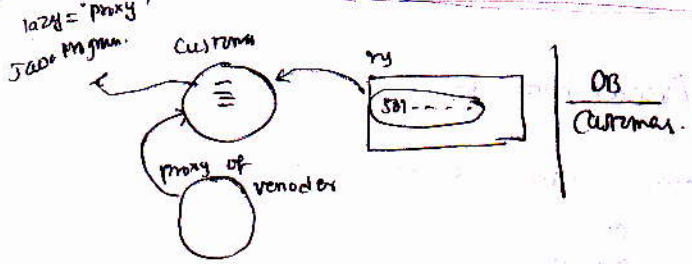
Vid	Vname
111	IBM

select * from customer;

Custid	custname	custadd	VENID
501	Infy	HYD	111
502	TCS	HYD	111
503	Verizon	US	111

refer page no (3) an application (15) of handout (2)





"lazy" attribute of many-to-one :-

→ in Many-to-one relationship, whenever a child object is loaded from the DataBase then its assossiated parent object is also loaded automatically.

→ in this Many-to-one, hibernate decides whether a parent object of a child to be loaded either immidietly or later, by depending on "Lazy" attribute.

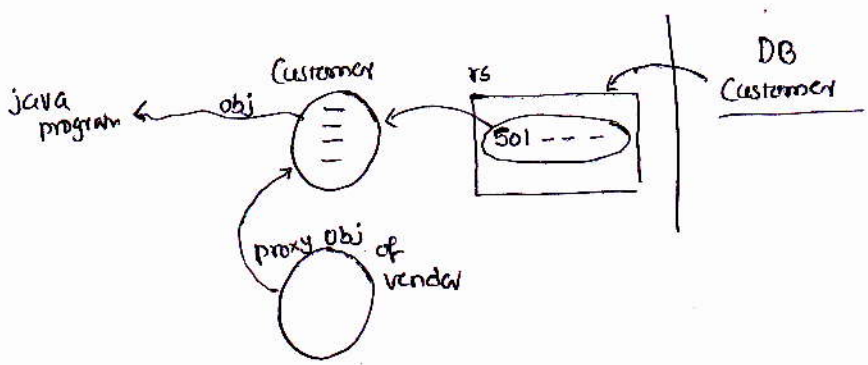
→ The default value of lazy attribute in many-to-one is "proxy". This is called proxy loading.

→ If you write "lazy=false" then it is called "early loading". In proxy loading, hibernate creates a proxy object of parent and then it is added to child object and after that the child object is given to our java program.

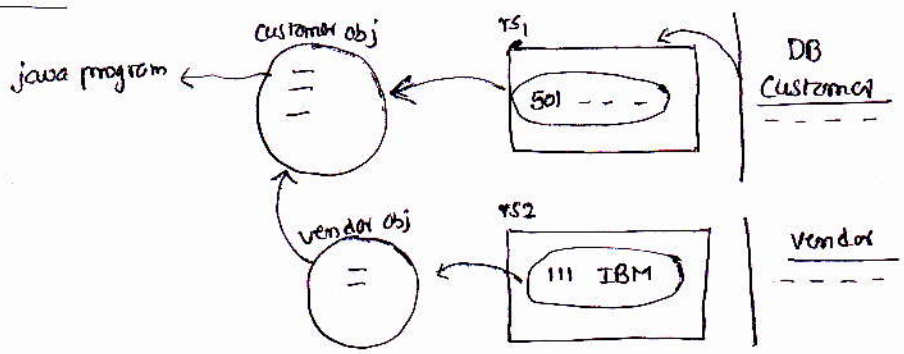
→ In case of early lodding, hibernate creates a real parent object and then it is added to the child object and finally the child object is given to java program.

Eg:- Customer c = (Customer) session.get((Customer.class, 501);

lazy="proxy" :-



lazy="false" diagram :-



Delete operation in many-to-one :-

→ When we are trying to delete a child object then hibernate tries to delete its parent object also from the Database.

→ If the parent object has any another child then Hibernate is going to throw an exception.

→ If the parent object of the child object does not have no more childs then hibernate deletes the child object and also its parent object from the Database.

→ If you want to delete only the child object by without effecting it on its parent object then we need to modify the cascade attribute to save (or) save-update, but not delete (or) all.

Eg:

```
Customer c=(Customer) session.get(Customer.class, 501);
    Transaction tx=session.beginTransaction();
        Session.delete(c);
        tx.commit();
```

One-to-many (Bi-directional) & many-to-one (bi-directional)

→ one-to-many (bi-directional). is a combination of one-to-many and many-to-one.

→ For one-to-many, we need a collection in parent class. and for many-to-one, we need a reference of parent is required in child class.

for Eg:- If we want to apply one-to-many in bi-directional blw vendor and Customer POJO classes then we need the following changes.

① Create a collection property in parent class.

② Create a reference of parent in child class

③ add <set> collection type of tag in parent mapping file and add <many-to-one> tag in child mapping file.

```
// Parent
public class Vendor
{
   private int vendorId;
   private String vendorName;
   private Set customers;
   .....
   ....
}
```

```
// child
public class Customer
{
   private int customerId;
   private String customerName;
   private String customerAddress;
   private Vendor vendor;
   .....
   ....
}
```

Vendor.hbm.xml

```
<h-m>
  <class name="Vendor" table="vendor" >
    <id ... .../>
    <property ... .../>
  <set name="customers" cascade="all" inverse="true">
    <key column="venid"/>
    <one-to-many class="Customer"/>
  </set>
  </class>
</h-m>
```

→ In the above mapping file, for <set> tag, we added an attribute "inverse=true"

This inverse attribute is used to inform the hibernate that the relationship is bi-directional.

→ The default value of inverse attribute is "false".

→ If we donot add "inverse=true" then hibernate uses more database operations to apply bi-directional relation ship.

Customer.hbm.xml

```
<h-m>
  <class name="Customer" table="customer">
    <id - - - />
    <property ... .../>
    <property ... .../>
    <many-to-one name="vender" class="Vendor" column="venid" cascade="all" />
  </class>
</h-m>
```

for one-to-many (bi-directional) page no: ⑥ an applicatton ⑯ of handout ②.

many-to-many relationship :-

→ many-to-many relationship is a one-to-many relationship from both sides.

→ To apply many-to-many relationship blw objects of two pojo classes using hibernate then we need a collection type of property at both POJO classes.

→ many-to-many relationship is always a bi-directional. we no need to inform the hibernate that it is bi~~-~~directional.

Eg:-

We have two POJO classes called Student and Course. The many-to-many relationship between the two POJO classes is, one Student joined in multiple Courses, i.e. one-to-many. In reverse, one Course joined by multiple students, so again it is one-to-many. Finally the relationship is many-to-many.

→ In Student POJO class, we need a Collection type of property for storing a group of Courses joined by a Student. Similierly, in Course POJO class, we need a Collection type of property for storing a group of students joined in a Course.

→ While applying many-to-many relationship blw two POJO classes objects, at database side, we need a third table also, called as Join table. The minimum no of tables required to apply many-to-many relationship is 3

→ A Join table contains foreign keys of many-to-many relationship

Example :- The POJO classes and the Database tables of this Student and Course relationship are

```
public class Student
{
    → StudentId
    → StudentName
    → StudentGrp
    private Set courses;
    ==
}
```

```
public class Course
{
    → CourseId
    → CourseName
    → duration
    private Set Students;
    ==
}
```

PK → Student

Sid | Sname | Grp

PK → Course

Cid | Cname | duration

Students_Courses (Join table)

Sid_fk | Cid_fk

→ Student.hbm.xml

```
<h-m>
  <class name="Student" table="student">
    <id ... />
    <property ... />
    <property ... />
*** <set name="courses" cascade="all" table="Students_Courses">   (Students_Courses → join table)
      <key column="Sid_fk" />
      <many-to-many class="course" column="Cid_fk" />   (Cid_fk → opposite foreign key)
    </set>
  </class>
</h-m>
```

→ course.hbm.xml

```
<h-m>
  <class name="Course" table="course">
    <id ... />
    <property ... />
    <property ... />
  <set name="Students" cascade="all" table="Students_course">
    <key column="Cid_fk" />
    <many-to-many class="Student" column="Sid_fk" />
  </set>
</class>
</h-m>
```

→ In a client application of Hibernate, the following is the procedure for adding a group of courses to a student.

```
Ser  ser1 = new  HashSet();
    ser1.add(c1);
    ser1.add(c2);
    s1.setCourses(ser1);

Ser  ser2 = new  HashSet();
    ser2.add(c1);
    ser2.add(c2);
    ser2.add(c3);
s2.setCourses(ser2);
```

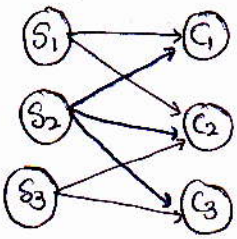
→ In the above code, two courses are added for student1 and three courses are added for student2. So, the relationship is one-to-many. At the same time, same course is added for student1 and student2. So, again the relationship is one-to-many.

finally the relationship added is many-to-many.

In a client appn of Hibernate, the following is the procedure for adding a group of students to a one course.

```
set  set1 = new  HashSet();
    set1.add(s1);
    set1.add(s2);
    c1.setStudents(set1);

set  set2 = new  HashSet();
    set2.add(s1);
    set2.add(s2);
    set2.add(s3);
c2.setStudents(set2);
```



many-to-many is nothing but, one-to-many relationship again reverse of one-to-many relationship.

S_1 student joined in C_1 & C_2 courses → one to many
C_1 course is joined by S_1 & S_2 student → one-to-many
} many-to-many

Many-To-many

→ Student.java
Course.java
Student.hbm.xml
Course.hbm.xml
hibernate.cfg.xml
InsertClient.java
*.class

refer page (8) application no: (17)

Output :-

SQL> Select * from Student ;

SID	SNAME	GRP
1001	aaa	Btech
1002	bbb	MCA

SQL> selet * from course ;

CID	CNAME	DURATION
902	Oracle	100
901	java	150

SQL> Selet * from Students_Courses ;

SID_FK	CID_FK
1001	902
1001	901
1002	902
1002	901

(*) Adding an additional course to a Student :-

→ To add a new course to the existing Student then the following "4" steps are required.

Step-I :- load a Student Object from the DataBase.

Step-II :- read the Set of Courses of the Student

Step-III :- Create a new Course Object with data

Step-IV :- add the Course Object to the Set with in a Transaction.

//Step-1 :-

```
Student s = (Student) session.get(Student.class, 1001);
```

//Step-2 :-

```
Set set = s.getCourses();
```

//Step-3 :-

```
Course c = new Course();
c.setCourseId(903);
c.setCourseName(cpp);
c.setDuration(150);
```

//Step-4 :-

```
Transaction tx = session.beginTransaction();
set.add(c);
tx.commit();
```

SQL> select * from Course;

CID	CNAME	Duration
902	oracle	100
901	java	150
903	cpp	150

SQL> select * from students_coourses;

SID_FK	CID_FK
1001	901
1002	901
1001	902
1002	902
1001	903

To add a new Student to the existing Course then the following 4 steps are required.

Step-I :- load a Course object from the DataBase

Step-II :- read the set of students of the course

Step-III :- create a new student object with data

Step-IV :- add the student object with the set with in a Transaction.

Step-I :-

```
Course c = (Course) session.get(course.class, 901);
```

Step-II :-

```
Set set = c.getStudents();
```

Step-III :-

```
Student s = new Student();
s.setStudentId(1003);
s.setStudentName("xyz");
s.setStudentCity("B.Tech");
```

Step-IV :-

```
Transaction tx = session.beginTransaction();
set.add(s);
tx.commit();
```

Deleting a Course from a Student :-

→ To delete and existing course from a Student the following steps are required.

① load the student from DataBase

② read all the courses joined by a collection object

③ load the course to the removed from a student

④ remove that course from the collection, with in a Transaction.

```
//Step-1
Student s=(Student) session.get(Student.class, 1001);
//Step-2
Set set = s.getCourses();
//Step-3
Course c=(Course) session.get(Course.class, 902);
// Step-4
Transaction tx = session.beginTransaction();
    set.remove(c);
    tx.commit();
```

*) One-to-one :-

one-to-one relationship can be applied in two ways.

① one-to-one with Foreign key

② one-to-one with Primary key

One-to-one with Foreign key:-

→ one-to-one with Foreign key is almost equal to applying many-to-one, but the difference is Foreign key column of child table should not allow duplicate values and also null values.

→ While Configuring this one-to-one relationship with Foreign key, we use <many-to-one> tag in the mapping file, but we add "Unique" & "not-null" attributes for the <many-to-one> tag.

→ While applying this one-to-one with Fk relationship, we need a reference in child class pointing to its parent class.

Example:-

We have two POJO classes called Person and License. The relationship is one person has one License.

→ In hibernate, while applying this one-to-one relationship, the relationship must be configured from child class to parent class.

→ The POJO classes of this one-to-one relationship are

```
public class Person
{
  private int personId;
  private String personName;
    setters & getters
}
```

```
public class License
{
  private int licenseId;
  private Date issuedDate;
  private Date expiredDate;
(**) private Person person;
    setters & getters
}
```

person	
pid (PK)	pname

license			
lid (PK)	idate	edate	perid (FK)

// person.hbm.xml

```
<h-m>
    <class  name="Person"  table="person" >
        <id  - - />
        <property  - - />
        <property  - - />
    </class>
</h-m>
```

// license.hbm.xml

```
<h-m>
    <class  name="License"  table="license" >
        <id  - - />
        <property  - - />
        <property  - - />
(***) <many-to-one  name="person"  class="Person"  column="perid" />
</class>
</h-m>
```

name="person" → reference name

class="Person" → parent class

column="perid" → FK

(***) unique="true"

(***) not-null="true"

cascade="all" />

One to One Fk

- Person.java
- License.java
- person.hbm.xml
- license.hbm.xml
- hibernate.cfg.xml
- InsertClient.java
- *.class

refer page (11) on appn (18) of handout (2).

O/P:

```
Select * from person;
```

pid	pname
1101	ABC

```
Select * from license;
```

Lid	idate	Edate	per_id
1010	28-mar-13	28-mar-13	1101

→ In this no additional columns are added bcz foreign key column does not allow duplicate values.

→ In this delete operation is performed when the parent object is removed from the table automatically child obj will be removed from the table.

one-to-one with primary key :-

→ In one-to-one with Foreign key, in child table a separate foreign key column is taken (required) and it does not allow duplicate & null values

→ In child table, there is a primary key column also, which does not allow duplicate and null values.

→ Instead of taking primary key and Foreign key separately, we can make primary key column also as a Foreign key in the child table. This type of relationship is called one-to-one with primary key.

→ In this one-to-one relationship, primary key value of parent object will be copied into primary key of child object.

→ In this one-to-one with primary key, we use a generator class called "foreign"

→ with the help of "foreign" generator, we can inform the hibernate that copy the primary key value of a parent object into primary key value of the child object.

```
public class person
{
  private int personId;
  private int personName;
    setters & getters
}
```

```
// person.hbm.xml
<h-m>
  <class name="Person" table="person">
    <id -- />
    <property -- />
  </class>
</h-m>
```

```
public class License
{
  private int licenseId;
  private Date issuedDate;
  private Date expiredDate;
  private Person person;
    setters & getters
}
```

```
// License.hbm.xml
<h-m>
  <class name="License" table="license">
    <id name="licenseId" column="lid">
      (****) <generator class="foreign">
        <param name="property">person</param>
      </generator>
    </id>
    <property -- />
    <property -- />
    (****) <one-to-one name="person" class="Person" cascade="all" />
  </class>
</h-m>
```

PK → Person (parent) | PK&FK → License (child)

Pid pname | Lid idate edate

One to One PK

→ person.java
License.java
License.hbm.xml
person.hbm.xml
hibernate.cfg.xml
InsertClient.java
*.class

refer page (13) an appn (14) of handout (2)

program is not inserting into jointable. hibernate inserts into jointable.

*** <idbag> of many-to-many :-

→ While applying many-to-many relationship, hibernate uses Join table for storing foreign keys. In the join table the foreign keys values are duplicated.

→ In a join table, there is no uniqueness (or) uniquecolumn, to access (or) modify the data easily. so, for adding uniqueness to the join table, we use <idbag> tag.

→ To configure <idbag> tag in a mapping file, the collection type in the pojo class must be "List" (java.util.List)

→ in <idbag> tag, we use another tag called <collection-id> for mapping the primary key column of join table.

The following are the changes required in previous many-to-many example we use <idbag> tag.

① in Student.java, replace collection type Set with collection type List.

```
import java.util.*;
Public class Student
{
    private int studentId;
    private String studentName;
    private String studentAddr;
} private List courses;
```

② in Student.hbm.xml, replace <Set> tag with <idbag> tag like the following.

```
<idbag name="courses" table="Students_Courses" cascade="all">
    <collection-id column="S_C_id" type="int">
        <generator class="increment" />
    </collection-id>
    <key column="sid_fk" />
    <many-to-many class="Course" column="cid_fk" />
</idbag>
```

NOTE:- In the above <collection-id> tag, generator class should not be "assigned".

③ in InsertClient.java, in place of Set object we need list object.

```
List list = new ArrayList();
list.add(C1);
list.add(C2);
S1.setCourses(list);
S2.setCourses(list);
```

④ After executing the InsertClient, we obtain the following output in DataBase.

Select * from Student;

Sid	Sname	GRP
1001	aaa	BTech
1002	bbb	MCA

Select * from course;

Cid	Cname	Duration
901	java	160
902	oracle	100

Select * from Students_Courses;

Sid_FK	Cid_FK	S_C_ID (PK)
1001	901	1
1001	902	2
1002	901	3
1002	902	4

***

Q:- What is 1+n (or) n+1 problem in hibernate? (n → no of parent objects)

Ans:- In relationships, when loading parent objects with their respective child objects from the DataBase then hibernate uses 1 select operation for loading all the parent objects and hibernate uses a separate select operation for loading child objects of each parent object. It means, if there are 5 parent objects then hibernate uses "1" select operation for loading all the 5 parent objects and 5 select operations for loading the childs of each parent object. Totally 1+5 Select operations are generated by hibernate. This is called "1+n" problem in Hibernate.

NOTE:- In 1+n problem "n" represents no. of parent objects.

Joins :-

→ Join statements are used to select the data of multiple POJO class objects, by using the relationship added b/w them, using a Single Select operation.

→ Join Statements are only applicable, when a relationship exist between the POJO class objects.

→ Hibernate Supports the following 4 types of Joins.

1. Inner Join
2. Left Outer Join
3. Right Outer Join
4. Full Join

→ The following HQL query is to Select a Vendor name and its assosciated Customer name using One-to-many relationship added b/w a Vendor and a Customer objects.

V.customers as c

Select v.vendorName, c.customerName from Vendor v join v.customers c

(v.customers → collection property)

→ The following HQL Command is for Select a Vendor name and its assosciated Customer name, by using many-to-one relationship b/w

Customer and Vendor.

Select v.vendorName, c.customerName from Customer c join c.vendor V

→ If we don't specify a Join type then by default the join type is "Inner Join".

→ Inner Join is an equi Join of SQL. It means, the Join stmt selects only the data from table, if an assosiated data exist for it.

(¥) The following client appn uses one-to-many relationship blw a vendor and a Customer pojo classes. and obtains the data from two tables using Join stmt

```
// JoinTest.java
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class JoinTest
{
  p s v m (String[] args)
  {
SessionFactory factory = new Configuration().configure().buildSessionFactory();
  Session session = factory.openSession();
Query qry = session.createQuery("select v.vendorName, c.customerName from Vendor
                                 join v.customers c");
    List l=qry.list();
  Iterator it= l.iterator();
  While(it.hasNext())
  {
    Object row[] = (Object[])it.next();
      S.o.p(row[0]+" "+row[1]);
  }
  Session.close();
  factory.close();
  }
}
```

→ Incase of LeftOuter join, Hibernate selects equal data from bothsides of the & join and also more data from LeftSide of the Join, even though there is

no assosciated data at Right side of the Join.

for Eg:-, The following Join Statement selects more data from Vendor, even though there is no assosciated data at Right side (i.e. at customer)

Select v.vendorName, c.customerName from Vendor v left outer join v.customers c

→ In case of right outer join, hibernate selects more data from right side of the join, even though there is no assosciated data at Left Side of the Join.

→ In case of Full Join, hibernate selects both equal data and also unequal data from both sides of the Join statement.

Solution for 1+n problem :- in HQL

→ To reduce 1+n select operations to a single select operation, for loading the parent objects and their assosciated child objects, we need to use "Join fetch" statement in HQL.

→ If we construct the following HQL query then hibernate selects all vendors and their respective customers with a single select operation

Query qry = session.createQuery ("from Vendor v join fetch v.customers");

↳ it is automatically disable the lazy loading

→ If we execute the above query then hibernate automatically disables Lazy loading (nothing but early Loading is applied.) and generates one select operation for loading all parents and childs

reducing the no. of select operations, the performance of an application will be increase.

Solution for 1+n problem in Criteria :-

→ The following is for reducing 1+n problem through Criteria API of hibernate.

```
Criteria crit = session.createCriteria(Vendor.class);
crit.setFetchMode("customers", FetchMode.JOIN);
List l = crit.list();
```

In the above code, if we donot set Fetch mode then by default the fetch mode will be taken as "SELECT". SELECT mode means "1+n" select operations are generate to reduce "1+n" select operations into a single select operation, we need to change the fetch mode as "JOIN".

fetching Strategies of Hibernate :-

1. select
2. Join
3. subselect

① → by default fetching strategy is "select". select means, one select operation is generated for reading the parents and a seperate select operation is generated for reading childs of each parent. here, 1+n problem occures.

② → "Join" strategy generates "1" select operation for all parents and their respective collection of childs, to load from the DataBase. It means "1+n" operations are reduced to "1" select operation.

③ → "subselect" fetching strategy, hibernate generates "1" select operation for all parent objects and "1" select operation for all childs of all parents. Totally "1+1" select operations are generated.

*) If we call SetFetchMode (-,-) method of Criteria then it is possible to set either "Select (or) Join" fetching strategy. But subselect fetching strategy can't be set.

*) Instead of calling SetFetchMode (-,-) method, we can configure the fetching strategy in a mapping file, by adding "fetch" attribute for the relationship configuration of mapping file.

```
<set name="customers" cascade="all" fetch="select|join|subselect">
    .....
    .....
</set>
```

NetBeans IDE :-

Steps to develop an Hibernate appn using NetBeans IDE.

Step-1:- Create a table in the DataBase like the following.

SQL> create table demo (sid number(5) primary key, sname varchar2(10));

Step-2:- Start NetBeans IDE

Step-3:- Click on File Menu → New Project → (Category: Java) → JavaApplication → next → Project Name: [Hibernate1] → ☐ create Main class → Finish

Step-4:- Right click on Libraries folder → Add Library → Hibernate → Add Library

Step-5:- Right click on project name (Hibernate1) → new → Other → (Left side: Hibernate) → Hibernate Configuration Wizard → next → next → Database Connection: → New Database Connection → enter the following details →

Driver Name : [▽] (New Driver) → [ojdbc14.jar] [add] → OK

Host : [localhost]

port : [1521]

Service ID : [Orcl]

Username : [scott]

Password : [tiger] → OK → finish

Step-6:- Right click on project name → new → Other → (Left side: Hibernate) → Hibernate Reverse Engineering Wizard → next → next → select Demo table at left side → add → finish

Step-7:- Right click on project name → new → other → (Left side: Hibernate) → select Hibernate Mapping Files and POJOs from Database → package : [P1] → finish

Step-8:- Right Click on project name → new → Java Class → class Name : [InsertClient] → finish

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

```
public class InsertClient {
    P s v m (String args[])
    {
SessionFactory factory = new Configuration().configure().BuildSessionFactory();
Session session = factory.openSession();
P1.Demo d = new P1.Demo();
  d.setSid (111);
  d.setSname ("abc");
  Transaction tx = session.beginTransaction();
      Session.save(d);
      tx.commit();
    Session.close();
    factory.close();
  }
}
```

Step-9:- Right click on InsertClient.java ⟶ RunFile

Annotation :-

① // This is used in String format ⟶ metadata
String S1="10" ⟶ data

② final class Demo (→ metadata)
{
int a=10;
String s="100"; } data
}

③ hibernate.cfg.xml
<property name="org.hibernate.connection" (→ metadata) > Oracle.jdbc.OracleDriver </property>
data

④ hbm.xml
<class ... > } → it is containing totally metadata
<id ... />
<property ... />

To reduce the mapping file based on xml file use the Annotation. bcz it contains only metadata. Only metadata content will be write in Annotation format. Data/metadata content does not write in Annotation format. So, hibernate mapping file entirely write in Annotation format, but not Configuration file write in Annotation format.

Annotations in Hibernate

→ While writing Java apps, there are different ways to provide metadata.

① We can write Comments in a Java program, define some kind of metadata for Java statements in a program.

② In Java, we have access modifiers, which also defines metadata for the elements of a Java application.

③ We can create Xml file to define metada.

→ In the above 3ways, Xml way of defining metadata is increasing the burden on Java programers, Because as part of a Java project development, a Java application need more no of Xml files.

→ In order to reduce the heavy use of Xml's in a Java Project development, we got annotations, introduced into java from Jdk1.5

→ While Constructing Hibernate applications, we are providing ORM metadata in the form of mapping xml files.

→ In order to reduce the mapping Xml files from an Hibernate apps, we got annotations in Hibernate.

→ By using annotations, we can't replace a Configuration file of Hibernate. because a Configuration file is going to contain data also.

→ Annotations in Hibernate for persistance are the implementations of the annotations provided by SUN as part of Java persistence API (JPA).

→ JPA is a specification released by SUN as part of EJB 3.x and implementations of this specification are provided by the Server vendor supporting EJB 3.x and ORM tool vendors.

→ The annotations in all ORM tools for persistence are same in all ORM tools, because the all ORM tools are provided implementation for common JPA Specification.

→ While creating an annotation, some metadata is provided for an annotation. It means metadata of metadata.

The following are the two informations added for an annotation while creating it.

① To what type of Java element an annotation is applicable. That is for a class (or) for a method (or) for a property ... etc.

② To what level this annotation is visible. i.e. weather an annotation is visible for compile time level (or) Runtime level.

→ The 1st information for an annotation is set by using @Target. This is called Target Specification (or) Target Information

→ The 2nd information is added using @Retention. This is called setting the Retention policy.

Common Annotations With POJO class :-

→ While creating a pojo class of hibernate, we need to import a package of JPA called Javax.persistence

→ The following 4 annotations are commonly using while creating a pojo class

① @Entity
② @Table
③ @Id
④ @Column

@Entity and @Table are applicable at class level and @Id & @Column are applicable at either property level or getter method level.

*** → For each pojo class, we should add @Entity annotation. By adding this annotation, a pojo class acts like an Entity bean

→ @Table is used to map a pojo class to a DataBase table. If pojo class name and table name are matched then this @Table annotation is optional.

@Id is mapped to a property with a primary key column of a DataBase table.

→ @Column is to map a property with a Column of a DataBase table.

→ If property name and column name are matched then @Column annotation is optional.

→ In configuration file of hibernate, instead of mapping resource, we should add mapping "class".

→ In case of before hibernate 3.6, in a client appn we need AnnotationConfiguration class object. but from hibernate 3.6 AnnotationConfiguration class is merged into Configuration class only. so, we don't required any changes in a client application of hibernate.

NOTE:-

(i) Incase of annotations, annotations are added to POJO class. so, If any changes are made to Annotation then we need to recompile POJO class. but this is not a draw back. The reason is, in an application metadata is not changed frequently. only data changes frequently.

(ii) If we use annotations and also we create mapping xml file for metadata then finally mapping xml metadata is considered.

① The following appn is for inserting a Student object in a DataBase using Annotations.

Application 1

→ Student.java
hibernate.cfg.xml
InsertClient.java
*.class

refer page (16) an appn (2) of handout (2)

Q:- Difference blw Save(-) & persist(-) method ?

Ans:- both Save(-) & persist(-) methods are used to Save an Object in a DataBase. but Save(-) method returns the id of the Saved object back to a Java program, but persist(-) method doesn't return the id of Saved Object.

→ return type of Save(-) method is Serializable, but the return type of persist(-) method is void

→ Save(-) method is useful, to see the generatorId for an Object, when generator class is used as other than "assigned"

→ While calling Save(-) method, we can store the returned value into a Serialized Object

```
Integer i = (Integer) session.save(s);
  └ Serialized object
```

→ If we use generator is assigned, then persist(-) method is usefull. It doesn't return any Object. bcz its return type is void.

Annotations for inheritance mapping :-

→ for inheritance mapping, we need a class level annotation called @Inheritance

→ For @Inheritance, we need to pass a parameter called "strategy". This strategy param value indicates the inheritance type.

@Inheritance(strategy = InheritanceType.SINGLE_TABLE) → Table per class

@Inheritance(strategy = InheritanceType.JOINED) → Table per sub class

@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS) → Table per concreate class

Table per class :-

→ When Table per class hirarchy is applied then we need a Discriminator Column in the DB table

→ To map a DiscriminatorColumn, we use @DiscriminatorColumn and to assign the value, we use @DiscriminatorValue.

→ Incase of Table per subclass (or) Table per Concreate class, DiscriminatorColumn is optional.

→

TablePerClass
- Payment.java
- CreditCardPayment.java
- ChequePayment.java
- hibernate.cfg.xml
- InsertClient.java
- *.class

refer page no: 17 an application (21) of an Handoout (2).

Table Per Subclass :-

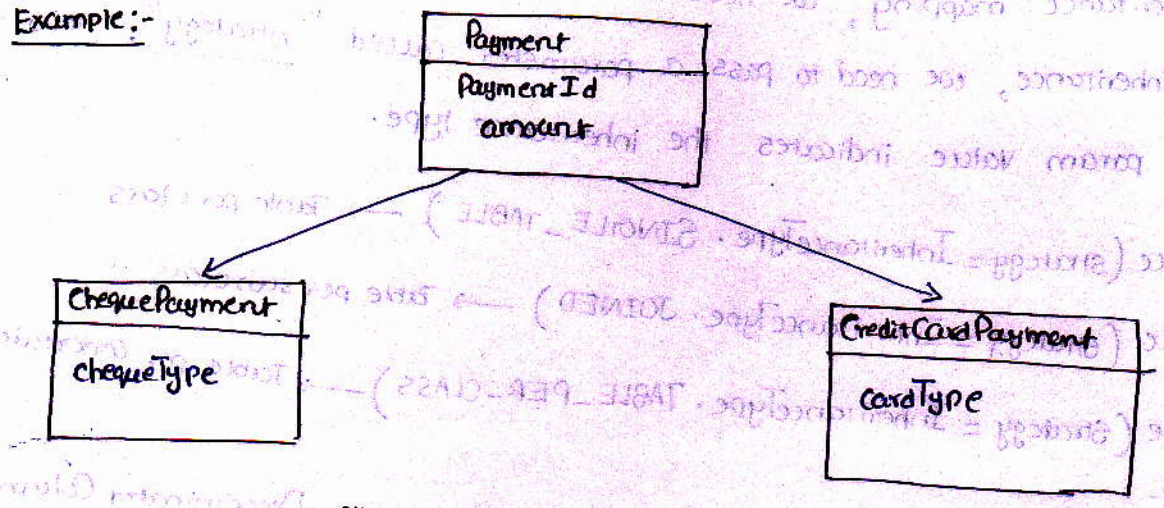
→ In case of Table Per Subclass hirarchy, in the DataBase separate tables are required for parent class and also for each sub class.

→ At the time of inserting the objects, Hibernate copies Parent properties into Parent table and Child properties into child table and copies primary key of parent table into child table.

→ In child table, the primary key of the child table acts like a Foreign key also.

→ To inform the hibernate that a column is acting like a primary key and also a Foreign key, we need an annotation called "@PrimaryKeyJoinColumn".

Example:-



pk
↓
payid — Payment — amount

pk&fk
↓
Pid — Credit_table — cctype

pk&fk
↓
Pid — cheque_table — chtype

TablePerSubclass

→ Payment.java
CreditCardPayment.java
ChequePayment.java
InsertClient.java
hibernate.cfg.xml

refer page (18) appn (23) of handout (2)

Annotations With Relationships

One-to-many with Annotations :-

→ The following two Annotations are required for applying one-to-many (Unique) relationship

@OneToMany (targetEntity = class obj of child pojo class, cascade = CascadeType.ALL, fetch = FetchType.EAGER/LAZY)

@JoinColumn (name = "fk", referencedColumnName = "pk")

OneToManyAnnotation
→ Vendor.java
Customer.java
hibernate.cfg.xml
InsertClient.java
*.class

refer page no (20) on appn (24) of handout (2)

many-to-one with Annotations :-

→ The following two Annotations are required for applying many-to-one (Unique) relationship.

→ we need a property in child class, which is pointing to the its parent class.

→ To the parent class reference property, we need to apply the following two Annotations.

@ManyToOne (targetEntity = class obj of parent pojo class, cascade = CascadeType.ALL, fetch = FetchType.EAGER)

@JoinColumn (name = "fk", referencedColumnName = "pk")

For Example, To apply many-to-one relationship from Customer to Vendor, we need to create the Vendor and Customer POJO classes like the following.

```
//Vendor.java
import javax.persistence.*;
@Entity
@Table (name="vendor") ⇒ optional
public class Vendor
{
  @Id
  @Column (name="vid");
  private int VendorId;
  @Column (name="vname");
  private String vendorName;
  → setters & getters
}
```

```
// Customer.java
import javax.persistence.*;
@Entity
@Table (name="customer")
p c Customer
{
  @Id
  @Column (name="custid")
  private int customerId;
  @Column (name="custname")
  private String customerName;
  @Column (name="custAdd")
  private String customerAddress;
```

in xml format we have proxy loading & eary loading.
But in Annotation we don't have proxy Loading.
we have LAZY (or) EAGER Loadings

```
@ManyToOne(targetEntity = Vendor.class, cascade = CascadeType.ALL);
@JoinColumn(name = "venid", referencedColumnName = "vid");
    private Vendor vendor;
        =
}
```

ManyToOne Annotation

→ Vendor.java
Customer.java
hibernate.cfg.xml
InsertClient.java
*.class

One-to-many (bi-directional) with Annotation :-

→ By adding both one-to-many and many-to-one annotation, we can develope bi-directional one-to-many assossiation.

→ while creating bi-directional relationship, instead of repeating @JoinColumn Annotation for both sides, we can avoid this @JoinColumn at many-to-one by adding "mappedBy" parameter for @ManyToOne(-,-,-,-) Annotation.

→ To apply one-to-many in bi-directional blw Vendor and Customer we Create Vendor and customer classes like the following.

```
public class Vendor
{
  @Id
  @Column(name="vid");
  private int VendorId;
  @Column(name="vname");
  private String vendorName;
@OneToMany(targetEntity = Customer.class, cascade = CascadeType.ALL)
@JoinColumn(name="venid", referencedColumnName="vid")
    private Set customers;
        =
}
```

```
public class Customer
{
    → customerId
    → customerName
    → customerAddress
@ManyToOne (targetEntity = Vendor.class, cascade = CascadeType.ALL,
                                          mappedBy = "customers")
    private Vendor vendor;
        ≡
}
```

↳ To Avoid @JoinColumn in child table.

Many-To-Many With Annotations :-

→ Many-To-Many relationship is always a bi-directional

→ when many-to-many relationship is applied then we need a JoinTable also in the DataBase.

→ A minimum of 3 tables are required to apply the many-to-many relationship.

→ The following are the 3 Annotations required for applying Many-to-Many Assosiation.

1) @ManyToMany
2) @JoinTable
3) @JoinColumn

→ Incase of ManyToMany, we need JoinTable and the foriengn keys of the relationship are stored in JoinTable

→ @JoinColumn annotation is used inside @JoinTable annotation.

Eg:- We have two POJO classes called Catagory & Item. The relationship blw these two class objects is Many-to-many.

i.e. one Catagory has Many items and one item belongs to Many Catagories.

→ The DataBase tables of this relationship are

PK
Categories
cat_id | cat_name

PK
items
item_id | item_name

categories_items
cat_id_FK | item_id_FK

```
// Category.java
import javax.persistence.*;
import java.util.*;
@Entity
@Table(name="categories")
public class Category
{
  @Id
  @Column(name="cat_id")
  private int categoryId;

  @Column(name="cat_name")
  private String categoryName;
@ManyToMany(targetEntity = Item.class, cascade = CascadeType.ALL)
@JoinTable(name="categories_items", joinColumns = @JoinColumn(name="cat_id_fk", referencedColumnName="catid"),
    inverseJoinColumns = @JoinColumn(name="item_id_fk", referencedColumnName="itemid"))
private Set items;
  → Setters & getters
}

// Item.java
import javax.persistence.*;
import java.util.*;
@Entity
@Table(name="items")
public class Item
{
  @Id
  @Column(name="item_id")
  private int itemId;
  @Column(name="item_name")
  private String itemName;
@ManyToMany(targetEntity = Category.class, cascade = CascadeType.ALL, mappedBy="items")
  private Set categories;
}
```

JoinTable (this table

no need to write @JoinTable once again

Property of category (Collection)

ManyToManyAnnotation

→ Category.java
Item.java
hibernate.cfg.xml
InsertClient.java

refer page no: (24) an appn (26)

(*) One-To-One Using Annotations :-

→ We can apply one-to-one relationship in two ways.

① one-to-one with Foreignkey

② one-to-one with primarykey

→ To apply one-to-one with Foreignkey we should not allow duplicate values and null values into Foreignkey column.

→ while applying one-to-one using Foreign key with annotation, we should add Unique and not null attributes for @JoinColumn annotation

→ For one-to-one with Foreignkey, we need @ManyToOne annotation only.

→ If you want to apply one-to-one relationship b/w person and License classes using annotations, in one-to-one with Foreignkey approach then the following are the classes with annotations.

```
// Person.java
import javax.persistence.*;
@Entity
@Table(name="person")
public class Person
{
  @Id
  @Column(name="pid")
  private int personId;
  @Column(name="pname")
  private String personName;
  ---
  ---
  --
}
```

```
//License.java
import javax.persistence.*;
@Entity
@Table(name="license");
public class License
{
  @Id
  @Column(name="lid")
  private int licenseId;
  @Column(name="idate")
  private Date issuedDate;
  @Column(name="edate")
  private Date expireDate;
@ManyToOne(targetEntity=Person.class, cascade=CascadeType.ALL)
@JoinColumn(name="per_id", referencedColumnName="pid",
            unique="true", nullable="false")
  private Person person;
  ---
  ---
  --
}
```

nullable="false" → null values are not allowed

adding a generator for an id using annotations :-

→ If we want to add any hibernate provided generator class to an Id then first we need to define that generator strategy using @GenericGenerator

→ after defining the strategy, to add a generator for an Id we need @GeneratedValue annotation.

→ @GenericGenerator is an own annotation of hibernate and it is not from JPA.

→ Hibernate has provided two annotations of its own @GenericGenerator & @Parameter.

→ These two annotations are from org.hibernate.Annotations pkg.

→ If you donot specify any generator for an Id by default assigned generator is used.

→ If we want to add "Increment" generator for an Id using annotations then we should create POJO class like the following

```
// Product.java
import javax.persistence.*;
import org.hibernate.annotation.*;
@Entity
@Table(name="product")
public class Product
{
@GenericGenerator(name="gen1", strategy="increment")
                  └alias name
@Id
@GeneratedValue(generator="gen1")
@Column(name="pid")
private int productId;
....
...
}
```

→ TO apply "Sequence" generator for an Id, we need the following annotations

```
@GenericGenerator(name="gen1", strategy="sequence", parameters=@Parameter(name="sequence",
                                                                value="mysequence"))
@Id
@GeneratedValue(generator="gen1")
@Column(name="pid")
private int productId;
```

→ To apply "hilo" generator for an Id, we need the following generator:

```
@GenericGenerator(name="gen1", strategy="hilo",
        parameters={@Parameter(name="table", value="mytable"),
                    @Parameter(name="column", value="next"),
                    @Parameter(name="max_lo", value="10")})
@Id
@GeneratedValue(generator="gen1")
@Column(name="pid")
private int productId;
...
```

OneToOne with primary key :-

→ To apply oneToOne with primary key, we need to make primary key of child table also as a foreign key.

→ To inform the hibernate that, copy the primary key of one object into primary key of another object, we use "foreign" generator.

→ for example, To apply oneToOne with primary key relationship b/w a person and a license class, we need to create the POJO classes with annotations like the following.

```
//Person.java
import javax.persistence.*;
@Entity
@Table(name="person")
public class Person
{
  @Id
  @Column(name="per_id")
  private int personId;
  @Column(name="per_name")
  private String personName;
  ...
}
```

```
//License.java
import java.util.*;          → Date class
import javax.persistence.*;  → JPA annotations
import org.hibernate.annotations.*;
@Entity
@Table(name="license")
public class License
{
@GenericGenerator(name="gen1", strategy="foreign",   ← aliasname
        parameters=@Parameter(name="property",
                              value="person"))
@Id
@GeneratedValue(generator="gen1")
@Column(name="lic_id")
private int licenseId;
@Column(name="idate")
private Date issueDate;
@Column(name="edate")
private Date expireDate;
```

@OneToOne(targetEntity = Person.class, cascade = CascadeType.ALL)

@PrimaryKeyJoinColumn → it informs the hibernate that a primary key is also acting like as a foreign key

private Person person;

}

OneToOne Annotation

- Person.java
- License.java
- hibernate.cfg.xml
- InsertClient.java
- *.class

refer page (26) appn (27) of handcrat (2)

~~SQL> Select * from person;~~

~~PER_ID~~

~~Plot~~

Component mapping in hibernate :-

→ Component mapping is a mapping of making a reference of one POJO class as a member variable of another POJO class.

→ If you apply Component mapping then Objects of two POJO classes are inserted to a Single table of DataBase.

→ In order to inform the hibernate that one class reference is acting as a Component of another class, In a hibernate mapping file we use <Component> tag

→ If you want to apply Component mapping using annotations then we use the following two annotations.

@Embeddable

@Embedded

→ While using annotations, to make a POJO class as (Embeddable) insertable into another class Object, we use an annotation @Embeddable

→ In a POJO class, by making another class reference as a Component, we use @Embedded

→ In the following example, a reference of PName class is acting as a Component of Person class.

```
//PName.java
import javax.persistence.*;
@Embeddable
public class PName
{
  @Column(name="I")
  private char initial;
  @Column(name="fname")
  private String firstName;
  @Column(name="lname")
  private String lastName;
  ...
}
```

```
//Person.java
import javax.persistence.*;
@Entity
@Table(name="persons")
public class Person
{
  @Id
  @Column(name="pid")
  private int personId;
  @Column(name="dob")
  private java.util.Date dob;
  @Embedded
  private PName pname;
  ...
}
```

ComponentTest
- Person.java
- PName.java
- hibernate.cfg.xml
- InsertClient.java
- *.class

refer page (28) appn (28) of handout (2)

```
SQL> select * from persons;
```

PID	DOB	FNAME	I	LNAME
111	06-APR-13	Abc	S	XYZ

Q:- What are different types of ORmappings (objectRelational mappings) ?

Ans:-
1. Full Object mapping
2. Partial Object mapping
3. Inheritance mapping
4. Component mapping
5. Assosiation / Relational mapping.

Q:- what is @Temporal annotation

Ans:- @Temporal annotation is used to inform the hibernate about weather a Date (or) Time (or) Date and Time are to be stored in the DataBase.

→ by default hibernate only inserts Date. If you want to inform Time (or) Time & Date is also inserted into DB then we apply this @Temporal annotation.

For eg:-
```
@Column(name="dob")
@Temporal(TemporalType.TIME)
private Date dob;
```

→ hibernate inserts only Time into DataBase from dob property.

for eg:-

```
@Column(name="dob")
@Temporal(TemporalType.TIMESTAMP)
private Date dob;
```

In the above example, hibernate inserts both date and time into a DataBase table.

Struts with Hibernate Integration :-

→ Integrating Struts with Hibernate is nothing but calling persistence logic implemented in Hibernate from Action class of Struts.

→ In Struts f/w, for integration is possible with other frameworks from Action class of Struts.

→ The following are the options available to integrate Struts with Hibernate.

① we can define all the persistence logic of hibernate in execute(-,-,-,-) of Action class. In this approach, no.of hibernate Factory objects are increased. because execute(-,-,-,-) of Action class is called for each request.

→ The drawback of this approach is, a burden on a server is increased, because of no.of SessionFactory objects.

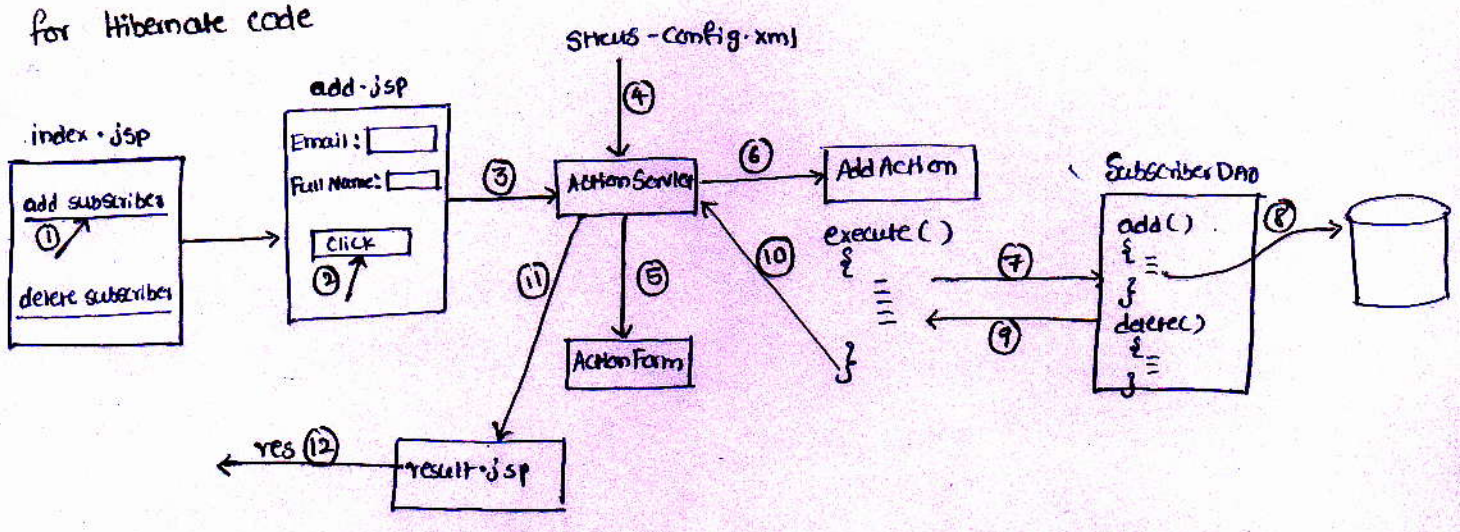
② we can create SessionFactory in Constructor of an Action class and session in execute(-,-,-,-) of an Action class. Here, factory is created for once, because in struts, ~~only~~ one object of Action class will be shared by multiple threads.

→ The drawback of this approach is, Factory created in on Actionclass can't be shared with another Action class. It means, for example if we have 10 Action classes then we need to create 10Factories, one for each class.

③ we can create a Separate java class for creating SessionFactory for once and for defining methods to do database operations by hibernate. This Separate class is called a DAO class.

→ This approach is a better approach, for integrating Struts and hibernate. because One hibernate-factory can be shared by multiple Actionclasses of Struts

(*) The following is a Struts with Hibernate Integration example, where we are performing add and delete operations on a Subscriber of data, by creating DAO class for Hibernate code



Subscribers

```
Application-1(Servlet with Hibernate: performing  insert,update,delete and select
                                                    operations(CURD operations))
-----------------index.html-----------------------------------
<center>
<h2>
<a   href="insert.html"> Create  Employee </a> <br>
<a   href="update.html">Update Employee </a> <br>
<a   href="delete.html"> Delete  Employee  </a>  <br>
<a   href="select.html">  Select  Employee  </a> <br>
</h2>
</center>
-----------------insert.html-----------------------------------
<center>
<h2>
   <form  action="insertsrv">
        Empno : <input  type=text   name="empno"><br>
        Ename  : <input  type=text   name="ename"><br>
        Sal         : <input  type=text   name="sal"><br>
        Deptno : <input  type=text   name="deptno"><br>
         <input  type=submit  value="INSERT">
   </form>
</h2>
</center>
----------------update.html-----------------------------
<center>
<h2>
   <form  action="updatesrv">
        Empno : <input  type=text   name="empno"><br>
        Ename  : <input  type=text   name="ename"><br>
        Sal         : <input  type=text   name="sal"><br>
        Deptno : <input  type=text   name="deptno"><br>
         <input  type=submit  value="UPDATE">
   </form>
</h2>
</center>
---------------select.html-----------------------------
<center>
<h2>
  <form  action="selectsrv">
      Empno : <input  type=text   name="empno"><br>
      <input  type=submit  value="SELECT">
  </form>
</h2>
</center>
--------------delete.html-----------------------------
<center>
<h2>
  <form  action="deletesrv">
      Empno : <input  type=text   name="empno"><br>
      <input  type=submit  value="DELETE">
  </form>
</h2>
</center>
-----------------web.xml--------------------------
<web-app>
  <servlet>
    <servlet-name>s1</servlet-name>
    <servlet-class>InsertServlet</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>s1</servlet-name>
     <url-pattern>/insertsrv</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>s2</servlet-name>
    <servlet-class>UpdateServlet</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>s2</servlet-name>
     <url-pattern>/updatesrv</url-pattern>
  </servlet-mapping>
```

```
  <servlet>
    <servlet-name>s3</servlet-name>
    <servlet-class>DeleteServlet</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>s3</servlet-name>
     <url-pattern>/deletesrv</url-pattern>
  </servlet-mapping>
  <servlet>
    <servlet-name>s4</servlet-name>
    <servlet-class>SelectServlet</servlet-class>
  </servlet>
  <servlet-mapping>
     <servlet-name>s4</servlet-name>
     <url-pattern>/selectsrv</url-pattern>
  </servlet-mapping>
</web-app>
----------------------Employee.java----------------------------
//Employee.java (POJO)
public class Employee
{
        private int employeeId;
        private String employeeName;
        private int employeeSal;
        private int deptNumber;
        public int getDeptNumber() {
                return deptNumber;
        }
        public void setDeptNumber(int deptNumber) {
                this.deptNumber = deptNumber;
        }
        public String getEmployeeName() {
                return employeeName;
        }
        public void setEmployeeName(String employeeName) {
                this.employeeName = employeeName;
        }
        public int getEmployeeId() {
                return employeeId;
        }
        public void setEmployeeId(int employeeId) {
                this.employeeId = employeeId;
        }
        public int getEmployeeSal() {
                return employeeSal;
        }
        public void setEmployeeSal(int employeeSal) {
                this.employeeSal = employeeSal;
        }
}
----------------------employee.hbm.xml-------------------------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <class name="Employee"  table="employee">
    <id  name="employeeId" column="empno"/>
    <property name="employeeName" column="ename"/>
    <property name="employeeSal"  column="sal"/>
    <property name="deptNumber"  column="deptno"/>
  </class>
</hibernate-mapping>
--------------------InsertServlet.java-----------------------
import  java.io.*;
import  javax.servlet.*;
import  org.hibernate.*;
import  org.hibernate.cfg.*;
public  class  InsertServlet   extends   GenericServlet
{
        private  SessionFactory  factory;
        //non-life cycle init method
        public  void  init()  throws  ServletException
```

```
143         {
144                 factory=new  Configuration().configure().buildSessionFactory();
145         }
146         //life cycle  service() method
147         public   void   service(ServletRequest req, ServletResponse  res)
148                             throws  ServletException,IOException
149         {
150                 //read  input  values
151                 String  s1 = req.getParameter("empno");
152                 String  s2 = req.getParameter("ename");
153                 String  s3 = req.getParameter("sal");
154                 String  s4 =    req.getParameter("deptno");
155                 //wrapping
156                 int  empno = Integer.parseInt(s1.trim());
157                 int  sal = Integer.parseInt(s3.trim());
158                 int  deptno = Integer.parseInt(s4.trim());
159                 //open  session
160                 Session   session = factory.openSession();
161                 //create  a  pojo class object
162                 Employee   e =  new  Employee();
163                 e.setEmployeeId(empno);
164                 e.setEmployeeName(s2);
165                 e.setEmployeeSal(sal);
166                 e.setDeptNumber(deptno);
167                 //begin  transaction
168                 Transaction  tx = session.beginTransaction();
169                 session.save(e);
170                 tx.commit();
171                  PrintWriter  pw = res.getWriter();
172                  pw.println("<h2> Employee  Saved Successfully</h2> <br>");
173                  pw.println("Return  <a href=index.html>HOME</a>");
174                  pw.close();
175                  session.close();
176         } //service
177         public  void  destroy()
178         {
179                 factory.close();
180         }
181 }
182 --------------------UpdateServlet.java-----------------------------
183 import  java.io.*;
184 import  javax.servlet.*;
185 import  org.hibernate.*;
186 import  org.hibernate.cfg.*;
187 public  class  UpdateServlet   extends   GenericServlet
188 {
189         private  SessionFactory  factory;
190         public  void  init()  throws  ServletException
191         {
192                 factory=new  Configuration().configure().buildSessionFactory();
193         }
194         public   void   service(ServletRequest req, ServletResponse  res)
195                             throws  ServletException,IOException
196         {
197                 //read  input  values
198                 String  s1 = req.getParameter("empno");
199                 String  s2 = req.getParameter("ename");
200                 String  s3 = req.getParameter("sal");
201                 String  s4 = req.getParameter("deptno");
202                 int  empno = Integer.parseInt(s1.trim());
203                 int  sal = Integer.parseInt(s3.trim());
204                 int  deptno = Integer.parseInt(s4.trim());
205                  PrintWriter  pw = res.getWriter();
206                 Session   session = factory.openSession();
207                 Object  o = session.get(Employee.class,empno);
208                 if(o==null)
209                 {
210                         pw.println("<h2> Employee Number   Not Found. So  row
211                         is not  updated </h2>");
212                         pw.println("Return  <a href=index.html>HOME</a>");
213                         return;
```

```
214                 }
215                 Employee   e = (Employee)o;
216                 Transaction  tx = session.beginTransaction();
217                 e.setEmployeeName(s2);
218                 e.setEmployeeSal(sal);
219                 e.setDeptNumber(deptno);
220                 tx.commit();
221                  pw.println("<h2> Employee  updated Successfully</h2> <br>");
222                  pw.println("Return  <a href=index.html>HOME</a>");
223                  pw.close();
224                  session.close();
225         } //service
226         public  void  destroy()
227         {
228                 factory.close();
229         }
230 }
231 -------------------------SelectServlet.java-------------------------
232 import  java.io.*;
233 import  javax.servlet.*;
234 import  org.hibernate.*;
235 import  org.hibernate.cfg.*;
236 public  class  SelectServlet   extends   GenericServlet
237 {
238         private  SessionFactory  factory;
239         public  void  init()  throws  ServletException
240         {
241                 factory=new  Configuration().configure().buildSessionFactory();
242         }
243         public   void   service(ServletRequest req, ServletResponse  res)
244                                         throws  ServletException,IOException
245         {
246                 String  s1 = req.getParameter("empno");
247                 int  empno = Integer.parseInt(s1.trim());
248                 PrintWriter  pw = res.getWriter();
249                 Session   session = factory.openSession();
250                 Object  o = session.get(Employee.class,empno);
251                 if(o==null)
252                 {
253                         pw.println("<h2> Employee Number   Not Found. So  row is
254                         not  selected </h2>");
255                         pw.println("Return  <a href=index.html>HOME</a>");
256                         return;
257                 }
258                 Employee  e =(Employee)o;
259                 pw.println("<h2>");
260                 pw.println(e.getEmployeeId());
261                 pw.println("<br>");
262                 pw.println(e.getEmployeeName());
263                 pw.println("<br>");
264                 pw.println(e.getEmployeeSal());
265                 pw.println("<br>");
266                 pw.println(e.getDeptNumber());
267                 pw.println("<br>");
268                 pw.println("</h2>");
269                 pw.println("Return  <a href=index.html>HOME</a>");
270                  pw.close();
271                  session.close();
272         } //service
273         public  void  destroy()
274         {
275                 factory.close();
276         }
277 }
278 ----------------------DeleteServlet.java----------------------------
279 import  java.io.*;
280 import  javax.servlet.*;
281 import  org.hibernate.*;
282 import  org.hibernate.cfg.*;
283 public  class  DeleteServlet   extends   GenericServlet
284 {
```

```
285         private  SessionFactory  factory;
286         public  void  init()  throws  ServletException
287         {
288                 factory=new  Configuration().configure().buildSessionFactory();
289         }
290         public   void   service(ServletRequest req, ServletResponse  res)
291                                         throws  ServletException,IOException
292         {
293                 //read  input  values
294                 String  s1 = req.getParameter("empno");
295                 int  empno = Integer.parseInt(s1.trim());
296                 PrintWriter  pw = res.getWriter();
297                 Session   session = factory.openSession();
298                 Object  o = session.get(Employee.class,empno);
299                 if(o==null)
300                 {
301                         pw.println("<h2> Employee Number   Not Found. So  row is
302                         not  deleted </h2>");
303                         pw.println("Return  <a href=index.html>HOME</a>");
304                         return;
305                 }
306                 Transaction  tx = session.beginTransaction();
307                 session.delete(o);
308                 tx.commit();
309                  pw.println("<h2> Employee  deleted Successfully</h2> <br>");
310                  pw.println("Return  <a href=index.html>HOME</a>");
311                  pw.close();
312                  session.close();
313         }//service
314         public  void  destroy()
315         {
316                 factory.close();
317         }
318 }
319 ================================================================================
320 application-2(connecting  with  multiple databases from an application)
321 ------------------Student.java-----------------------------
322 public class Student
323 {
324         private Integer studentid;
325         private String studentname;
326         private Integer marks;
327         public Integer getMarks() {
328                 return marks;
329         }
330         public void setMarks(Integer marks) {
331                 this.marks = marks;
332         }
333         public Integer getStudentid() {
334                 return studentid;
335         }
336         public void setStudentid(Integer studentid) {
337                 this.studentid = studentid;
338         }
339         public String getStudentname() {
340                 return studentname;
341         }
342         public void setStudentname(String studentname) {
343                 this.studentname = studentname;
344         }
345 }
346 -------------------student.hbm.xml-------------------------
347 <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
348     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
349 <hibernate-mapping>
350         <class name="Student" table="student">
351         <id name="studentid"  column="sid"  type="java.lang.Integer"/>
352         <property  name="studentname" column="sname" type="java.lang.String" length="10"/>
353         <property name="marks"/>
354     </class>
355 </hibernate-mapping>
```

```
--------------------hibernate_mysql.cfg.xml-----------------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
 <hibernate-configuration>
     <session-factory>
                <!-- connection properties -->
            <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
            <property name="connection.url">jdbc:mysql://localhost:3306/test</property>
            <property name="connection.username">root</property>
            <property name="connection.password"></property>
            <!-- hibernate properties -->
            <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
            <property name="show_sql">true</property>
            <property name="hbm2ddl.auto">update</property>
            <!-- mapping files -->
            <mapping resource="student.hbm.xml"/>
     </session-factory>
 </hibernate-configuration>
 --------------------hibernate_postgresql.cfg.xml--------------------
 <!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
 <!-- hibernate_postgresql.cfg.xml -->
 <hibernate-configuration>
     <session-factory>
          <!-- connection properties -->
            <property name="connection.driver_class">org.postgresql.Driver</property>
            <property name="connection.url">jdbc:postgresql://localhost:5432/postgres</prop
            <property name="connection.username">postgres</property>
            <property name="connection.password">postgres</property>
            <!-- hibernate properties -->
            <property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
            <property name="show_sql">true</property>
            <property name="hbm2ddl.auto">update</property>
            <!-- mapping files -->
            <mapping resource="student.hbm.xml"/>
     </session-factory>
</hibernate-configuration>
--------------------InsertClient.java----------------------
import org.hibernate.*;
import org.hibernate.cfg.*;
class InsertClient
{
        public static void main(String[] args)
        {
                SessionFactory factory1 = new Configuration().
                                configure("hibernate_mysql.cfg.xml").buildSessionFactory();
                Session session1 = factory1.openSession();
                SessionFactory factory2 = new Configuration().
                                configure("hibernate_postgresql.cfg.xml").buildSessionFactory()
                Session session2 = factory2.openSession();
                Student s = new Student();
                s.setStudentid(111);
                s.setStudentname("aaa");
                s.setMarks(200);
                Transaction tx1 = session1.beginTransaction();
                session1.persist(s);
                tx1.commit();
                Transaction tx2 = session2.beginTransaction();
                session2.persist(s);
                tx2.commit();
                session1.close();
                session2.close();
                factory1.close();
                factory2.close();
        }
}
================================================================================
        Application-3(Connection pooling with proxool)
------------Product.java-------------------------
```

```
public class Product
{
        private int productId;
        private String productName;
        private double price;
        public void setProductId(int productId)
        {
                this.productId = productId;
        }
        public int getProductId()
        {
                return productId;
        }
        public void setProductName(String productName)
        {
                this.productName = productName;
        }
        public String getProductName()
        {
                return productName;
        }
        public void setPrice(double price)
        {
                this.price=price;
        }
        public double getPrice()
        {
                return price;
        }
}
---------------product.hbm.xml-------------------------
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
   <class name="Product" table="product_info">
      <id name="productId" column="pid" type="int"/>
      <property name="productName" column="pname" length="10" type="string"/>
      <property name="price" type="double"/>
   </class>
</hibernate-mapping>
----------------hibernate.cfg.xml-------------------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
 <!-- hibernate.cfg.xml -->
 <hibernate-configuration>
     <session-factory>
          <!-- proxool properties -->
              <property name="hibernate.proxool.xml">proxool.xml</property>
              <property name="hibernate.proxool.pool_alias">abcd</property>
              <property name="hibernate.connection.provider_class">org.hibernate.
                                        connection.ProxoolConnectionProvider</property>
          <!-- hibernate properties -->
          <property name="dialect">org.hibernate.dialect.OracleDialect</property>
          <property name="show_sql">true</property>
          <property name="hbm2ddl.auto">update</property>

          <!-- mapping files -->
          <mapping resource="product.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
--------------------proxool.xml-------------------------
<!-- proxool.xml -->
<something>
   <proxool>
        <alias>abcd</alias>
        <driver-url>jdbc:oracle:thin:@localhost:1521:satya</driver-url>
        <driver-class>oracle.jdbc.OracleDriver</driver-class>
        <driver-properties>
            <property name="user" value="scott"/>
```

```
            <property  name="password"   value="tiger"/>
        </driver-properties>
        <max-connections-count>5</max-connections-count>
    </proxool>
</something>
--------------------ProductInsert.java-------------------
import  org.hibernate.*;
import  org.hibernate.cfg.*;
public  class  ProductInsert
{
        public static  void  main(String rags[ ])
        {
                Configuration  conf = new  Configuration();
                conf.configure("hibernate.cfg.xml");
                SessionFactory  factory = conf.buildSessionFactory();
                Session   session = factory.openSession();
                Product  p = new  Product();
                p.setProductId(101);
                p.setProductName("SONY");
                p.setPrice(9000);
                Transaction  tx = session.beginTransaction();
                session.save(p);
                tx.commit();
                session.close();
                System.out.println("Product saved in Database successfully");
                factory.close();
        }
}
================================================================
        Application-4(Server Connection pooling with  Jndi)
----------------Student.java-----------------------------
public class Student
{
        private Integer studentid;
        private String studentname;
        private Integer marks;
        public Integer getMarks() {
                return marks;
        }
        public void setMarks(Integer marks) {
                this.marks = marks;
        }
        public Integer getStudentid() {
                return studentid;
        }
        public void setStudentid(Integer studentid) {
                this.studentid = studentid;
        }
        public String getStudentname() {
                return studentname;
        }
        public void setStudentname(String studentname) {
                this.studentname = studentname;
        }
}
-------------student.hbm.xml-----------------------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
        <!-- student.hbm.xml -->
<hibernate-mapping>
    <class name="Student" table="student_info">
        <id name="studentid"  column="sid"  type="java.lang.Integer">
            <generator class="assigned"/>
       </id>
       <property  name="studentname" column="sname" type="java.lang.String" length="10"/>
       <property name="marks"/>
    </class>
</hibernate-mapping>
---------------hibernate.cfg.xml----------------------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

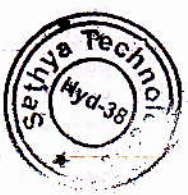
```
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
                 <!-- jndi  properties -->
                <property name="hibernate.connection.datasource">oraclejndi</property>
                <property name="hibernate.jndi.class">com.sun.enterprise.naming.
                                                SerialInitContextFactory</property>
                <property name="hibernate.jndi.url">iiop://localhost:4848</property>
                   <!--  hibernate properties -->
                <property name="show_sql">true</property>
                <property name="dialect">org.hibernate.dialect.OracleDialect</property>
                <property name="hbm2ddl.auto">update</property>
                  <!--  mapping  file  -->
                <mapping resource="student.hbm.xml"/>
        </session-factory>
 </hibernate-configuration>
-----------------Client.java------------------------------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class Client
{
        public static void main(String args[ ])
        {
                Configuration conf = new Configuration();
                conf.configure("hibernate.cfg.xml");
                SessionFactory factory=conf.buildSessionFactory();
                Session  ses = factory.openSession();
                Student  s1 = new Student();
                s1.setStudentid(102);
                s1.setStudentname("XYZ");
                s1.setMarks(600);
                Transaction tx = ses.beginTransaction();
                ses.save(s1);
                tx.commit();
                ses.close();
                factory.close();
        }
}
================================================================

        Application-5(Table per class  inheritance)
-----------------Payment.java----------------------------
import   java.util.*;
public   class  Payment
{
        private  int  paymentId;
        private  double  amount;
        private  Date   paymentDate;
        public double getAmount() {
                return amount;
        }
        public void setAmount(double amount) {
                this.amount = amount;
        }
        public int getPaymentId() {
                return paymentId;
        }
        public void setPaymentId(int paymentId) {
                this.paymentId = paymentId;
        }
        public  void   setPaymentDate(Date paymentDate)
        {
                this.paymentDate=paymentDate;
        }
        public  Date   getPaymentDate()
        {
                return  paymentDate;
        }
}
-----------------CreditCardPayment.java--------------------------
public class CreditCardPayment extends Payment
```

```
        <!--  payment2.hbm.xml -->
<hibernate-mapping>
   <class    name="Payment"    table="payments"  schema="scott">
     <id  name="paymentId"   column="payid">
           <generator   class="increment"/>
     </id>
     <property  name="amount"/>
     <property  name="paymentDate"   column="pdate"/>
     <joined-subclass   name="CreditCardPayment"  table="credit_table">
          <!--  foreign key mapping -->
          <key  column="pid"/>
          <property  name="cardType"   column="cctype" length="8"/>
     </joined-subclass>
     <joined-subclass  name="ChequePayment"  table="cheque_table">
          <key   column="pid"/>
          <property  name="chequeType"  column="chtype"  length="8"/>
     </joined-subclass>
  </class>
</hibernate-mapping>
=============================================================================
        Application-7(Table  per  concrete class)
----------------------payment3.hbm.xml-------------------------------
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
        <!--  payment3.hbm.xml -->
<hibernate-mapping>
   <class    name="Payment">
      <id   name="paymentId"   column="payid">
             <generator  class="increment"/>
      </id>
      <property   name="amount"/>
      <property   name="paymentDate"  column="pdate"/>
      <union-subclass   name="CreditCardPayment"  table="credit_table">
           <property   name="cardType"  column="cctype"  length="8"/>
      </union-subclass>
      <union-subclass  name="ChequePayment"  table="cheque_table">
            <property    name="chequeType"   column="chtype"  length="8"/>
      </union-subclass>
  </class>
</hibernate-mapping>
=============================================================================
        Application-8(HQL Select  operation)
----------------------Employee.java-------------------------------
//Employee.java (POJO)
public class Employee
{
        private int employeeId;
        private String employeeName;
        private int employeeSal;
        private int deptNumber;
        public int getDeptNumber() {
                return deptNumber;
        }
        public void setDeptNumber(int deptNumber) {
                this.deptNumber = deptNumber;
        }
        public String getEmployeeName() {
                return employeeName;
        }
        public void setEmployeeName(String employeeName) {
                this.employeeName = employeeName;
        }
        public int getEmployeeId() {
                return employeeId;
        }
        public void setEmployeeId(int employeeId) {
                this.employeeId = employeeId;
        }
        public int getEmployeeSal() {
                return employeeSal;
```

```
782         }
783         public void setEmployeeSal(int employeeSal) {
784                 this.employeeSal = employeeSal;
785         }
786 }
787 ------------------------employee.hbm.xml-------------------------------
788 <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
789     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
790 <!-- employee.hbm.xml -->
791 <hibernate-mapping>
792   <class name="Employee"  table="employee">
793     <id  name="employeeId" column="empno"/>
794     <property name="employeeName" column="ename"/>
795     <property name="employeeSal"  column="sal"/>
796     <property name="deptNumber"  column="deptno"/>
797   </class>
798 </hibernate-mapping>
799 ------------------------SelectClient.java---------------------------------
800 //SelectClient.java
801 import  java.util.*;
802 import  org.hibernate.*;
803 import  org.hibernate.cfg.*;
804 class  SelectClient
805 {
806         public static void main(String[ ] args)
807         {
808                 SessionFactory   factory = new  Configuration().configure().buildSessionFac
809                 Session  session = factory.openSession();
810                 //way-1
811                 Query  qry1=session.createQuery("from  Employee   e where e.employeeSal>:p1
812                 qry1.setParameter("p1",2000);
813                 List  l1 = qry1.list();
814                 Iterator  it1 = l1.iterator();
815                 while(it1.hasNext())
816                 {
817                         Employee  e =(Employee)it1.next();
818                         System.out.println(e.getEmployeeId()+"\t"+e.getEmployeeName()+"\t"
819                                         +e.getEmployeeSal()+"\t"+e.getDeptNumber());
820                 }
821                 System.out.println("==================");
822                 //way-2
823                 Query  qry2 = session.createQuery("select  e.employeeId,e.employeeName  fro
824                                         Employee e  where  e.deptNumber=:p1");
825                 qry2.setParameter("p1",30);
826                 List  l2 = qry2.list();
827                 Iterator  it2 = l2.iterator();
828                 while(it2.hasNext())
829                 {
830                         Object  r[ ] = (Object[ ])it2.next();
831                         System.out.println(r[0]+"\t"+r[1]);
832                 }
833                 System.out.println("==================");
834                 //way-3
835                 Query  qry3 = session.createQuery("select  e.employeeName  from  Employee
836                                         where e.employeeName  like  '%S%' ")
837                 List l3 = qry3.list();
838                 Iterator  it3 = l3.iterator();
839                 while(it3.hasNext())
840                 {
841                         String  s =(String)it3.next();
842                         System.out.println(s);
843                 }
844                 System.out.println("==============");
845                 session.close();
846                 factory.close();
847         }
848 }
849 ============================================================================
850         Application-9(HQL   Insert  operation)
851 ---------------------------Employee.java-------------------------------
852 public class Employee
```

```
{
        private int employeeId;
        private String employeeName;
        private int employeeSal;
        private int deptNumber;
        public int getDeptNumber() {
                return deptNumber;
        }
        public void setDeptNumber(int deptNumber) {
                this.deptNumber = deptNumber;
        }
        public String getEmployeeName() {
                return employeeName;
        }
        public void setEmployeeName(String employeeName) {
                this.employeeName = employeeName;
        }
        public int getEmployeeId() {
                return employeeId;
        }
        public void setEmployeeId(int employeeId) {
                this.employeeId = employeeId;
        }
        public int getEmployeeSal() {
                return employeeSal;
        }
        public void setEmployeeSal(int employeeSal) {
                this.employeeSal = employeeSal;
        }
}
----------------------employee.hbm.xml-----------------------------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- employee.hbm.xml -->
<hibernate-mapping>
  <class name="Employee"  table="employee">
    <id  name="employeeId" column="empno"/>
    <property name="employeeName" column="ename"/>
    <property name="employeeSal"  column="sal"/>
    <property name="deptNumber"  column="deptno"/>
  </class>
</hibernate-mapping>
----------------------hibernate.cfg.xml----------------------------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
 <!--  hibernate.cfg.xml -->
 <hibernate-configuration>
     <session-factory>
                <!--  connection  properties  -->
       <property  name="connection.driver_class">oracle.jdbc.OracleDriver</property>
           <property  name="connection.url">jdbc:oracle:thin:@localhost:1521:satya</propert
           <property  name="connection.username">scott</property>
           <property  name="connection.password">tiger</property>
           <!--  hibernate  properties  -->
           <property  name="dialect">org.hibernate.dialect.OracleDialect</property>
           <property  name="show_sql">true</property>
           <property  name="hbm2ddl.auto">update</property>
           <!--  mapping  files  -->
           <mapping  resource="employee.hbm.xml"/>
           <mapping  resource="testemployee.hbm.xml"/>
   </session-factory>
</hibernate-configuration>
----------------------TestEmployee.java----------------------------
public  class  TestEmployee
{
        private int empId;
        private String  empName;
        private int empSal;
        private int deptno;
        public  void  setEmpId(int empId)
```



```
924         {
925                 this.empId=empId;
926         }
927         public int  getEmpId()
928         {
929                 return  empId;
930         }
931         public  void  setEmpName(String empName)
932         {
933                 this.empName=empName;
934         }
935         public  String  getEmpName()
936         {
937                 return  empName;
938         }
939         public  void   setEmpSal(int  empSal)
940         {
941                 this.empSal=empSal;
942         }
943         public   int  getEmpSal()
944         {
945                 return  empSal;
946         }
947         public  void  setDeptno(int  deptno)
948         {
949                 this.deptno=deptno;
950         }
951         public  int  getDeptno()
952         {
953                 return  deptno;
954         }
955 };
956 ----------------------testemployee.hbm.xml-----------------------
957 <!--  testemployee.hbm.xml -->
958 <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
959     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
960         <!-- testemployee.hbm.xml -->
961 <hibernate-mapping>
962    <class  name="TestEmployee"  table="TESTEMPLOYEE">
963          <id  name="empId"   column="empid"/>
964          <property  name="empName"   column="ename"  length="12"/>
965          <property  name="empSal"   column="sal"/>
966          <property  name="deptno"/>
967  </class>
968 </hibernate-mapping>
969 ----------------------InsertClient.java-------------------------
970 import  org.hibernate.*;
971 import  org.hibernate.cfg.*;
972 public  class  InsertClient
973 {
974         public  static void main(String  s[ ])
975         {
976                 SessionFactory  factory = new  Configuration().configure().buildSessionFact
977                 Session  session = factory.openSession();
978                 Query  qry = session.createQuery("insert  into  TestEmployee(empId,empName,
979                 empSal,deptno) select  e.employeeId,e.employeeName,e.employeeSal,e.deptNumb
980                                                                 from  Employee  e");
981                 Transaction  tx = session.beginTransaction();
982                 int  k = qry.executeUpdate();
983                 tx.commit();
984                 System.out.println(k+" objects inserted");
985                 Query  qry1=session.createQuery("update Employee e  set  e.employeeSal=9000
986                                                         where  e.deptNumber=30");
987                 Transaction  tx1 = session.beginTransaction();
988                  int k1 = qry1.executeUpdate();
989                   tx1.commit();
990                   System.out.println(k1+" objects are updated");
991                 session.close();
992                 factory.close();
993         }
994 };
```

```
995  ================================================================================
996          Application-10(Criteria Test)
997  ------------------------Employee.java---------------------------
998  public class Employee
999  {
1000         private int employeeId;
1001         private String employeeName;
1002         private int employeeSal;
1003         private int deptNumber;
1004         public int getDeptNumber() {
1005                 return deptNumber;
1006         }
1007         public void setDeptNumber(int deptNumber) {
1008                 this.deptNumber = deptNumber;
1009         }
1010         public String getEmployeeName() {
1011                 return employeeName;
1012         }
1013         public void setEmployeeName(String employeeName) {
1014                 this.employeeName = employeeName;
1015         }
1016         public int getEmployeeId() {
1017                 return employeeId;
1018         }
1019         public void setEmployeeId(int employeeId) {
1020                 this.employeeId = employeeId;
1021         }
1022         public int getEmployeeSal() {
1023                 return employeeSal;
1024         }
1025         public void setEmployeeSal(int employeeSal) {
1026                 this.employeeSal = employeeSal;
1027         }
1028 }
1029 ----------------------employee.hbm.xml---------------------
1030 <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
1031     "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
1032 <!-- employee.hbm.xml -->
1033 <hibernate-mapping>
1034   <class name="Employee"  table="employee">
1035     <id  name="employeeId" column="empno"/>
1036     <property name="employeeName" column="ename"/>
1037     <property name="employeeSal"  column="sal"/>
1038     <property name="deptNumber"  column="deptno"/>
1039   </class>
1040 </hibernate-mapping>
1041 ------------------------CritClient.java--------------------------
1042 import  org.hibernate.*;
1043 import  org.hibernate.cfg.*;
1044 import  org.hibernate.criterion.*;
1045 import  java.util.*;
1046 class  CritClient
1047 {
1048         public static void main(String[] args)
1049         {
1050                 SessionFactory  factory = new  Configuration().configure().buildSessionFact
1051                 Session  session = factory.openSession();
1052         //loading  full  objects(way1)
1053                 Criteria  crit  =  session.createCriteria(Employee.class);
1054                 //condition is  empname should contain 'S' and order is ascending
1055                 Criterion  condition1 = Restrictions.ilike("employeeName","%s%");
1056                 crit.add(condition1);
1057                 crit.addOrder(Order.asc("employeeName"));
1058                 List  l = crit.list();
1059                 //for each  loop
1060                 for(Object  o : l)
1061                 {
1062                         Employee  e=(Employee)o;
1063                         System.out.println(e.getEmployeeId()+"    "+e.getEmployeeName()+
1064                                         "    "+e.getEmployeeSal()+"     "+e.getDeptNumber());
1065                 }
```

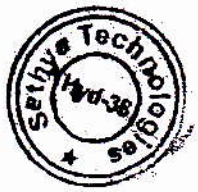
```
                System.out.println("============");
                //loading partial properties of objects(way2)
                Criteria  crit2 = session.createCriteria(Employee.class);
                Projection p1 = Projections.property("employeeName");
                Projection p2 = Projections.property("employeeSal");
                ProjectionList  plist = Projections.projectionList();
                plist.add(p1);
                plist.add(p2);
                crit2.setProjection(plist);
                List  l2 = crit2.list();
                Iterator  it2 = l2.iterator();
                while(it2.hasNext())
                {
                        Object  obj[ ] =(Object[ ])it2.next();
                        System.out.println(obj[0]+"          "+obj[1]);
                }
                System.out.println("======================");
                //loading objects with single property(way3)
                Criteria crit3 = session.createCriteria(Employee.class);
                 Projection  p3 = Projections.property("employeeSal");
                 crit3.setProjection(p3);
                 List  l3 = crit3.list();
                 Iterator  it3 = l3.iterator();
                 while(it3.hasNext())
                {
                        Integer  i =(Integer)it3.next();
                        System.out.println(i);
                }
                System.out.println("==================");
                session.close();
                factory.close();
        }
}
================================================================================
        Application-11(Native SQL  application)
----------------------Employee.java---------------------------------------
public class Employee
{
        private int employeeId;
        private String employeeName;
        private int employeeSal;
        private int deptNumber;
        public int getDeptNumber() {
                return deptNumber;
        }
        public void setDeptNumber(int deptNumber) {
                this.deptNumber = deptNumber;
        }
        public String getEmployeeName() {
                return employeeName;
        }
        public void setEmployeeName(String employeeName) {
                this.employeeName = employeeName;
        }
        public int getEmployeeId() {
                return employeeId;
        }
        public void setEmployeeId(int employeeId) {
                this.employeeId = employeeId;
        }
        public int getEmployeeSal() {
                return employeeSal;
        }
        public void setEmployeeSal(int employeeSal) {
                this.employeeSal = employeeSal;
        }
}
-----------------------employee.hbm.xml-----------------------------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
```

```
1137    <class name="Employee"  table="employee">
1138      <id  name="employeeId" column="empno"/>
1139      <property name="employeeName" column="ename"/>
1140      <property name="employeeSal"  column="sal"/>
1141      <property name="deptNumber"  column="deptno"/>
1142    </class>
1143  </hibernate-mapping>
1144  ------------------------NativeClient.java------------------------------
1145  import  org.hibernate.*;
1146  import  org.hibernate.cfg.*;
1147  import  java.util.*;
1148  class  NativeClient
1149  {
1150          public static void main(String[] args)
1151          {
1152                  SessionFactory   factory = new  Configuration().configure().buildSessionFac
1153                  Session  session = factory.openSession();
1154                  //non-select
1155                  SQLQuery  qry1 = session.createSQLQuery("insert  into  employee
1156                                                                  values(?,?,?,?)");
1157                  qry1.setParameter(0,7799);
1158                  qry1.setParameter(1,"ROM");
1159                  qry1.setParameter(2,5000);
1160                  qry1.setParameter(3,10);
1161                  Transaction  tx = session.beginTransaction();
1162                  int  k = qry1.executeUpdate();
1163                  tx.commit();
1164                  System.out.println(k+" row  inserted....");
1165                  System.out.println("================");
1166                  //select  operation
1167                  SQLQuery   qry2 = session.createSQLQuery("select  *  from  employee
1168                                                          where deptno=10");
1169                  qry2.addEntity(Employee.class); //Entity query object
1170                  List  l = qry2.list();
1171                  Iterator  it = l.iterator();
1172                  while(it.hasNext())
1173                  {
1174                          Employee  e = (Employee)it.next();
1175                          System.out.println(e.getEmployeeId()+"\t"+e.getEmployeeName()+"\t"+
1176                                          e.getEmployeeSal()+"\t"+e.getDeptNumber());
1177                  }
1178                  session.close();
1179                  factory.close();
1180          }
1181  }
1182  ================================================================================
1183          Application-12(calling  a  function using  Hibernate)
1184  ------------------------Employee.java-----------------------------------------
1185  public class Employee
1186  {
1187          private int employeeId;
1188          private String employeeName;
1189          private int employeeSal;
1190          private int deptNumber;
1191          public int getDeptNumber() {
1192                  return deptNumber;
1193          }
1194          public void setDeptNumber(int deptNumber) {
1195                  this.deptNumber = deptNumber;
1196          }
1197          public String getEmployeeName() {
1198                  return employeeName;
1199          }
1200          public void setEmployeeName(String employeeName) {
1201                  this.employeeName = employeeName;
1202          }
1203          public int getEmployeeId() {
1204                  return employeeId;
1205          }
1206          public void setEmployeeId(int employeeId) {
1207                  this.employeeId = employeeId;
```

```
1208          }
1209          public int getEmployeeSal() {
1210                  return employeeSal;
1211          }
1212          public void setEmployeeSal(int employeeSal) {
1213                  this.employeeSal = employeeSal;
1214          }
1215  }
1216  ----------------------------employee.hbm.xml--------------------------------------
1217  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
1218      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
1219  <hibernate-mapping>
1220    <class name="Employee"  table="employee">
1221      <id  name="employeeId" column="empno"/>
1222      <property name="employeeName" column="ename"/>
1223      <property name="employeeSal"  column="sal"/>
1224      <property name="deptNumber"  column="deptno"/>
1225    </class>
1226    <sql-query name="s1"  callable="true">
1227      <return  class="Employee"   alias="e"/>
1228            {?=call selectAllEmployees()}
1229     </sql-query>
1230  </hibernate-mapping>
1231  -------------------------client.java-----------------------------------------
1232  import org.hibernate.*;
1233  import org.hibernate.cfg.*;
1234  import  java.util.*;
1235  public class client
1236  {
1237          public static void main(String args[])
1238          {
1239                  SessionFactory factory =new Configuration().configure().buildSessionFactory
1240                  Session  ses = factory.openSession();
1241                  Query qry = ses.getNamedQuery("s1");
1242                  List l = qry.list();
1243                  Iterator it = l.iterator();
1244                  while(it.hasNext())
1245                  {
1246                          Employee  e =(Employee)it.next();
1247                          System.out.println(e.getEmployeeId()+"\t"+e.getEmployeeName()+"\t"+
1248                                          e.getEmployeeSal()+"\t"+e.getDeptNumber());
1249                          System.out.println("==============");
1250                  }
1251            ses.close();
1252            factory.close();
1253          }
1254  }
1255  ================================================================================
1256          Application-13(Pagination Application using Servlet with Hibernate)
1257  ----------------------Employee.java--------------------------
1258  public class Employee
1259  {
1260          private int employeeId;
1261          private String employeeName;
1262          private int employeeSal;
1263          private int deptNumber;
1264          public int getDeptNumber() {
1265                  return deptNumber;
1266          }
1267          public void setDeptNumber(int deptNumber) {
1268                  this.deptNumber = deptNumber;
1269          }
1270          public String getEmployeeName() {
1271                  return employeeName;
1272          }
1273          public void setEmployeeName(String employeeName) {
1274                  this.employeeName = employeeName;
1275          }
1276          public int getEmployeeId() {
1277                  return employeeId;
1278          }
```

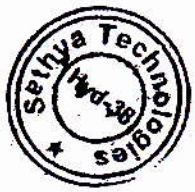
```
1279          public void setEmployeeId(int employeeId) {
1280                  this.employeeId = employeeId;
1281          }
1282          public int getEmployeeSal() {
1283                  return employeeSal;
1284          }
1285          public void setEmployeeSal(int employeeSal) {
1286                  this.employeeSal = employeeSal;
1287          }
1288  }
1289  ----------------------employee.hbm.xml-------------------------
1290  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
1291      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
1292  <hibernate-mapping>
1293    <class name="Employee"  table="employee">
1294      <id  name="employeeId" column="empno"/>
1295      <property name="employeeName" column="ename"/>
1296      <property name="employeeSal"  column="sal"/>
1297      <property name="deptNumber"  column="deptno"/>
1298    </class>
1299  </hibernate-mapping>
1300  ----------------------web.xml-----------------------------
1301  <web-app>
1302    <servlet>
1303        <servlet-name>paginationtest</servlet-name>
1304        <servlet-class>PaginationServlet</servlet-class>
1305    </servlet>
1306    <servlet-mapping>
1307          <servlet-name>paginationtest</servlet-name>
1308          <url-pattern>/ps</url-pattern>
1309    </servlet-mapping>
1310   </web-app>
1311  ----------------------PaginationServlet.java--------------------------------
1312  import java.io.*;
1313  import javax.servlet.*;
1314  import java.util.*;
1315  import org.hibernate.*;
1316  import org.hibernate.cfg.*;
1317  import org.hibernate.criterion.*;
1318  public class PaginationServlet extends GenericServlet
1319  {
1320          int pageIndex=0;
1321          SessionFactory factory;
1322          public void init() throws ServletException
1323          {
1324                  factory = new Configuration().configure().buildSessionFactory();
1325                  System.out.println("factory created.........");
1326          }
1327          public void service(ServletRequest req, ServletResponse res)
1328                          throws ServletException, IOException
1329          {
1330                  int totalNumberOfRecords=0;
1331                  int numberOfRecordsPerPage=3;
1332                  //logic-1
1333                  String sPageIndex = req.getParameter("pageindex");
1334                  if(sPageIndex==null)
1335                  {
1336                          pageIndex=1;
1337                  }
1338                  else
1339                  {
1340                          pageIndex=Integer.parseInt(sPageIndex);
1341                  }
1342                  Session session = factory.openSession();
1343                  //logic-2
1344                  int si=(pageIndex*numberOfRecordsPerPage) - numberOfRecordsPerPage;
1345                  Criteria crit2 = session.createCriteria(Employee.class);
1346                  crit2.setFirstResult(si);
1347                  crit2.setMaxResults(numberOfRecordsPerPage);
1348                  List l2 = crit2.list();
1349                  Iterator it2 = l2.iterator();
```

```
1350                    PrintWriter pw = res.getWriter();
1351                    pw.println("<center><table border=2 width='80%' height='70%'>");
1352                    pw.println("<tr>");
1353                    pw.println("<th>EMPNO</th><th>ENAME</th><th>SAL</th><th>DEPTNO</th>");
1354                    pw.println("</tr>");
1355                    while(it2.hasNext())
1356                    {
1357                            Employee e =(Employee)it2.next();
1358                            pw.println("<tr>");
1359                            pw.println("<td>"+e.getEmployeeId()+"</td>");
1360                            pw.println("<td>"+e.getEmployeeName()+"</td>");
1361                            pw.println("<td>"+e.getEmployeeSal()+"</td>");
1362                            pw.println("<td>"+e.getDeptNumber()+"</td>");
1363                            pw.println("</tr>");
1364                    }//while
1365                    pw.println("</table>");
1366                    //logic-3
1367                    Criteria crit1 =session.createCriteria(Employee.class);
1368                    crit1.setProjection(Projections.rowCount());
1369                    List  l1 = crit1.list();
1370                    Iterator it1 = l1.iterator();
1371                    if(it1.hasNext())
1372                    {
1373                            Object o = it1.next();
1374                            totalNumberOfRecords=Integer.parseInt(o.toString());
1375                    }
1376                    //logic-4
1377                    int noOfPages = totalNumberOfRecords/numberOfRecordsPerPage;
1378                    if(totalNumberOfRecords > (noOfPages * numberOfRecordsPerPage))
1379                    {
1380                            noOfPages = noOfPages+1;
1381                    }
1382                    pw.println("<br>");
1383                    //logic-5
1384                    pw.println("<table width=70% height=20%>");
1385                    pw.println("<tr>");
1386                    pw.println("<td>");
1387                    if(pageIndex>1)
1388                    {
1389
1390                            pw.println("<a  href=ps?pageindex="+(pageIndex-1)+">  Prev </a>");
1391                    }
1392                    else
1393                    {
1394                            pw.println(" ");
1395                    }
1396                    pw.println("</td>");
1397                    pw.println("<td>   </td>");
1398                    pw.println("<td> </td>");
1399                    pw.println("<td>");
1400                    if(pageIndex <noOfPages)
1401                    {
1402                            pw.println("<a  href=ps?pageindex="+(pageIndex+1)+"> Next </a>");
1403                    }
1404                    else
1405                    {
1406                            pw.println(" ");
1407                    }
1408                    pw.println("</table>");
1409                    pw.println("</center>");
1410                    session.close();
1411                    pw.close();
1412            }
1413            public void destroy()
1414            {
1415                    factory.close();
1416                    System.out.println("factory  closed");
1417            }
1418    }
```

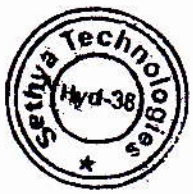



```
Application-14(One-To-Many Example)
----------Vendor.java----------
//Parent POJO
//Vendor.java
import java.util.*;
public class Vendor
{
    private int vendorId;
    private String vendorName;
    private Set customers;
    public Set getCustomers() {
        return customers;
    }
    public void setCustomers(Set customers) {
        this.customers = customers;
    }
    public int getVendorId() {
        return vendorId;
    }
    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }
    public String getVendorName() {
        return vendorName;
    }
    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
    }
}
----------Customer.java----------
//Child POJO
//Customer.java
public class Customer
{
    private int customerId;
    private String customerName;
    private String customerAddress;
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setCustomerAddress(String customerAddress)
    {
        this.customerAddress=customerAddress;
    }
    public String getCustomerAddress()
    {
        return customerAddress;
    }
}
----------vendor.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- vendor.hbm.xml -->
<hibernate-mapping>
  <class name="Vendor" table="vendor">
    <id name="vendorId" column="vid"/>
    <property name="vendorName" column="vname" length="10"/>
    <set name="customers" cascade="all" lazy="true">
      <key column="venid"/>
```

```
            <one-to-many class="Customer"/>
        </set>
    </class>
</hibernate-mapping>
----------customer.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- customer.hbm.xml -->
<hibernate-mapping>
  <class name="Customer" table="customer">
    <id name="customerId" column="custid"/>
    <property name="customerName" column="custname" length="10"/>
    <property name="customerAddress" column="custadd" length="10"/>
  </class>
</hibernate-mapping>
----------InsertClient.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
    public static void main(String args[])
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        //Parent object
        Vendor v = new Vendor();
        v.setVendorId(111);
        v.setVendorName("IBM");
        //Child object-1
        Customer c1 = new Customer();
        c1.setCustomerId(501);
        c1.setCustomerName("INFY");
        c1.setCustomerAddress("HYD");
        //Child object -2
        Customer c2 = new Customer();
        c2.setCustomerId(502);
        c2.setCustomerName("TCS");
        c2.setCustomerAddress("HYD");
        //Child object -3
        Customer c3 = new Customer();
        c3.setCustomerId(503);
        c3.setCustomerName("VERIZON");
        c3.setCustomerAddress("US");
        //step1
        Set s = new HashSet();
        s.add(c1);
        s.add(c2);
        s.add(c3);
        //step2
        v.setCustomers(s);
        Transaction tx = session.beginTransaction();
        session.save(v);
        tx.commit();
        session.close();
        factory.close();
    }
}
----------AdditionalChild.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class AdditionalChild
{
    public static void main(String args[])
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
```

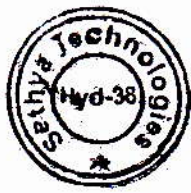



```
137             //step-1
138             Vendor v =(Vendor)session.get(Vendor.class,111);
139             //step-2
140             Set s = v.getCustomers();
141             //step-3
142             Customer c4 = new Customer();
143             c4.setCustomerId(504);
144             c4.setCustomerName("CTS");
145             c4.setCustomerAddress("US");
146             //step-4
147             Transaction tx = session.beginTransaction();
148             s.add(c4);
149             tx.commit();
150             session.close();
151             factory.close();
152         }
153     }
154     ----------SelectClient.java----------
155     import java.util.*;
156     import org.hibernate.*;
157     import org.hibernate.cfg.*;
158     public class SelectClient
159     {
160         public static void main(String args[])
161         {
162             SessionFactory factory = new Configuration().configure().buildSessionFactory();
163             Session session = factory.openSession();
164             Vendor v= (Vendor)session.get(Vendor.class,111);
165             System.out.println("Vendor name = "+v.getVendorName());
166             Set s = v.getCustomers();
167             Iterator it = s.iterator();
168             while(it.hasNext())
169             {
170                 Object o = it.next();
171                 Customer c =(Customer)o;
172                 System.out.println(c.getCustomerId()+"  "+c.getCustomerName()
173                         +"  "+c.getCustomerAddress());
174             }
175             session.close();
176             factory.close();
177         }
178     }
179     ----------DeleteClient.java----------
180     import org.hibernate.*;
181     import org.hibernate.cfg.*;
182     public class DeleteClient
183     {
184         public static void main(String args[])
185         {
186             SessionFactory factory = new Configuration().configure().buildSessionFactory();
187             Session session = factory.openSession();
188             Vendor v=(Vendor)session.get(Vendor.class,111);
189             Transaction tx = session.beginTransaction();
190             session.delete(v);
191             tx.commit();
192             session.close();
193             factory.close();
194         }
195     }
196     ==========
197         Application-15(Many-To-One Example)
198     ----------Vendor.java----------
199     //Parent POJO
200     //Vendor.java
201     public class Vendor
202     {
203         private int vendorId;
204         private String vendorName;
```

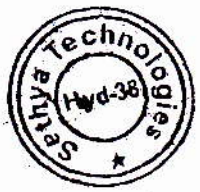



```
        public int getVendorId() {
            return vendorId;
        }
        public void setVendorId(int vendorId) {
            this.vendorId = vendorId;
        }
        public String getVendorName() {
            return vendorName;
        }
        public void setVendorName(String vendorName) {
            this.vendorName = vendorName;
        }
}
----------Customer.java----------
//Child POJO
//Customer.java
public class Customer
{
        private int customerId;
        private String customerName;
        private String customerAddress;
        private Vendor vendor;
        public void setVendor(Vendor vendor)
        {
            this.vendor=vendor;
        }
        public Vendor getVendor()
        {
            return vendor;
        }
        public int getCustomerId() {
            return customerId;
        }
        public void setCustomerId(int customerId) {
            this.customerId = customerId;
        }
        public String getCustomerName() {
            return customerName;
        }
        public void setCustomerName(String customerName) {
            this.customerName = customerName;
        }
        public void setCustomerAddress(String customerAddress)
        {
            this.customerAddress=customerAddress;
        }
        public String getCustomerAddress()
        {
            return customerAddress;
        }
}
----------vendor.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- vendor.hbm.xml -->
<hibernate-mapping>
  <class name="Vendor" table="vendor">
    <id name="vendorId" column="vid">
      <generator class="assigned"/>
    </id>
    <property name="vendorName" column="vname" length="10"/>
  </class>
</hibernate-mapping>
----------customer.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- customer.hbm.xml -->
<hibernate-mapping>
```

```
    <class name="Customer" table="customer">
      <id name="customerId" column="custid">
        <generator class="assigned"/>
      </id>
      <property name="customerName" column="custname" length="10"/>
      <property name="customerAddress" column="custadd" length="10"/>
      <many-to-one name="vendor" class="Vendor" column="venid" cascade="all"
                                        lazy="proxy"/>
   </class>
</hibernate-mapping>
-------------------------InsertClient.java-----------------------------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
      public static void main(String args[])
      {
            SessionFactory factory = new Configuration().configure().buildSessionFactory();
            Session session = factory.openSession();
            //Parent object
            Vendor v = new Vendor();
            v.setVendorId(111);
            v.setVendorName("IBM");
            //Child object-1
            Customer c1 = new Customer();
            c1.setCustomerId(501);
            c1.setCustomerName("INFY");
            c1.setCustomerAddress("HYD");
            //Child object -2
            Customer c2 = new Customer();
            c2.setCustomerId(502);
            c2.setCustomerName("TCS");
            c2.setCustomerAddress("HYD");
            //Child object -3
            Customer c3 = new Customer();
            c3.setCustomerId(503);
            c3.setCustomerName("VERIZON");
            c3.setCustomerAddress("US");
            //add parent object to child objects
            c1.setVendor(v);
            c2.setVendor(v);
            c3.setVendor(v);
            Transaction tx = session.beginTransaction();
            session.save(c1);
            session.save(c2);
            session.save(c3);
            tx.commit();
            session.close();
            factory.close();
      }
}
---------------------SelectClient.java-----------------------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class SelectClient
{
      public static void main(String args[])
      {
            SessionFactory factory = new Configuration().configure().buildSessionFactory();
            Session session = factory.openSession();
            Customer c =(Customer)session.get(Customer.class,501);
            System.out.println(c.getCustomerName()+"   "+c.getCustomerAddress());
            Vendor v = c.getVendor();
            System.out.println(v.getVendorId()+"   "+v.getVendorName());
            session.close();
            factory.close();
      }
}
```




```
----------DeleteClient.java----------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class DeleteClient
{
    public static void main(String args[])
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        Customer c =(Customer)session.get(Customer.class,501);
        Transaction tx = session.beginTransaction();
        session.delete(c);
        tx.commit();
        session.close();
        factory.close();
    }
}
================================================================================
    Application-16(One-To-Many Bi-directional Example)
----------Vendor.java----------
//Parent POJO
//Vendor.java
import java.util.*;
public class Vendor
{
    private int vendorId;
    private String vendorName;
    private Set customers;
    public Set getCustomers() {
        return customers;
    }
    public void setCustomers(Set customers) {
        this.customers = customers;
    }
    public int getVendorId() {
        return vendorId;
    }
    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }
    public String getVendorName() {
        return vendorName;
    }
    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
    }
}
----------Customer.java----------
//Child POJO
//Customer.java
public class Customer
{
    private int customerId;
    private String customerName;
    private String customerAddress;
    private Vendor vendor;
    public void setVendor(Vendor vendor)
    {
        this.vendor=vendor;
    }
    public Vendor getVendor()
    {
        return vendor;
    }
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
```




```
            this.customerId = customerId;
        }
        public String getCustomerName() {
            return customerName;
        }
        public void setCustomerName(String customerName) {
            this.customerName = customerName;
        }
        public void setCustomerAddress(String customerAddress)
        {
            this.customerAddress=customerAddress;
        }
        public String getCustomerAddress()
        {
            return customerAddress;
        }
}
----------vendor.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- vendor.hbm.xml -->
<hibernate-mapping>
  <class name="Vendor" table="vendor">
    <id name="vendorId" column="vid">
      <generator class="assigned"/>
    </id>
    <property name="vendorName" column="vname" length="10"/>
    <set name="customers" cascade="all" lazy="true" inverse="true">
      <key column="venid"/>
      <one-to-many class="Customer"/>
    </set>
  </class>
</hibernate-mapping>
----------customer.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- customer.hbm.xml -->
<hibernate-mapping>
  <class name="Customer" table="customer">
    <id name="customerId" column="custid">
      <generator class="assigned"/>
    </id>
    <property name="customerName" column="custname" length="10"/>
    <property name="customerAddress" column="custadd" length="10"/>
    <many-to-one name="vendor" class="Vendor" column="venid" cascade="all"
      lazy="proxy"/>
  </class>
</hibernate-mapping>
----------InsertClient.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
    public static void main(String args[])
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        //Parent object
        Vendor v = new Vendor();
        v.setVendorId(111);
        v.setVendorName("IBM");
        //Child object-1
        Customer c1 = new Customer();
        c1.setCustomerId(501);
        c1.setCustomerName("INFY");
        c1.setCustomerAddress("HYD");
        //Child object -2
```

```
477             Customer  c2 = new  Customer();
478             c2.setCustomerId(502);
479             c2.setCustomerName("TCS");
480             c2.setCustomerAddress("HYD");
481             //Child object -3
482             Customer  c3 = new  Customer();
483             c3.setCustomerId(503);
484             c3.setCustomerName("VERIZON");
485             c3.setCustomerAddress("US");
486              //many-to-one
487             c1.setVendor(v);
488             c2.setVendor(v);
489             c3.setVendor(v);
490             //one-to-many
491             //step1
492             Set   s = new  HashSet();
493             s.add(c1);
494             s.add(c2);
495             s.add(c3);
496             //step2
497             v.setCustomers(s);
498             Transaction  tx = session.beginTransaction();
499             session.save(c1); / session.save(v)
500             tx.commit();
501             session.close();
502             factory.close();
503       }
504  }
505  ================================================================================
506       Application-17(Many-To-Many Example)
507  ----------Student.java----------
508  import java.util.*;
509  public class Student
510  {
511       private int studentId;
512       private String studentName;
513       private String studentGrp;
514       private Set courses;
515       public Set getCourses() {
516            return courses;
517       }
518       public void setCourses(Set courses) {
519            this.courses = courses;
520       }
521       public String getStudentGrp() {
522            return studentGrp;
523       }
524       public void setStudentGrp(String studentGrp) {
525            this.studentGrp = studentGrp;
526       }
527       public int getStudentId() {
528            return studentId;
529       }
530       public void setStudentId(int studentId) {
531            this.studentId = studentId;
532       }
533       public String getStudentName() {
534            return studentName;
535       }
536       public void setStudentName(String studentName) {
537            this.studentName = studentName;
538       }
539  }
540  ----------Course.java----------
541  import java.util.*;
542  public class Course
543  {
544       private int courseId;
```

```
        private String courseName;
        private int duration;
        private Set students;
        public int getCourseId() {
              return courseId;
        }
        public void setCourseId(int courseId) {
              this.courseId = courseId;
        }
        public String getCourseName() {
              return courseName;
        }
        public void setCourseName(String courseName) {
              this.courseName = courseName;
        }
        public int getDuration() {
              return duration;
        }
        public void setDuration(int duration) {
              this.duration = duration;
        }
        public Set getStudents() {
              return students;
        }
        public void setStudents(Set students) {
              this.students = students;
        }
}
----------student.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- student.hbm.xml -->
<hibernate-mapping>
  <class name="Student"  table="student">
     <id  name="studentId" column="sid"/>
     <property name="studentName" column="sname" length="10"/>
     <property name="studentGrp" column="grp" length="10"/>
      <set name="courses"  table="students_courses"  cascade="all">
         <key column="sid_fk"/>
         <many-to-many class="Course"  column="cid_fk"/>
      </set>
  </class>
</hibernate-mapping>
----------course.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--  course.hbm.xml -->
<hibernate-mapping>
 <class name="Course"  table="course">
     <id name="courseId" column="cid"/>
      <property name="courseName" column="cname" length="10"/>
      <property name="duration"/>
      <set name="students" table="students_courses" cascade="all">
        <key column="cid_fk"/>
        <many-to-many class="Student"  column="sid_fk"/>
      </set>
  </class>
</hibernate-mapping>
----------InsertClient.java----------
import org.hibernate.*;
import org.hibernate.cfg.*;
import java.util.*;
public class  InsertClient
{
      public static void main(String args[ ])
      {
            SessionFactory factory=new Configuration().configure().buildSessionFactory();
            Session  session = factory.openSession();
```

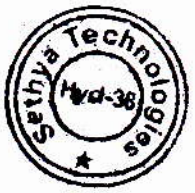
```
613             //Student obj--1
614             Student s1 = new Student();
615             s1.setStudentId(1001);
616             s1.setStudentName("aaa");
617             s1.setStudentGrp("BTech");
618             //Student obj -- 2
619             Student  s2=new Student();
620             s2.setStudentId(1002);
621             s2.setStudentName("bbb");
622             s2.setStudentGrp("MCA");
623             //Course obj -- 1
624             Course c1 = new Course();
625             c1.setCourseId(901);
626             c1.setCourseName("java");
627             c1.setDuration(150);
628             //Course obj -- 2
629             Course c2 = new Course();
630             c2.setCourseId(902);
631             c2.setCourseName("oracle");
632             c2.setDuration(100);
633             Set  set = new  HashSet();
634             set.add(c1);
635             set.add(c2);
636             s1.setCourses(set);
637             s2.setCourses(set);
638             Transaction tx = session.beginTransaction();
639             session.save(s1);
640             session.save(s2);
641             tx.commit();
642             session.close();
643             factory.close();
644         }
645 }
646 ----------AdditionalCourse.java----------
647 import org.hibernate.*;
648 import org.hibernate.cfg.*;
649 import java.util.*;
650 public class  AdditionalCourse
651 {
652         public static void main(String args[ ])
653         {
654             SessionFactory factory=new Configuration().configure().buildSessionFactory();
655             Session  session = factory.openSession();
656             //step-1
657             Student  s=(Student)session.get(Student.class,1001);
658             //step-2
659             Set  set = s.getCourses();
660             //step-3
661             Course  c3 = new Course();
662             c3.setCourseId(903);
663             c3.setCourseName("Cpp");
664             c3.setDuration(50);
665             //step-4
666             Transaction  tx = session.beginTransaction();
667             set.add(c3);
668             tx.commit();
669             session.close();
670             factory.close();
671         }
672 }
673 ----------DeleteCourse.java----------
674 import org.hibernate.*;
675 import org.hibernate.cfg.*;
676 import java.util.*;
677 public class  DeleteCourse
678 {
679         public static void main(String args[ ])
680         {
```

```
            SessionFactory factory=new Configuration().configure().buildSessionFactory();
            Session  session = factory.openSession();
            //step-1
            Student  s=(Student)session.get(Student.class,1001);
            //step-2
            Set set = s.getCourses();
            //step-3
            Course  c =(Course)session.get(Course.class,902);
            //step-4
            Transaction  tx = session.beginTransaction();
            set.remove(c);
            tx.commit();
            session.close();
            factory.close();
      }
}
==========================================================================================
      Application-18(One-To-One  with  Foreign Key Example)
----------Person.java----------
//Person.java
public class Person
{
      private int personId;
      private String personName;
      public void setPersonId(int personId)
      {
            this.personId=personId;
      }
      public int getPersonId()
      {
            return personId;
      }
      public void setPersonName(String personName)
      {
            this.personName=personName;
      }
      public String getPersonName()
      {
            return personName;
      }
};
----------License.java----------
//License.java
import java.util.*;
public class License
{
      private int licenseId;
      private Date issuedDate;
      private Date expireDate;
      private Person person;
      public void setLicenseId(int licenseId)
      {
            this.licenseId=licenseId;
      }
      public int getLicenseId()
      {
            return licenseId;
      }
      public void setIssuedDate(Date issuedDate)
      {
            this.issuedDate=issuedDate;
      }
      public Date  getIssuedDate()
      {
            return issuedDate;
      }
      public void setExpireDate(Date expireDate)
      {
```


```
            this.expireDate=expireDate;
        }
        public Date  getExpireDate()
        {
            return expireDate;
        }
        public void setPerson(Person person)
        {
            this.person=person;
        }
        public Person getPerson()
        {
            return person;
        }
}
----------person.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- person.hbm.xml -->
<hibernate-mapping>
  <class  name="Person"  table="person">
    <id  name="personId"  column="pid"/>
      <property name="personName" column="pname" length="10"/>
  </class>
</hibernate-mapping>
----------license.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- license.hbm.xml -->
<hibernate-mapping>
  <class  name="License"  table="license">
    <id  name="licenseId"  column="lid"/>
      <property name="issuedDate" column="idate"/>
      <property name="expireDate" column="edate"/>
      <many-to-one name="person"  class="Person"
       column="per_id"  unique="true"  not-null="true" cascade="all"/>
  </class>
</hibernate-mapping>
----------InsertClient.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
class InsertClient
{
    public static void main(String[] args)
    {
        SessionFactory factory= new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        //Person object
        Person p = new Person();
        p.setPersonId(1101);
        p.setPersonName("ABC");
        //License object
        License l1= new License();
        l1.setLicenseId(1010);
        l1.setIssuedDate(new Date());
        l1.setExpireDate(new Date());
    /*  License l2= new License();
        l2.setLicenseId(1011);
        l2.setIssuedDate(new Date());
        l2.setExpireDate(new Date()); */
        l1.setPerson(p);
    //  l2.setPerson(p);
    Transaction tx = session.beginTransaction();
        session.save(l1);
        //session.save(l2);
        tx.commit();
        session.close();
```

```
817             factory.close();
818         }
819     }
820     ================================================================================
821         Application-19(One-To-One with Primary key)
822     ----------Person.java----------
823     //Person.java
824     public class Person
825     {
826         private int personId;
827         private String personName;
828         public void setPersonId(int personId)
829         {
830             this.personId=personId;
831         }
832         public int getPersonId()
833         {
834             return personId;
835         }
836         public void setPersonName(String personName)
837         {
838             this.personName=personName;
839         }
840         public String getPersonName()
841         {
842             return personName;
843         }
844     };
845     ----------License.java----------
846     //License.java
847     import java.util.*;
848     public class License
849     {
850         private int licenseId;
851         private Date issuedDate;
852         private Date expireDate;
853         private Person person;
854         public void setLicenseId(int licenseId)
855         {
856             this.licenseId=licenseId;
857         }
858         public int getLicenseId()
859         {
860             return licenseId;
861         }
862         public void setIssuedDate(Date issuedDate)
863         {
864             this.issuedDate=issuedDate;
865         }
866         public Date  getIssuedDate()
867         {
868             return issuedDate;
869         }
870         public void setExpireDate(Date expireDate)
871         {
872             this.expireDate=expireDate;
873         }
874         public Date  getExpireDate()
875         {
876             return expireDate;
877         }
878         public void setPerson(Person person)
879         {
880             this.person=person;
881         }
882         public Person getPerson()
883         {
884             return person;
```

```
    }
}
----------person.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- person.hbm.xml -->
<hibernate-mapping>
  <class name="Person" table="person">
    <id name="personId" column="pid"/>
      <property name="personName" column="pname" length="10"/>
  </class>
</hibernate-mapping>
----------license.hbm.xml----------
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- license.hbm.xml -->
<hibernate-mapping>
  <class name="License" table="license">
    <id name="licenseId" column="lid">
         <generator class="foreign">
              <param name="property">person</param>
       </generator>
    </id>
      <property name="issuedDate" column="idate"/>
      <property name="expireDate" column="edate"/>
      <one-to-one name="person" class="Person"
       cascade="all"/>
 </class>
</hibernate-mapping>
----------InsertClient.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
class InsertClient
{
     public static void main(String[] args)
     {
          SessionFactory factory= new Configuration().configure().buildSessionFactory();
          Session session = factory.openSession();
          //Person object
          Person p = new Person();
          p.setPersonId(1101);
          p.setPersonName("ABC");
          //License object
          License l = new License();
          l.setIssuedDate(new Date());
          l.setExpireDate(new Date());
          l.setPerson(p);
          Transaction tx = session.beginTransaction();
          session.save(l);
          tx.commit();
          session.close();
          factory.close();
     }
}
==========
     Application-20(component mapping example)
----------Person.java----------
//Person.java
import java.util.*;
public class Person
{
     private int personId;
     private PName pname;
     private Date dob;
     public void setPersonId(int personId)
     {
          this.personId=personId;
```

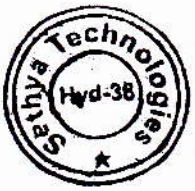



```
953         }
954         public int getPersonId()
955         {
956             return personId;
957         }
958         public void  setPname(PName pname)
959         {
960             this.pname=pname;
961         }
962         public PName getPname()
963         {
964             return pname;
965         }
966         public  void setDob(Date dob)
967         {
968             this.dob=dob;
969         }
970         public Date  getDob()
971         {
972             return dob;
973         }
974     };
975     ----------PName.java----------
976     //PName.java
977     public class PName
978     {
979         private char initial;
980         private String firstName;
981         private String lastName;
982         public void  setInitial(char initial)
983         {
984             this.initial=initial;
985         }
986         public char getInitial()
987         {
988             return initial;
989         }
990         public void setFirstName(String firstName)
991         {
992             this.firstName=firstName;
993         }
994         public String getFirstName()
995         {
996             return firstName;
997         }
998         public void setLastName(String lastName)
999         {
1000            this.lastName=lastName;
1001        }
1002        public String getLastName()
1003        {
1004            return lastName;
1005        }
1006    }
1007    ----------person.hbm.xml----------
1008    <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
1009      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
1010    <!-- person.hbm.xml -->
1011    <hibernate-mapping>
1012     <class  name="Person"  table="persons">
1013       <id name="personId"  column="pid"/>
1014          <property name="dob"  column="dob"/>
1015          <component name="pname">
1016             <property name="initial"  column="init" length="5"/>
1017                <property name="firstName"  column="fname"  length="10"/>
1018                <property name="lastName"  column="lname"  length="10"/>
1019          </component>
1020     </class>
```

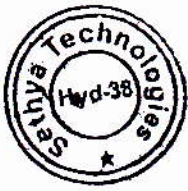
```
</hibernate-mapping>

----------Client.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
class Client
{
    public static void main(String[ ] args)
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        PName p = new PName();
        p.setInitial('S');
        p.setFirstName("Abc");
        p.setLastName("Xyz");
        Person pr = new Person();
        pr.setPersonId(111);
        pr.setDob(new Date());
        //assign PName object to Person
        pr.setPname(p);
        Transaction tx = session.beginTransaction();
        session.save(pr);
        tx.commit();
        session.close();
        factory.close();
    }
}
================================================================================
    Application-21(A First example using Annotations)
----------Student.java----------
//Student.java
import javax.persistence.*;
@Entity
@Table(name="student_table")
public class Student
{
    @Id
    @Column(name="stno")
    private Integer sno;
    @Column(name="sname",length=10)
    private String sname;
    @Column(name="stadd",length=10)
    private String address;
    public void setAddress(String address) {
        this.address = address;
    }
    public String getAddress() {
        return address;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSname() {
        return sname;
    }
    public void setSno(Integer sno) {
        this.sno = sno;
    }
    public Integer getSno() {
        return sno;
    }
}
----------hibernate.cfg.xml----------
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
```

```
<session-factory>
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
<property name="hibernate.connection.username">scott</property>
<property name="hibernate.connection.password">tiger</property>
<property name="show_sql">true </property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
<mapping     class="Student"/>
</session-factory>
</hibernate-configuration>
-------------------------InsertClient.java----------------------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
      public static void main(String args[])
      {
            SessionFactory factory=new Configuration().configure().buildSessionFactory();
            Session ses = factory.openSession();
            Student s = new Student();
            s.setSno(101);
            s.setSname("ssreddy");
            s.setAddress("hyd");
            Transaction tx = ses.beginTransaction();
            ses.persist(s);
            tx.commit();
            ses.close();
            factory.close();
      }
}
=============================================================================================
      Application-22(Table per class  using  annotations)
--------------------Payment.java-----------------------------
import javax.persistence.*;
@Entity
@Table(name="PAYMENT")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="pmode",discriminatorType=DiscriminatorType.STRING,length=4)
public class Payment
{
      @Id
      @Column(name="payid")
      private Integer paymentId;
      @Column(name="amount")
      private Double amount;
      public Double getAmount() {
            return amount;
      }
      public void setAmount(Double amount) {
            this.amount = amount;
      }
      public Integer getPaymentId() {
            return paymentId;
      }
      public void setPaymentId(Integer paymentId) {
            this.paymentId = paymentId;
      }
}
---------------------ChequePayment.java----------------------
//ChequePayment.java
import javax.persistence.*;
@Entity
@DiscriminatorValue(value="CH")
public class ChequePayment extends Payment
{
      @Column(name="chtype",length=10)
      private String chequeType;
```




```
        public String getChequeType() {
            return chequeType;
        }
        public void setChequeType(String chequeType) {
            this.chequeType = chequeType;
        }
}
----------CreditCardPayment.java----------
//CreditCardPayment.java
import javax.persistence.*;
@Entity
@DiscriminatorValue(value="CR")
public class CreditCardPayment extends Payment
{
        @Column(name="cctype",length=10)
        private String cardType;
        public String getCardType() {
            return cardType;
        }
        public void setCardType(String cardType) {
            this.cardType = cardType;
        }
}
----------hibernate.cfg.xml----------
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
<property name="hibernate.connection.username">scott</property>
<property name="hibernate.connection.password">tiger</property>
<property name="show_sql">true </property>
<property name="hbm2ddl.auto">update</property>
<property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
<mapping class="Payment"/>
<mapping  class="CreditCardPayment"/>
<mapping  class="ChequePayment"/>
</session-factory>
</hibernate-configuration>
----------InsertClient.java----------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
        public static void main(String[] args)
        {
            SessionFactory factory = new Configuration().configure().buildSessionFactory();
            Session session=factory.openSession();
            CreditCardPayment  ccp = new CreditCardPayment();
            ccp.setPaymentId(101);
            ccp.setAmount(5000.0);
            ccp.setCardType("VISA");
            ChequePayment  cp = new  ChequePayment();
            cp.setPaymentId(102);
            cp.setAmount(8000.0);
            cp.setChequeType("ORDER");
            Transaction  tx = session.beginTransaction();
            session.save(ccp);
            session.save(cp);
            tx.commit();
            session.close();
            factory.close();
        }
}
=====================================================================
        Application-23(Table per  subclass  using  annotations)
```



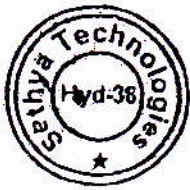

```
1225  ----------Payment.java----------
1226  //Payment.java
1227  import javax.persistence.*;
1228  @Entity
1229  @Table(name="PAYMENT")
1230  @Inheritance(strategy=InheritanceType.JOINED)
1231  public class Payment
1232  {
1233        @Id
1234        @Column(name="payid")
1235        private Integer paymentId;
1236        @Column(name="amount")
1237        private Double amount;
1238        public Double getAmount() {
1239              return amount;
1240        }
1241        public void setAmount(Double amount) {
1242              this.amount = amount;
1243        }
1244        public Integer getPaymentId() {
1245              return paymentId;
1246        }
1247        public void setPaymentId(Integer paymentId) {
1248              this.paymentId = paymentId;
1249        }
1250  }
1251  ----------CreditCardPayment.java----------
1252  //CreditCardPayment.java
1253  import javax.persistence.*;
1254  @Entity
1255  @Table(name="CREDIT_TABLE")
1256  @PrimaryKeyJoinColumn(name="pid")
1257  public class CreditCardPayment extends Payment
1258  {
1259        @Column(name="cctype",length=10)
1260        private String cardType;
1261        public String getCardType() {
1262              return cardType;
1263        }
1264        public void setCardType(String cardType) {
1265              this.cardType = cardType;
1266        }
1267  }
1268  ----------ChequePayment.java----------
1269  //ChequePayment.java
1270  import javax.persistence.*;
1271  @Entity
1272  @Table(name="CHEQUE_TABLE")
1273  @PrimaryKeyJoinColumn(name="p_id")
1274  public class ChequePayment extends Payment
1275  {
1276        @Column(name="chtype",length=10)
1277        private String chequeType;
1278        public String getChequeType() {
1279              return chequeType;
1280        }
1281        public void setChequeType(String chequeType) {
1282              this.chequeType = chequeType;
1283        }
1284  }
1285  ----------InsertClient.java----------
1286  import org.hibernate.*;
1287  import org.hibernate.cfg.*;
1288  public class InsertClient
1289  {
1290        public static void main(String[] args)
1291        {
1292              SessionFactory factory = new Configuration().configure().buildSessionFactory();
```

```
            Session session=factory.openSession();
            CreditCardPayment  ccp = new CreditCardPayment();
            ccp.setPaymentId(101);
            ccp.setAmount(5000.0);
            ccp.setCardType("VISA");
            ChequePayment  cp = new  ChequePayment();
            cp.setPaymentId(102);
            cp.setAmount(8000.0);
            cp.setChequeType("ORDER");
            Transaction  tx = session.beginTransaction();
            session.save(ccp);
            session.save(cp);
            tx.commit();
            session.close();
            factory.close();
        }
}
================================================================================
        Application-24(One-To-Many Relationship using Annotations)
----------------------------Vendor.java-------------------------------------
//Parent POJO
//Vendor.java
import javax.persistence.*;
import java.util.*;
@Entity
@Table(name="vendor")
public class Vendor
{
        @Id
        @Column(name="vid")
        private int vendorId;
        @Column(name="vname",length=10)
        private String vendorName;
        @OneToMany(targetEntity=Customer.class,cascade=CascadeType.ALL)
        @JoinColumn(name="venid",referencedColumnName="vid")
        private Set  customers;
        public Set getCustomers() {
                return customers;
        }
        public void setCustomers(Set customers) {
                this.customers = customers;
        }
        public int getVendorId() {
                return vendorId;
        }
        public void setVendorId(int vendorId) {
                this.vendorId = vendorId;
        }
        public String getVendorName() {
                return vendorName;
        }
        public void setVendorName(String vendorName) {
                this.vendorName = vendorName;
        }
}
----------------------Customer.java-----------------------------
//Child  POJO
//Customer.java
import  javax.persistence.*;
@Entity
@Table(name="customer")
public class Customer
{
        @Id
        @Column(name="custid")
        private int customerId;
        @Column(name="custname",length=10)
        private String customerName;
```


```
    @Column(name="custadd",length=10)
    private String customerAddress;
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setCustomerAddress(String customerAddress)
    {
        this.customerAddress=customerAddress;
    }
    public String getCustomerAddress()
    {
        return customerAddress;
    }

}
-------------------------hibernate.cfg.xml-----------------------------
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<!-- hibernate.cfg.xml -->
<hibernate-configuration>
  <session-factory>
        <!-- connection properties -->
    <property name="connection.driver_class">oracle.jdbc.OracleDriver</property>
      <property name="connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
      <property name="connection.username">scott</property>
      <property name="connection.password">tiger</property>
      <!-- hibernate properties -->
      <property name="dialect">org.hibernate.dialect.OracleDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <!-- mapping classes -->
      <mapping class="Vendor"/>
      <mapping class="Customer"/>
 </session-factory>
</hibernate-configuration>
-------------------------InsertClient.java-----------------------------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
    public static void main(String args[ ])
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        //Parent object
        Vendor v = new Vendor();
        v.setVendorId(111);
        v.setVendorName("IBM");
        //Child object-1
        Customer c1 = new Customer();
        c1.setCustomerId(501);
        c1.setCustomerName("INFY");
        c1.setCustomerAddress("HYD");
        //Child object -2
        Customer c2 = new Customer();
        c2.setCustomerId(502);
        c2.setCustomerName("TCS");
```

```
1429            c2.setCustomerAddress("HYD");
1430            //Child object -3
1431            Customer  c3 = new  Customer();
1432            c3.setCustomerId(503);
1433            c3.setCustomerName("VERIZON");
1434            c3.setCustomerAddress("US");
1435            //step1
1436            Set   s = new  HashSet();
1437            s.add(c1);
1438            s.add(c2);
1439            s.add(c3);
1440            //step2
1441            v.setCustomers(s);
1442            Transaction  tx = session.beginTransaction();
1443            session.save(v);
1444            tx.commit();
1445            session.close();
1446            factory.close();
1447        }
1448  }
1449  ==========================================================================
1450        Application-25(Many-To-One  using  Annotations)
1451  ----------------------Vendor.java-----------------------------------
1452  //Vendor.java
1453  import  javax.persistence.*;
1454  @Entity
1455  @Table(name="vendor")
1456  public class Vendor
1457  {
1458        @Id
1459        @Column(name="vid")
1460        private int vendorId;
1461        @Column(name="vname",length=10)
1462        private String vendorName;
1463        public int getVendorId() {
1464              return vendorId;
1465        }
1466        public void setVendorId(int vendorId) {
1467              this.vendorId = vendorId;
1468        }
1469        public String getVendorName() {
1470              return vendorName;
1471        }
1472        public void setVendorName(String vendorName) {
1473              this.vendorName = vendorName;
1474        }
1475  }
1476  ----------------------Customer.java----------------------------
1477  //Customer.java
1478  import  javax.persistence.*;
1479  @Entity
1480  @Table(name="customer")
1481  public class Customer
1482  {
1483        @Id
1484        @Column(name="custid")
1485        private int customerId;
1486        @Column(name="custname",length=10)
1487        private String customerName;
1488        @Column(name="custadd",length=10)
1489        private  String  customerAddress;
1490        @ManyToOne(targetEntity=Vendor.class,cascade=CascadeType.ALL,fetch=FetchType.EAGER)
1491        @JoinColumn(name="venid")
1492        private  Vendor  vendor;
1493        public  void  setVendor(Vendor vendor)
1494        {
1495              this.vendor=vendor;
1496        }
```

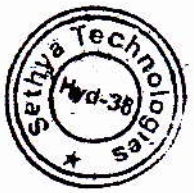



```
    public Vendor getVendor()
    {
        return vendor;
    }
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
    public void setCustomerAddress(String customerAddress)
    {
        this.customerAddress=customerAddress;
    }
    public String getCustomerAddress()
    {
        return customerAddress;
    }
}
----------hibernate.cfg.xml----------
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<!-- hibernate.cfg.xml -->
<hibernate-configuration>
  <session-factory>
        <!-- connection properties -->
    <property name="connection.driver_class">oracle.jdbc.OracleDriver</property>
      <property name="connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
      <property name="connection.username">scott</property>
      <property name="connection.password">tiger</property>
      <!-- hibernate properties -->
      <property name="dialect">org.hibernate.dialect.OracleDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <!-- mapping classes -->
      <mapping class="Vendor"/>
      <mapping class="Customer"/>
 </session-factory>
</hibernate-configuration>
----------InsertClient.java----------
import org.hibernate.*;
import org.hibernate.cfg.*;
public class InsertClient
{
    public static void main(String args[])
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        //Parent object
        Vendor v = new Vendor();
        v.setVendorId(111);
        v.setVendorName("IBM");
        //Child object-1
        Customer c1 = new Customer();
        c1.setCustomerId(501);
        c1.setCustomerName("INFY");
        c1.setCustomerAddress("HYD");
        //Child object -2
        Customer c2 = new Customer();
        c2.setCustomerId(502);
        c2.setCustomerName("TCS");
```

```
            c2.setCustomerAddress("HYD");
            //Child object -3
            Customer  c3 = new  Customer();
            c3.setCustomerId(503);
            c3.setCustomerName("VERIZON");
            c3.setCustomerAddress("US");
            //add parent  object  to  child objects
            c1.setVendor(v);
            c2.setVendor(v);
            c3.setVendor(v);
            Transaction  tx = session.beginTransaction();
            session.save(c1);
            session.save(c2);
            session.save(c3);
            tx.commit();
            session.close();
            factory.close();
      }
}
==========================================================================================
      Application-26(Many-To-Many  using  Annotations)
----------------------Item.java-----------------------------------
//Item.java
import javax.persistence.*;
import java.util.*;
@Entity
@Table(name="items")
public class Item
{
      @Id
      @Column(name="item_id")
      private int itemId;
      @Column(name="item_name",length=10)
      private String itemName;
      @ManyToMany(targetEntity=Category.class,mappedBy="items")
      private  Set   categories;
      public Set getCategories() {
            return categories;
      }
      public void setCategories(Set  categories) {
            this.categories = categories;
      }
      public int getItemId() {
            return itemId;
      }
      public void setItemId(int itemId) {
            this.itemId = itemId;
      }
      public String getItemName() {
            return itemName;
      }
      public void setItemName(String itemName) {
            this.itemName = itemName;
      }
}
----------------------Category.java-----------------------------
//Category.java
import javax.persistence.*;
import java.util.*;
@Entity
@Table(name="categories")
public class Category
{
      @Id
      @Column(name="cat_id")
       private int categoryId;
      @Column(name="cat_name",length=10)
      private String categoryName;
```

```
1633        @ManyToMany(targetEntity=Item.class,cascade=CascadeType.ALL)
1634        @JoinTable(name="categories_items",
1635          joinColumns=@JoinColumn(name="cat_id_fk",referencedColumnName="cat_id"),
1636          inverseJoinColumns=@JoinColumn(name="item_id_fk",referencedColumnName="item_id"))
1637        private Set  items;
1638        public int getCategoryId() {
1639            return categoryId;
1640        }
1641        public void setCategoryId(int categoryId) {
1642            this.categoryId = categoryId;
1643        }
1644        public Set getItems() {
1645            return items;
1646        }
1647        public void setItems(Set items) {
1648            this.items = items;
1649        }
1650        public String getCategoryName() {
1651            return categoryName;
1652        }
1653        public void setCategoryName(String categoryname) {
1654            this.categoryName = categoryname;
1655        }
1656  }
1657  ----------hibernate.cfg.xml----------
1658  <!DOCTYPE hibernate-configuration PUBLIC
1659        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
1660        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
1661  <hibernate-configuration>
1662  <session-factory>
1663  <property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
1664  <property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
1665  <property name="hibernate.connection.username">scott</property>
1666  <property name="hibernate.connection.password">tiger</property>
1667  <property name="show_sql">true </property>
1668  <property name="hbm2ddl.auto">update</property>
1669  <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
1670  <mapping class="Category"/>
1671  <mapping class="Item"/>
1672  </session-factory>
1673  </hibernate-configuration>
1674  ----------InsertClient.java----------
1675  import org.hibernate.*;
1676  import org.hibernate.cfg.*;
1677  import java.util.*;
1678  public class InsertClient {
1679        public static void main(String args[])
1680        {
1681            SessionFactory  factory=new Configuration().configure().buildSessionFactory();
1682            Session  ses = factory.openSession();
1683            Category c1 = new Category();
1684            c1.setCategoryId(1);
1685            c1.setCategoryName("aaa");
1686            Category c2=new Category();
1687            c2.setCategoryId(2);
1688            c2.setCategoryName("bbb");
1689            Item i1=new Item();
1690            i1.setItemId(101);
1691            i1.setItemName("item1");
1692            Item i2 = new Item();
1693            i2.setItemId(102);
1694            i2.setItemName("item2");
1695            Set  s =new HashSet();
1696            s.add(i1);
1697            s.add(i2);
1698            c1.setItems(s);
1699            c2.setItems(s);
1700            Transaction tx = ses.beginTransaction();
```

```
            ses.save(c1);
            ses.save(c2);
            tx.commit();
            ses.close();
        }
}
==========================================================================
        Application-27(One-To-One using Annotations)
--------------------Person.java---------------------------------
//Person.java
import javax.persistence.*;
@Entity
@Table(name="person")
public class Person
{
    @Id
    @Column(name="per_id")
    private int personId;
    @Column(name="per_name",length=10)
    private String personName;
    public void setPersonId(int personId)
    {
        this.personId=personId;
    }
    public int getPersonId()
    {
        return personId;
    }
    public void setPersonName(String personName)
    {
        this.personName=personName;
    }
    public String getPersonName()
    {
        return personName;
    }
};
--------------------License.java---------------------------------
//License.java
import java.util.*;
import javax.persistence.*;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Parameter;
@Entity
@Table(name="License")
public class License
{
    @GenericGenerator(name="gen1",strategy="foreign",
            parameters=@Parameter(name="property",value="person"))
    @Id
    @GeneratedValue(generator="gen1")
    @Column(name="lic_id")
    private int licenseId;
    @Column(name="idate")
    private Date issuedDate;
    @Column(name="edate")
    private Date expireDate;
    @OneToOne(targetEntity=Person.class,cascade=CascadeType.ALL)
    @PrimaryKeyJoinColumn
    private Person person;
    public void setLicenseId(int licenseId)
    {
        this.licenseId=licenseId;
    }
    public int getLicenseId()
    {
        return licenseId;
    }
```




```
1769        public void setIssuedDate(Date issuedDate)
1770        {
1771            this.issuedDate=issuedDate;
1772        }
1773        public Date  getIssuedDate()
1774        {
1775            return issuedDate;
1776        }
1777        public void setExpireDate(Date expireDate)
1778        {
1779            this.expireDate=expireDate;
1780        }
1781        public Date  getExpireDate()
1782        {
1783            return expireDate;
1784        }
1785        public void setPerson(Person person)
1786        {
1787            this.person=person;
1788        }
1789        public Person getPerson()
1790        {
1791            return person;
1792        }
1793    }
1794    ----------hibernate.cfg.xml----------
1795    <!DOCTYPE hibernate-configuration PUBLIC
1796        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
1797        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
1798     <!-- hibernate.cfg.xml -->
1799     <hibernate-configuration>
1800       <session-factory>
1801            <!-- connection properties -->
1802         <property name="connection.driver_class">oracle.jdbc.OracleDriver</property>
1803           <property name="connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
1804           <property name="connection.username">scott</property>
1805           <property name="connection.password">tiger</property>
1806           <!-- hibernate properties -->
1807           <property name="dialect">org.hibernate.dialect.OracleDialect</property>
1808           <property name="show_sql">true</property>
1809           <property name="hbm2ddl.auto">update</property>
1810           <!-- mapping files -->
1811           <mapping class="Person"/>
1812           <mapping class="License"/>
1813      </session-factory>
1814    </hibernate-configuration>
1815    ----------InsertClient.java----------
1816    import java.util.*;
1817    import org.hibernate.*;
1818    import org.hibernate.cfg.*;
1819    class InsertClient
1820    {
1821        public static void main(String[] args)
1822        {
1823            SessionFactory factory= new Configuration().configure().buildSessionFactory();
1824            Session session = factory.openSession();
1825            //Person object
1826            Person p = new Person();
1827            p.setPersonId(1101);
1828            p.setPersonName("ABC");
1829            //License object
1830            License l = new License();
1831            l.setIssuedDate(new Date());
1832            l.setExpireDate(new Date());
1833            l.setPerson(p);
1834            Transaction tx = session.beginTransaction();
1835            session.save(l);
1836            tx.commit();
```

```
            session.close();
            factory.close();
        }
}
==========================================================================
        Application-28(Component Test using Annotations)
----------Person.java----------
//Person.java
import java.util.*;
import javax.persistence.*;
@Entity
@Table(name="persons")
public class Person
{
    @Id
    @Column(name="pid")
    private int personId;
    @Embedded
    private PName pname;
    @Column(name="dob")
    private Date dob;
    public void setPersonId(int personId)
    {
        this.personId=personId;
    }
    public int getPersonId()
    {
        return personId;
    }
    public void setPname(PName pname)
    {
        this.pname=pname;
    }
    public PName getPname()
    {
        return pname;
    }
    public void setDob(Date dob)
    {
        this.dob=dob;
    }
    public Date getDob()
    {
        return dob;
    }
};
----------PName.java----------
//PName.java
import javax.persistence.*;
@Embeddable
public class PName
{
    @Column(name="I",length=1)
    private char initial;
    @Column(name="fname",length=10)
    private String firstName;
    @Column(name="lname",length=10)
    private String lastName;
    public void setInitial(char initial)
    {
        this.initial=initial;
    }
    public char getInitial()
    {
        return initial;
    }
    public void setFirstName(String firstName)
    {
```

```
            this.firstName=firstName;
        }
        public String getFirstName()
        {
            return firstName;
        }
        public void setLastName(String lastName)
        {
            this.lastName=lastName;
        }
        public String getLastName()
        {
            return lastName;
        }
}
----------hibernate.cfg.xml----------
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<!-- hibernate.cfg.xml -->
<hibernate-configuration>
  <session-factory>
        <!-- connection properties -->
   <property name="connection.driver_class">oracle.jdbc.OracleDriver</property>
     <property name="connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
     <property name="connection.username">scott</property>
     <property name="connection.password">tiger</property>
     <!-- hibernate properties -->
     <property name="dialect">org.hibernate.dialect.OracleDialect</property>
     <property name="show_sql">true</property>
     <property name="hbm2ddl.auto">update</property>
     <!-- mapping files -->
     <mapping class="Person"/>
     <mapping class="PName"/>
  </session-factory>
</hibernate-configuration>
----------Client.java----------
import java.util.*;
import org.hibernate.*;
import org.hibernate.cfg.*;
class Client
{
    public static void main(String[ ] args)
    {
        SessionFactory factory = new Configuration().configure().buildSessionFactory();
        Session session = factory.openSession();
        PName p = new PName();
        p.setInitial('S');
        p.setFirstName("Abc");
        p.setLastName("Xyz");
        Person pr = new Person();
        pr.setPersonId(111);
        pr.setDob(new Date());
        //assign PName object to Person
        pr.setPname(p);
        Transaction tx = session.beginTransaction();
        session.save(pr);
        tx.commit();
        session.close();
        factory.close();
    }
}
==========================================================================
    Application-29(Easy Hibernate caching)
----------Employee.java----------
//Employee.java (POJO)
public class Employee
{
```

```
1973        private int employeeId;
1974        private String employeeName;
1975        private int employeeSal;
1976        private int deptNumber;
1977        public int getDeptNumber() {
1978             return deptNumber;
1979        }
1980        public void setDeptNumber(int deptNumber) {
1981             this.deptNumber = deptNumber;
1982        }
1983        public String getEmployeeName() {
1984             return employeeName;
1985        }
1986        public void setEmployeeName(String employeeName) {
1987             this.employeeName = employeeName;
1988        }
1989        public int getEmployeeId() {
1990             return employeeId;
1991        }
1992        public void setEmployeeId(int employeeId) {
1993             this.employeeId = employeeId;
1994        }
1995        public int getEmployeeSal() {
1996             return employeeSal;
1997        }
1998        public void setEmployeeSal(int employeeSal) {
1999             this.employeeSal = employeeSal;
2000        }
2001   }
2002   ----------------------------employee.hbm.xml---------------------------------
2003   <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
2004      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
2005   <!-- employee.hbm.xml -->
2006   <hibernate-mapping>
2007    <class name="Employee"  table="employee">
2008     <cache  usage="read-only"/>
2009     <id  name="employeeId" column="empno"/>
2010     <property name="employeeName" column="ename"/>
2011     <property name="employeeSal"  column="sal"/>
2012     <property name="deptNumber"  column="deptno"/>
2013    </class>
2014   </hibernate-mapping>
2015   ----------------------------hibernate.cfg.xml----------------------------
2016   <!DOCTYPE hibernate-configuration PUBLIC
2017         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
2018         "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
2019    <!--  hibernate.cfg.xml -->
2020    <hibernate-configuration>
2021       <session-factory>
2022             <!-- connection properties -->
2023        <property  name="connection.driver_class">oracle.jdbc.OracleDriver</property>
2024          <property  name="connection.url">jdbc:oracle:thin:@localhost:1521:satya</property>
2025          <property  name="connection.username">scott</property>
2026          <property  name="connection.password">tiger</property>
2027          <!-- hibernate properties -->
2028          <property  name="dialect">org.hibernate.dialect.OracleDialect</property>
2029          <property  name="show_sql">true</property>
2030          <property  name="hbm2ddl.auto">update</property>
2031          <property  name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider</property>
2032          <!-- mapping files -->  property name = "hibernate.use_second_level_cache" true </PROPERTY>
2033          <mapping  resource="employee.hbm.xml"/>
2034    </session-factory>
2035   </hibernate-configuration>
2036   ----------------------------ehcache.xml---------------------------------
2037   <!-- ehcache.xml -->
2038   <ehcache>
2039     <defaultCache  maxElementsInMemory="100"
2040                    eternal="false"  -> eternal= false means we are setting some time for the objects in cache.
```


if we dont set any time limits then we should make eternal="true"

```
                    timeToIdleSeconds="120"
                    timeToLiveSeconds="200"/>
  <cache name="Employee"
          maxElementsInMemory="10"
          eternal="false"
                  timeToIdleSeconds="8"
          timeToLiveSeconds="300"
          />
</ehcache>
----------------------client.java-----------------------------------
//client.java
import org.hibernate.*;
import org.hibernate.cfg.*;
public class client
{
    public static void main(String args[])
    {
        SessionFactory factory=new Configuration().configure().buildSessionFactory();

        Session ses1 = factory.openSession();
        Session ses2 = factory.openSession();
        Session ses3 =factory.openSession();

        Employee e1=(Employee)ses1.get(Employee.class,7876);
        System.out.println(e1.getEmployeeName());
        ses1.clear();
        System.out.println("session1  cleared");
        e1=(Employee)ses1.get(Employee.class,7876);
        System.out.println(e1.getEmployeeName());
        System.out.println("================");
        try
        {
            Thread.sleep(6000);
        }
        catch(Exception e)
        { }

        Employee e2=(Employee)ses2.get(Employee.class,7876);
        System.out.println(e2.getEmployeeName());
        System.out.println("================");

        try
        {
            Thread.sleep(6000);
        }
        catch(Exception e)
        { }

        Employee e3=(Employee)ses3.get(Employee.class,7876);
        System.out.println(e3.getEmployeeName());
        ses1.close();
        ses2.close();
        ses3.close();
        factory.close();
    }
}
================================================================================
```

I would try to update our site JavaEra.com everyday with various interesting facts, scenarios and interview questions. Keep visiting regularly.....

Thanks and I wish all the readers all the best in the interviews.

www.JavaEra.com

A Perfect Place for All Java Resources