2016

HIBERNATE

ORM FRAMEWORK

PRASAD NAIDU 9494172257

03-05-2016

ORM is a Methodology (or) The Best-Practices that helps us to Create Objects according to the Relational Model.So that persisting the data that is there in the objects Model into a corresponding Relational Model,or accessing the data that is there in the Relation Model into a corresponding object Model will become easy.Such kind of guidelines which are provided by group of experts,are being called as ORM Standards.

Based on these Guidelines Every Programming language has started making available the relavant API's to work with ORM. Ex:Incase of Java ,If we wanted to persist the data into the Database, then Java has to provide the classes for persisting the data into the Database.

Lets say If Connection,ResultSet,Statement are not being provided by JAVA as part of the JDBC API,Then its difficult to write the logic to persist/access the data easilly by our own.So To make the programmer convienent in accessing the data from the Database, JAVA has provided one API called JDBC.

Similarly ORM provided some guidelines and Best-practices of modeling Objects and RelationalSchema,so that one can be converted into another easilly.Before ORM there is no Standard Mechanism to and Persist the data.

UseCase: Lets Assume If we have Tables like "GAS_CONSUMER" and "REFILL_REQUEST" ,Then A Gas Consumer can place multiple RefillRequests,so here the Relationship is One-to-Many.If we have Tables related in this way we need to create classes also in this way,so that data contained in this objects will be easilly mapped into to the Tables.Whenever we want to access the data from those tables also we can retrieve and populate the data into objects by adopting ORM standards.

Can we use JDBC API in working with persisting the data or not? Yes ,we can write the code in JDBC API and we can design our classes based on ORM Guidelines and we can map the data and we can persist the data in the RelationalModel.But there are problems like Boiler-plate-logic,Maintenance problem and etc.. If we use JDBC API.)

Thats why Based on thes Guidelines many of the programming languages like .NET started uptaking these ORM Guidelines.

So the programing languages has to provide, the relavant set of classes,to make the programmer free from writing logic to convert Object into Relation,rather it has to give the object directly to the programmer,and the API provided by programming language will take care of converting the object into relational format based on ORM Guidelines.

Did java provided any API to persist the data interms of Object format?

Initially SunMicroSystems didn't realize the strength of ORM,and it hasn't provided any API to work with ORM,Its badly supported to work with JAVA JDBC API only,Though rest of The Programming languages has provided API's to work with ORM.

Due to this people started rasing questions concerned to the difficulty in working with JDBC API, Meanwhile to fill the gap between JAVA and Relational databases,several Third party API's came into existence ,So one of such Third party API's to work with Relational model incase of Java is "HIBERNATE".

HIBERNATE is not an API provided by SUN people,rather it is an API or a FrameWork that is Developed by Third party organization JBOSS. On successful implementation of Hibernate ,Sub-Sequently multiple vendors started providing different libraries to work with ORM.Some of them are iBATIS,TopLink and this list is un-ending in the space of JAVA to work with ORM.

HIBERNATE

List Of Contents

PART-A

- Session.delete()
- Dynamic-update
- Dynamic-insert
- Session.merge
- Differences between update() and merge()
- Hibernate EntityObject LifeCycle States(Transient━Persistent━Detached━Removed)
- ContextualSessions
  ✔Differences between session.openSession()&session.getCurrentSession().
  ✔Configuring our own Context Session.

PART-B

- Mappings and Its Types
- Impedance-mismatch problems
  ✔granularity✔subTypes✔Identity✔Navigation✔association
- Inheritance Mapping
- **Table Per Class hierarchy**
  ✔Working with XML Mapping✔Working with Annotations
- **Table per SubClass hierarchy**
  ✔Working with XML Mapping✔Working with Annotations
- **Table per concrete class (unions) hierarchy**
  ✔Working with XML Mapping✔Working with Annotations
- **Implicit Polymorphism**
- **Pros** and **Cons** of Inheritance mapping models.
- Association Mapping
  ✔AssociationMapping ✔ComponentMapping
- AssociationMapping
- **Many-to-One**
  ✔ManyToOne(XML& Annotations)
- **One-to-One**
  ✔OneToOne(XML& Annotations)

- One-to-Many
  ✔OneToManySet(XML&annotations)
  ✔OneToManyList(XML &Annotations)
  ✔OneToManyMap(XML&Annotations)
- Cascade attriutes in Hibernate
  ✔none✔all✔save-update✔delete✔orphan-delete✔all-orphan-delete
- Many-to-Many
  ✔ManyToManySet(XML&annotations)
  ✔ManyToManyList(XML &Annotations)
  ✔ManyToManyMap(XML&Annotations)
- Working with bag collection Type
- Bidirectional association mapping
- Inverse="true" or "flase"
- Component mapping
  ✔XML Mapping ✔Annotation Mapping.

PART-C

- HibernateQueryLanguage(HQL)
  ✔Advantages Over SQL ✔Writing queries
- Criteria API
  ✔Advantages ✔Disadvantages ✔Writing queries
- Detached Criteria
- Named queries
- Native SQL Queries
  Second Level Cache
  ✔Working with EH Cache
- Global Transactions management.

01-06-2016

1)Why one should go for Hibernate?

2)How does Hibernate will takes care of converting a data that is coming from a database table to the entity class object through mapping?

To convert the data that is coming from a Database Table to the Entity class object,Hibernate itself will not have intelligence,so we need to provide that information like Entity class and Mapper information to the Hibernate.

The Entity Class will looks as Below..

```
public class Customer{
                    private int         id;
                    private String      firstName;
                    private String      lastName;
                    private String      mobile;
                    private String      email;
       //Setters&Getters
                 }
```

In order to convert the data that is coming from a database table into the corresponding Entity class,we should not write the logic for reading the data and populating the data into the Entity Class. Instead of we writing it ,As it is a common requirement that every application wants to map the data from Database to the Entity class,So Hibernate should take care of it.

Hibernate is a Framework which provides bunch of classes which already contains pre-idenfied functionalities which eliminates us from writing BiolerPlateLogic.

How do we need to tell to the Hibernate that for our Class , we need the data from specific Table and from a specific coulmn to be mapped into our class attributes?

Hibernate will not takes care of accessing the data automatically and would not be able to map the data.So Programmer has to provide the metadata information to the Hibernate.

What is metadata?

Example:Whenever we want a Customer Data for the given ID=1,Instead of we trying to get the data from the database and store into the Entity class,we will go to the Hibernate asking for the Customer Data whose ID=1,Then Hibernate has to get that Data and map the Data into the Entity class.

But If Hibernate has to do That process means,It has to know the details like What is the Table for Customer Class?what Is the Coulmn for these attributes?.
As Progrmammer knows all these things ,so Programmer only has to tell this data to the Hibernate to map the data from table to the class,the programmer provided data is called as MetaData(mapping information).
Who is going to use the Mapping Information provided by programmer?
What is Hibernate Mapping file?
Hibernate should be able to read this mapping information to map the data from a Table to the Class. So we need to write this information in Hibernate understandable format,otherwise it will not allows us to facilitate the data from Database Tables to the Entity class.
So Hibernate will expect a configuration file from us in which we need to provide the information about our class and the Table from which the data has to be queried and should get populated,this configuration file is called HibernateMapping file.
What does this HibernateMapping file is all about?
HibernateMapping file is basically an XML file,This file contains the way how to populate the data from a Table to Our Entity class.
As We know that Every XML will starts with "Prologue" and it has one and only one Root Element.
The sample HibernateMapping file will looks like as below..

```
<?xml version="1.0"? encoding=UTF-8>(prologue)
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN""http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<!--Generated Jun 23,2016 7:00:05 PM by HibernateTools 3.4.0.CR1->
<hibernate-mapping>(RootElement)
----------------------------
-----------------------------
</hibernate-mapping>
```

Lets Construct one sample HibernateMapping file and we will see how it will works exactly.
This mapping file contains information about to which class ,Hibernate has to get the data from which Table.Here we need to pass the fully qualified name of the Hibernate as follows...

```
<hibernate-mapping>
<class name="com.fh.entities.Customer" table="CUSTOMER">
```

Once it get the data from the Customer Table,then Hibernate has to store this data in the attributes of Customer Class,for this we need to tell to the Hibernate whcih columns has to be populated with which attributes of Customer class?

To provide this information to the Hibernate ,we need to write one child tag within <Hibernate-mapping> which holds the information about attributes and their columns.Within Customer class,attributes are there,and into this attributes the data should be populated by querying the data from the table and by reading the data from those columns.

All the columns in a table are not same,but there should be one column which is special column and is PRIMARY KEY,it hepls in identifying Our Class.

For the attribute id in the Customer Class,hibernate has to get the data from the Customer table so we need to mention the coumn name also as `<id name="id" type="int">     <column  name="ID"/>`

This class contains multiple attributes so we need to configure all those attributes also in mapping file with the help of property tags as follows...

```
<property name="firstName"    type="java.lang.String">

<column name="FIRSTNAME" />

        </property>

        <property name="lastName" type="java.lang.String">

            <column name="LASTNAME" />

        </property>

        <property name="mobile" type="java.lang.String">

            <column name="MOBILE" />

        </property>

        <property name="email" type="java.lang.String">

            <column name="EMAIL" />

        </property>

    </class>

</hibernate-mapping>
```

So Hibernate read this mapping information and tries to get the data from the corresponding Table and populate according to the attributes of our class.

Note:As it is a special Mapping XML File that is wriiten specifc to the Hibernate standards,so we need(Not cumpulsory) to name it as "**Customer.hbm.xml**".

Now We have the Entity class and the Corresponding Mapping file and how we can access the data from corresponding table and how can we ask the Hibernate to populate the data into the Entity class is still pending to address Lets see.

To access the data,Creating Connection,Creating Statement, Executing a Query,getting the ResultSet,iterating over the ResultSet and asking the Hibernate to map that particular ResultSet into an Entity class Object is also a BoilerPlated logic.Hibernate should take care of all the above activities instead of we writing manually.

So Hibernate Should provide a class to us which eliminates us to write the above saidBoilerPlateLogic.

Lets say Hibernate provided class is EntityManager,To the Entity-Manager we need to pass the Database Details like driverClass Name,URL,UNAME,PWD.EntityManager requires the Customer Table within the Database,so it has to go to the Database and it has to look for The Customer Table To query the data from the Table whosse ID=1.

Lets say If we wanted to get the data from a Car Table within the same database also, again we have to pass the Database Details. Instead of this we need to write these database details in a separate place to read the configuration details by EntityManager if the requests are coming to the same Database.

Database configuration information should not write in Hibernate-mapping.hbm.xml, because along with the mapping information for a speciifc class, the Database Configuration information also would gets duplicated across all the maaping files. As it is a Configuration information related to the database and will be used by Hibernate so we can write it in a separate file and we can name it as "**hibernate.cfg.xml**".

What does the hibernate.cfg.xml file is all about?

It contains the Information about database configuration to allow the Hibernate to access the data within the database.

As it also an XML so it will contain one root and it lloks like as below

```
<?xml version="1.0"  encoding="UTF-8"?>
<!DOCTYPE  hibernate-configuration  PUBLIC"-//Hibernate/Hibernate
Configuration DTD   3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<!--   Database configuration     -->
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property              name="connection.password">welcome1</property>
</hibernate-configuration>
```

Now to the Entity manager we need to pass **hibernate.cfg.xml** ,hibernate-mapping.hbm.xml.

The EntityManager role will be played by **Session** in case of Hibernate, **Session** not only creates connection,in addition to it, Session will understand the information about how to map the data from Table to a Entity class based on the mapping information and it has an ability to querying the data ,mapping and converting the data into an object and directly gives the **SessionObject** to us.So first and foremost we need to create Session which is provided by Hibernate. 02-06-2016

What is Session?What is the purpose of Session?
Why do we need Session?

Session is an Object that is used for talking to the Database in accessing the data in an Object Format,It acts as a Connecton with which we can establish a connection to the Database in querying and accessing the data from the Database.

What should we pass as an input to the Session?

Basically A Session requires Databse Information,We need to pass Database as an input to the Session.Unless and untill it can not create a connection to connect to the database.That means Internally Every Session holds one connection.

Once we provided the relavant database information then Session should able to execute the query to get the data from database.To access the data from the Table,we need to pass TableName.As we want the data from the Customer table whose id=1,So we need to pass PRIMARYKEY column and the Value with which we wanted to query the data from the Table.Apart from this we need to pass the mapping information(hbm file) also.

Lets say if we wanted to query the data from two different tables(Customer,Car) then we need to Create Two session objects.Every time in creating the session we dont need to pass the Database Configuration details if the tables belongs to the same database.So we will write the configuration details in one gloabl configuraion file as hibernate.cfg.xml and lets the session to read the configuration file in creating the connection and querying the data.
Session should be able to read the configuration file and would be able to create the connection,statement,query the statement against the primary key and gets the ResultSet and maps into an object and able to return it. When we gave the table to the Session To identify the corresponding entity class into which the data has to populated then it will read the information from mapping file.So providing the class name is better than Tablename because if we provide the Table name then it gets hard coded across all the classes wihtin our application.
How many Entity classes will be created per a Table?
Per one Table within the Database most of the times we will create one Entity class only.So that we can find the table name based on the class name. So we should not pass Table name to the Session rather we will pass Entity class as input why because a change in the table name will not impact more if we pass className.
If we want to get the data from the Database,To the session we need to pass Entity class to identify the Corresponding Table.
Customer customer = (Customer) session.get(Customer.class, 1);
and we need to pass the Key Coulmn.
Can we write two mapping files with the same name?
We can not have two mapping files with the same class name,ambiguity will occurs if we have like that.The mapping file name and Class name need to be same.Per one class one mapping file is enough we can not have multile mapping files for an Entity class.
When we give the Class name and coulmn value and configuration to the session,it will goes to the configuration file and search for the relavant mapping file and gets it based on the class name

and identifies the Table.
Hibernate configuration file contains two parts one is about database information and the other one is about mapping information,once the mapping has been identified then session goes to the configuration file and creates the connection,statement,executing the query.
Do we need to create the Query?Or Hibernate itself is able to roll a query to Query the data?
Here the query should be standard whenever we are querying the data using primary key.So it shoud be done by Hibernate only.If session has to create the query,it has to know the information about database from which we are trying to get the data.So to the session we need to tell the database name becuase from database to database the queries and syntaxes will gets changed.
Lets say If we pass the database name as Oracle then session would be able to generate Oracle speciifc query.
As Session itself is responsible for understanding the semantics and syntaxes of classes of each and every database in creating the query and accessing the data.Lets say If the new database has been added into the market then session should be able to work as per the newely added database syntaxes to generate query.But session is a class that is not only responsible in generating the query,but also resonsible for understanding the mapping metadata and configuration data,creating the necessary objects to query the data from underlying databases,talking to the metadata manager and metadata mappers and etc.So If a new database added into market then session has to modified ,but modifying the session means modifying the core components of Entire Hibernate.So Session should not generate query,to genearate the Query **dialect** came into existence.
So To the dialect, Session need to pass the query,dialect is an **Interface** which will have multiple implementations like **Oracle10g-dialect** and MySql dialect and etc.Per one database one dialect will be there.So even in future if a new data came into market also,we dont need to modify the session rather its enough to change the Dialect informatio-n according to the database.dialect information will be passed through Configurationfile.
We can not create session as empty,unless and untill it has the configuration and mapping information,it would not able to work against the databases.Inorder to use the session object we need both the mapping and configuration.As we dont how to create a session,which is a complicated job because to create session object we need to pass and read the configuration and mapping metadata.

So To create the Session we need to gor **SessionFactory**(like creating a car from CarFactory) and we need to pass the Configuration file as an input to the SessionFactory.SessionFactory will takes care of reading the configuration file and mapping file and keep the meta data with it.So first we need to cretae SessionFactory with the metadata.Whenever we required the Session then SessionFactory should be able to create the session instead of going the configuraton file rather the configuration data is already preloaded in the SessionFactory and will gives the session object to us.

03-06-2016

Lets say we have a class called 'A' with some bsiness logic as follows....

```
ClassA
{
  Private   int   i:
---some    business  logic----
}
```

Lets say creating the object of 'A' seems to be difficult,so we will go for Factories.

```
Class       AFactory{
  public    StaticA(){
   //contains the logic for creating object of A  class.
     }
}
```

Factories are meant for creating the objects of other class,But Always the factories will creates hallow(empty) objects,they will not initialize/ populate any piece of information as part of the object.Whenever we created the object as A a1=Afactory.createA(),then BeanFactory will be able to create the object of a1.The value within the 'i' is zero(default). whenever we created one more onject for A as A a2=Afactory.createA() ,now also the alue of 'i' wil be zero,if we dont want the 'i' value to be zero rather we want to the "i' value to be 10,For this after creating the object of A,we need to set the values as a1.setI=10,a2.setI=10.So here Programmer has to write the logic for initializing the values across all the places immedialtely after the object has been created.

Instead of Afactory creating the object of class A as empty,if we pass the' i' value to the Afactory to initialize the object with our value. Public AFactory(int i), AFactory afactory=new AFactory(10);so when we call A a=Afactory.createA() then we dont need to set value for 'i'explicitly.So now the objects are complete one so we will not call these as factories rather these are called as Buiders.

What are Builders?What are Factories?

Factories will just creates hallow objects,But to the Builder we will pass the data with which we wanted to create the objects of our classes. now If we want the object of session,then we should not create session as Session session=new Session(); why because to create the session we need configuration and mapping meta data.To create the session by using SessionFactory then we will get session as empty. But we want the session should not be empty so we need a builder rather than a factory.So to the SessionFactory we need to pass the Configuration to create the session with pre-loaded metadata. So SessionFactory is being called as BuilderPattern rather than calling it as FactoryPattern.If we want one more session object then again we should not create the SessionFactory because the configuration information will not change frequently,it remains same untill the database is same.So its enough to create one SessionFactory per one application/one database.

Why we need SessionFactory?

1.SessionFactory acts as GlobalObject or a Registry of Information containing the configuration and mapping metadata,which can be always reusable in creating any number of sessions.
2.It acts as GlobalRegistry in having the configuration and mapping meatadat required to create session.

How to create a SessionFactory?

If we wanted to create a SessionFactory then we need the configuration information and which is being written in "hibernate.cfg.xml".

What does hibernate.cfg.xml file contains?

It contains several inputs that are required to work with Hibernate,such configuration information will be passed as an input to the Hibernate. The configuration file will looks as below..

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate
Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<!-- Database configuration -->
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</propert>
```

```
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<mapping resource="com/fh/entities/Customer.hbm.xml"/>
</hibernate-configuration>
```

why do we need to write dialect in hibernate.cfg.xml?

As it is a constant within the application untill the database got changed,so it would be relavant to write it in configuration file rather than refering it in all the places.

Once we have the configuration file,mapping file,entity class then we need to create a session,to create session we need SessionFactory,Even to the SessionFactory we need to pass configuration information for not to cretaed it as empty.

So we need to read and pass the Hibernate configuration file as input to the SessionFactory to create SessionFactory and SessionFactory will takes care of creating the connection.

But it is difficult to read the configuration file by the programmer,so programmer will creates the Configuration class by passing configuration file and ask the configurationclass to read the configuration file to create the SessionFactory.ConfigurationClass will create SessionFactory by passing Configuration file which is being provided by the user.Once SessionFactory has been created then we can get the session with the help of SessionFactory.

```
Configuration configuration = new Configuration().configure();
SessionFactory sfactory = configuration.buildSessionFactory();
Session session = sfactory.openSession();
```

now to the session we can pass the class name and the keyColoumn value to which we wanted to get the data from the database.

```
Customer customer = (Customer) session.get(Customer.class,1) ;
System.out.println("firstName : " + customer.getFirstName());
```

04-06-2016

Requirement#1

Accessing the Customer data from Database.

To work with Hibernate we need Hibernate jars,and we will place these jars in lib directory wihtin our Java project.

The following are the required jars to work with Hibernate...

Required jars

antlr-2.7.7
dom4j-1.6.1
hibernate-commons-annotations-4.0.4.Final
hibernate-core-4.3.5.Final
hibernate-jpa-2.1-api-1.0.0.Final
jandex-1.1.0.Final
javassist-3.18.1-GA
jboss-logging-3.1.3.GA
jboss-logging-annotations-1.2.0.Beta1
jboss-transaction-api_1.2_spec-1.0.0.Final
and Ojdbc6.jar

After placing all the jars in the' lib' directory we need to add these jars to the classpath by configuring buid-path option in Eclipse.

```
FirstHibernateExample
|-Src
    |-com.fh.entities
        |-Customer.java
        |-Customer.hbm.xml
    |-com.fh.test
        |-FHTest.java
|-hibernate.cfg.xml
```

First we need to create the Entity class,The Customer Entity class will looks as below...

Customer.java

```
public class Customer {
                private int id;
                private String firstName;
                private String lastName;
                private String mobile;
                private String email;
        //Setters &Getters
              }
```

Once we have the Entity class,If we wanted to access/persist the data from/into the database we need to have Mapping file,so lets create a mapping file.

We can not write any file and we can not call any file as mapping file,why because the mapping file that we are writing should be wriiten in Hibernate understandable format.As we discussed earlier hibernate mapping is an xml file ,the root-element is <hibernate-maaping> and the corresponding child element is <class name="---" table="—"> we can not write any name elements as part of our root element apart from the hibernate provided tags,Hibernate provided one DTD for mapping file,The elements and the structure of xml is defined by Hibernate people as part of hibernate-mapping file.Based on the DTD only we need to write the XML,if these XML has been written as part of the Hibernate provided DTD then to validate the contents of these xml we need to link this XML to the DTD by writing the following tag. <!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN""http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"> this tag has to be wriiten by programmer then only Hibernate would be able to inject mapping file,would be able to validate the XML file that we have provided against to the DTD.we can not remember the path of the DTD,so we will take the help of IDE to generate this mapping file.
RightClick(Entitiespackage)→new→other→hbm.xml→next→finish.
To achieve this we need to install one plug-in as follows.. Help→marketplace→Jbosstools(plug-in)(ifavailable).
If we dont have this option then we can create normal XML file and we can copy the the required tag from the any of the Hibernate mapping file.

Recommendation#1

Its better to place Hibernate mapping file for a particular Entity class along wth it under the same package.If we do like this we can find the mapping file easilly.

Recommendation#2

It is suggestable to name the mapping file as ClassName.hbm.xml

Customer.hbm.xml

Every XML Starts with Prologue. <?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN""http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd"><!-- Generated Jun 23, 2016 7:00:05 PM by Hibernate Tools 3.4.0.CR1 -->

Every XML has one and only one Root Element,and here it is..

```
<hibernate-mapping>
```

For which class we are providing this mapping,from which table we need to get the Data.

```
<class name="com.fh.entities.Customer"   table="CUSTOMER">
```

Recommendation#3

It is good to write all our Table names,columns names in capital letters,so that we can easilly distinguish which parts of the file are talking about database and which parts of the file are talking about java configuration.

For Every table there will be one primaryKey and for our every class there will be one corresponding attribute which acts as ID for the database table.

```
<id name="id" type="int"> <column name="ID"  />
```

Similary we need to declare all the attributes as above.

```
<property name="firstName"     type="java.lang.String">
          <column       name="FIRSTNAME"/>
</property>
<property name="lastName"      type="java.lang.String">
               <column   name="LASTNAME"/>
 </property>
        <property       name="mobile"    type="java.lang.String">
              <column   name="MOBILE"/>   </property>
        <property       name="email"     type="java.lang.String">
             <column    name="EMAIL"/>
        </property>
      </class>
</hibernate-mapping>
```

Now we have the Entity class and the corresponding mapping file and we need to have configuration file torepresent the environment information with which hibernate has to work.

Recommendation#4(mandatory)

It is suggestable to place hibernate configuration file under classpath directory(src).

hibernate.cfg.xml

This is also an XML file,it will also have the Hibernate provided DTD and we need to link the DTD with XML ,and it starts with prologue and It will also has one and only RootElement as follows...

Is it Mandatory to write DOCTYPE tag in configuration file?

It is mandatory to write,because Once we write the Configuration file,we need to pass this configuration file to the ConfigurationClass and it will checks wether the XML is well-formed or not? And validate it using Hibernate configurationDOCTYPE,But if we didn't pass the DOCTYPE then the ConfigurationClass will not identify the corresponding DOCTYPE between the mapping DOCTYPE and Configuration DOCTYPE to validate the XML,so its better to provide DOCTYPE.

```
<?xml version="1.0"  encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate Configuration DTD 3.0//EN""http:// hibernate.sourceforge.net/ hibernate-configuration-3.0.dtd">
```

Here the <hibernate-configuration> is the root element and within it we will have <session-factory>as child tag.Within the <session-factory> we will pass the database configuration information as follows.

```
<hibernate-configuration>
<!--  Database configuration   -->
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<mapping resource="com/fh/entities/Customer.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

Now we have Entity class,mapping file and configuration file so now we need to create the Session with SessionFactory and Configuration class to pass the configuration file.

If we want to access the data from the database we need to use **session** object incase of Hibernate.Session acts as connection with which it is going to query the data from the database.

We can not directly create the Session, so we need to take the help of SessionFatory.SessionFactory acts as a builder for creating the session because it creates session not as hallow object.
To create the SessionFactory we need to pass the Configuration but we can not pass the configuration directly to the SessionFactory.So again we need to take the help of Configuration class.

```
public class FHTest {
    public static void main(String[] args) {
Configuration configuration = new Configuration().configure();
```

Here if we didnt call configure() method, and if we just write Configuration configuration = new Configuration(); then an empty configuration class will be created.so we need to call configure() methood also,then only Configuration class goes to the classpath of our project and looks for the configuration file name called hibernate.cfg.xml,if file is there, it reads the configuration and creates the metadata,if the file is not there it will creates an empty object rather than throwing an exception.
Once we have the Configurtion then we can create SessionFactory

```
SessionFactory sfactory =configuration.buildSessionFactory();
```

Once we have the SessionFactory then we can create the session obejct as `Session session =sfactory.openSession();`
Once we have the session object,we need the data for the customer so

```
Customer customer =(Customer)session.get(Customer.class,1);
```

once we get the data we can see the data in the customer class as

```
System.out.println("firstName : " + customer.getFirstName());
```

With this we can able to get the data from the database if we have the data in the Database.
Here application is not yet terminated,so we need to close the SessionFactory then it will releaese the resources,but before closing the SeesionFactory we need to close Session.Session holds one connection if we dont close the connection then the connection will not be released which will cause resource leakage problem so we need to close both the Session and SessionFactory as follows...

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

CompleteExample:

```
FirstHibernateExample
|-Src
    |-com.fh.entities
        |-Customer.java
        |-Customer.hbm.xml
    |-com.fh.test
        |-FHTest.java
|-hibernate.cfg.xml
```

Customer.java

```
public class Customer {
                        private int id;
                        private String firstName;
                        private String lastName;
                        private String mobile;
                        private String email;
         //Setters &Getters
                      }
```

Customer.hbm.xml

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN""http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<!-Generated Jun 23,2016 7:00:05 PM by Hibernate Tools 3.4.0.CR1 ->
<hibernate-mapping>
<class name="com.fh.entities.Customer" table="CUSTOMER">
  <id name="id" type="int"> <column name="ID" />
 <generator class="assigned" /></id>
<property name="firstName" type="java.lang.String">
          <column name="FIRSTNAME" />
</property>
<property name="lastName" type="java.lang.String">
          <column name="LASTNAME" />
</property>
        <property name="mobile" type="java.lang.String">
          <column name="MOBILE" />
    </property>
        <property name="email" type="java.lang.String">
       <column name="EMAIL" /> </property>
         </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate
Configuration DTD 3.0//EN""http:// hibernate.sourceforge.net/
hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<!-- Database configuration -->
<session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:orale:thin:@localhost:1521:xe</property>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<mapping resource="com/fh/entities/Customer.hbm.xml" />
</session-factory>
</hibernate-configuration>
```

FHTest.java

```
public class FHTest {

public static void main(String[] args) {
        Configuration configuration = new Configuration().configure();
        SessionFactory sfactory = configuration.buildSessionFactory();
        Session session = sfactory.openSession();
       Customer customer = (Customer) session.get(Customer.class, 1);
System.out.println("firstName : " + customer.getFirstName());
          session.close();
          sfactory.close();

    }

}
```

Output:

If we Run the above FHTest class then we wll get the data from the DataBase.

Requirement#2

Creating/Saving a WorkShop into the DataBase.

```
SecondHibernateExample
|-Src
    |-com.sh.entities
        |-Worksop.java
        |-Workshop.hbm.xml
    |-com.sh.test
        |-FHTest.java
|-hibernate.cfg.xml
```

Here The Entity class that we have is "Workshop",It contains the attributes corresponding to the coulmns of the Table into which we wanted to store the data.

Recommendation#5

It is always recommended to make the Entity class to implements from Serializable.

Recommendation#6

It is recommended to declare the attributes of Entity classes as protected why because sometimes we might inherit one Entity class from Another class,declaring them as private will not makes them inheritable.The Workshop Entity class will looks as below by following the above recommendations..

```
public class Workshop implements   Serializable{
        protected  int    no;
        protected  String      name;
        protected  Date        estDate;
        protected  String      websiteAddress;
        protected  String      addressLine1;
        protected  String      addressLine2;
        protected  String      city;
        protected  String      state;
        protected  int         zip;
        protected  String      country;
        protected  String      primaryContactNo;
        protected  String      altContactNo;
        protected  String      email;
        protected  String      fax;
//setters&getters
}
```

Is it mandatory to implement an Entity class from Serializable?
No it is not mandatory but it is recommneded.
Is it mandatory to declare the Entity class attributes as Protected?
It is not mandatory,we can also declare the attributes as private ,If we declare the attributes of Entity class as private then Hibernate will not able to set or get the data from these attributes.If we made the attributes as private then we can not access the private varibles outside of the class.But Still Hibernate would be able to access the data from these attributes even those are being declared as protected,its due to reflection used by hibernate.Hibernate never directly tries to access the attributes of Entity class rather its going to call corresponding accessor methods(Setters&Getters).So we need to generate the setters and getters for our Entityclass.

Once we have the Entity class,next we need to create the mapping file under the same package in which The Entity class also resides.The mapping file name can be **Workshop.hbm.xml** to map the corresponding EntityClass.Within the mapping file we have <hibernate-mapping> and it contains Workshop class mapping formation.
How many classes mapping information we can write as part of the one mapping file?
We can write any number of classes mapping information in one mapping file.it eliminates writing multiple mapping files to configure mapping information for individual classes.If we write multiple classes mapping information in one mapping file(Globalmapping.hbm.xml)then it is difficult to manage thats where it is recommended to write one mapping file for one Entityclass.

Lets write **workshop.xml**,Inorde to make our file to be acts as hibernate mapping file we need to add the hibernate mapping DOCTYPE as follows..

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN""http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

Do we need to write the FullyQualifiedName of the class?
we should not write the shortname of the class,we must and should write the FullyQualifiedName of the class.
How do we need to avoid prefixing the package name to Entity Class in mapping file?

within a mapping file we can configure multiple classes as well,in such cases the package name will become will redundant across all the classes,the package name has to be modified across all the places thats why we dont need to declare the package name at class level rather we can write an element called “package’,If we declared the package as an attribute at mapping level then all the subsequent classes will be qualified for the same package so that we can avoid duplicate references of package name across all the classes.

```
<hibernate-mapping package="com.sh.entities">
```

after this we will configure the Entity class name and the Corresponding Tablename in the database.

```
<class    name="Workshop"   table="WORKSHOP">
```

within this we need to declare the attributes as follows... `<id name="no" column="WORKSHOP_NO" />` and we will declare the rest of the attributes as `<property name=”name” type=”java.lang.String”><column name=”WORKSHOP_NM”/> </property>`

Instead of writing in the above way we can write the column also as child tag `<property   name="name"   column="WORKSHOP_NM"/>`

if we wite like this we can have a scope to impose integrity constraints like NotNull=true also.

```
<property name="estDate"       column="EST_DT"/>
<property name="websiteAddress"   column="WEBSITE_ADDRESS"/>
<property name="addressLine1"     column="ADDRESS_LINE1"/>
<property name="addressLine2"     column="ADDRESS_LINE2"/>
```

Is it mandatory to provide the Column name for the attribute?

It is not required to specify the column name but the attribute name and coumn name should be same.If we have not provided the corresponding column name of the table then the Hibernate will assume the attribute name that we have wriiten itself will be taken as Column name.If we see the property addressLine1 there the attribute name and the column name both are different,insuch cases it is good to write the column name,where as for the properties like ‘city’ both the attribute name and the coulmn name are same so we dont need to configure the column name,it is called convention over configuration.

```
<property name="city"/>
<property name="state"/>
<property name="zip"/>
<property name="country"/>
```

```
<propertyname="primaryContactNo"
column="PRIMARY_CONTACT_NO"/>
    <property name="altContactNo"    column="ALT_CONTACT_NO"/>
       <property  name="email"    column="EMAIL"/>
        <property name="fax"       column="FAX"/>
   </class>
</hibernate-mapping>
```

Once we have the Entiy class,Mapping file ,next we need to write the configuration file under the classpath.it will looks as below..

```
<?xml version="1.0"  encoding="utf-8"?>
<!DOCTYPE  hibernate-configuration  PUBLIC"-//Hibernate/Hibernate
ConfigurationDTD3.0//EN""http://hibernate.sourceforge.net
/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
        <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe
</property>
<property name="connection.username">next_gen_auto_sys
</property>
<property name="connection.password">welcome1</property>
```

Is it mandatory to Configure dialect?

Configuring the dialect is optional,if we didn't configure any dialect then hibernate will do any introspection of underlying database and able to identify the dialect automatically.So dialect is mandatory in working with Hibernate,but Configuring it in configuration file is optional.

```
<!--<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>-->
     <mapping    resource="com/sh/entities/Workshop.xml"/>
      </session-factory>
</hibernate-configuration>
```

Once we have the Entity class,mapping file and configuration file next we need to create the session to store the workshop object into DB.

```
Configuration configuration = new    Configuration().configure();
SessionFactory sfactory = configuration.buildSessionFactory();
Session session = sfactory.openSession();
```

Here we haven't write any try..catch blocks,because in Hibernate All the Exceptions are runtime Exceptions,so annoying try..catch blocks need not to be written.when we created a session,we need to close it properly at the end of the method.Lets say while this logic is working if there is a Runtime exception that is being thrown then the session will not gets closed why because as it is a Runtime exception it throws to the Default JVM Hnadler, as we are not handling it.So the session and SessionFactory will not gets closed properly.Thatswhy it is often recommneded to write try and finally blocks.

Recommendation#7

It is required to write the try..and finaly blocks to handle the resources (closing Session and SessionFactory)properly.

05-06-2016 & 06-06-2016 (No class)

07-06-2016(Internal flow)

Now we need to create the Session,SessionFactory, Configuration classes by enclosing them in try…finally blocks to close the session and SessionFactory properly irrespective of the Exception.
Whenever we tried creating Configuration Class then Configuration object will gets created by reading the physical configuration file,which is there under classpath of our application.By default the Hibernate looks for the configuration file under classpath only.

Recommendation#8

Even If it is WebApplication, then also hibernate.cfg.xml should be placd under classpath only,instead of under WEB-INF,placing this file externally is of no use.
Creating configuration object means loading and placing all the configuration and maping information in the memory. once the Configuration has been created,we need to create SessionFactory as configuration.BuidSessionFactory();,even if we want to create one more SessionFactory also then again we dont need to cretae Configuration object,because Configuration object will be created only once and with that object only we can create any number of SessionFactories.

Is it recommneded to create more sessionFactories with the same Configuration object?

No it is recommended,why because even though we create multiple SessionFactories,all are equal only without any modification, because configuration is same for both the SessionFactories.

Now the Configuration Class will takes all the mapping and Configuration information and then copies into sessionFactory instance and creates the sessionFactory object and pours the metadata into it.SessionFactory will never hold the reference of configuration object and it just hold mapping and configuration.

How do we need to release a configuration object that is being created? If we release the Configuration object then SessionFactory will works or not?

When we call Session session=sfactory.openSession(),SessionFactory will reads the database configuration from hibernate.cfg.xml and creates the object of connection with setAutoCommit=false,which means whenever we want to executing a DML operation then Transaction should not gets commited automatically,unless and untill we issue an commit() or rollback().and It will passes that connection as an input to the session ad creates one session object.One session represents one active Connectio with which it is pointing to the daabase.Unless and untill setting a connection to the database,there is no use of creating a session.

If we want to store the data of the workshop then we can call session.save().when we call session.save() then Session object goes to the object and tries to get the canonical(fullyQualified name) name of the class as object.getClass().getCanonicalName() with this class name it goes to the mapping information that is there in the sessionFactory and searches for mapping file whose class name is equal to the canonical name of object that we have passed,If the relavent class mapping is not available then it will throw an exception.Where as if the relavant mapping file is available then session object creates Prepared Statement with the help of SQL Query by going to the dialect and telling about the mapping metadata and operation based on the method we called,and ask the dialect to generte one query with substitutional parameters,now the PreparedStament object will gets created with the insert query that is being generated by dialect.Once the PreparedStateement object is being created ,it passes the mertadata and pstmt object to the mapper.Now the whole object of data will be mapped into the corresponding substitutional parameters of the pstmt object.Once the PreparedStatement has been created and it substitutional parameters are substituted then the session is going to call pstmt.executeUpdate(),then the query with the mapped data will executed and the data will be saved into the database ,whenever we call transaction.commit() or transaction.rollback() and returns the value of number of rows being inserted into the database.

If we wants to store the workshop object into database,Then we will ask the session but Session never takes care of commiting /rollback a a transaction,Its jsut store the data why because if session does this then it will supports only local/Global transaction,so we need one more component called "TrandactionManager".This will go and searches in the hibernate.cfg.xml to verify the type of transaction, based on that it creates the object of transaction.One Transaction object refers to one Type of Transaction.

08-06-2016

when we call session.save(workshop) then the data will not gets commited into the database,rather its just persist the data.To commit the data, Session takes the help of Hibernate provided one **Interface** called "**Transaction**" and it will have **methods** as **commit()** and **rollback(),**and it have implementations like JdbcTransaction and JtaTransaction

Before calling the save() first we need to get the reference of Transaction object,and we need to tell to this transaction about either commiting or rollback the changes that we are doing from now onwards.So we need to create Transaction,when we call session.beginTransaction(),then session will goes to one internal class called "transactional strategy helper" and it will ask him to visit the configuration file and determine the type of transaction and ask to give the object of specified transaction. we can switch from one type of Transaction to another transaction by modifying the configuration file.

Lets try to write Test class with Transations and try..finally blocks.

```
public static void main(String[] args) {
                    Configuration configuration = null;
                    SessionFactory sfactory = null;
                    Workshop workshop = null;
                    Session session = null
try {
                     configuration = new Configuration().configure();
                     sfactory = configuration.buildSessionFactory();
                     session = sfactory.openSession();
                     Transaction transaction = session.beginTransaction();
      workshop = new Workshop();
            workshop.setNo(2);
            workshop.setName("Antony Workshop");
            workshop.setEstDate(new Date());
```

```
                              workshop.setAddressLine1("PostOffice");
                              workshop.setAddressLine2("Madhapur");
                              workshop.setCity("Hyderabad");
                              workshop.setState("TS");
                              workshop.setZip(35252);
                              workshop.setCountry("India");
                              workshop.setPrimaryContactNo("33838");
                              workshop.setAltContactNo("39394");
                              workshop.setEmail("antony@gmail.com");
                              workshop.setFax("3939382938");
                  workshop.setWebsiteAddress("www.antonyservice.com");
                             session.save(workshop);
```

If we tried checking here wether the data is inserted or not?but the will not be there as part of the workshop table because the transaction hasn't commited.so we need to depend upon the transaction logic.

```
 transaction.commit();
                             }finally{
                  if(session!=null){
                             session.close();
                                    }
                   if(sfactory!=null){
                            sfactory.close();

                             }
                         }
                     }
                 }
```

CompleteExample:

```
SecondHibernateExample
|-Src
    |-com.sh.entities
         |-Worksop.java
         |-Workshop.hbm.xml
    |-com.sh.test
         |-FHTest.java
|-hibernate.cfg.xml
|-lib(jars)
```

Workshop.java

```
public class Workshop implements Serializable {
        protected int no;
        protected String name;
        protected Date estDate;
        protected String websiteAddress;
        protected String addressLine1;
        protected String addressLine2;
        protected String city;
        protected String state;
        protected int zip;
        protected String country;
        protected String primaryContactNo;
        protected String altContactNo;
        protected String email;
        protected String fax;
//setters&getters
}
```

workshop.xml

```
<hibernate-mapping package="com.sh.entities">
<class name="Workshop" table="WORKSHOP">
<id name="no" column="WORKSHOP_NO" />
<property name="name" column="WORKSHOP_NM" />
<property name="estDate" column="EST_DT" />
<property name="websiteAddress" column="WEBSITE_ADDRESS" />
<property name="addressLine1" column="ADDRESS_LINE1" />
<property name="addressLine2" column="ADDRESS_LINE2" />
<property name="city" />
<property name="state" />
<property name="zip" />
<property name="country" />
<propertyname="primaryContactNo"
column="PRIMARY_CONTACT_NO" />
    <property name="altContactNo" column="ALT_CONTACT_NO" />
       <property name="email" column="EMAIL" />
        <property name="fax" column="FAX" />
   </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate
Configuration DTD
3.0//EN""http://hibernate.sourceforge.net/hibernate-configuration-
3.0.dtd">
```

```
<hibernate-configuration>
        <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe
</property>
<property name="connection.username">next_gen_auto_sys
</property>
<property name="connection.password">welcome1</property>
<!-- <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty> -->
      <mapping resource="com/sh/entities/Workshop.xml" />
          </session-factory>
</hibernate-configuration>
```

FHTest.java

```
Public class SHTest{
public static void main(String[] args) {
                      Configuration configuration = null;
                      SessionFactory sfactory = null;
                      Workshop workshop = null;
                      Session session = null
try {
                       configuration = new Configuration().configure();
                       sfactory = configuration.buildSessionFactory();
                       session = sfactory.openSession();
                       Transaction transaction = session.beginTransaction();
       workshop = new Workshop();
                        workshop.setNo(2);
                        workshop.setName("Antony Workshop");
                        workshop.setEstDate(new Date());
                        workshop.setAddressLine1("Post Office");
                         workshop.setAddressLine2("Madhapur");
                         workshop.setCity("Hyderabad");
                         workshop.setState("TS");
                         workshop.setZip(35252);
                         workshop.setCountry("India");
                         workshop.setPrimaryContactNo("33838");
                         workshop.setAltContactNo("39394");
                         workshop.setEmail("antony@gmail.com");
```

```
                    workshop.setFax("3939382938");
          workshop.setWebsiteAddress("www.antonyservice.com");
                    session.save(workshop);
```
if we tried checking here wether the data is inserted or not?but the will not be there as part of the workshop table because the transaction hasn't commited.so we need to depend upon the transaction logic.

```
 transaction.commit();
                    } finally {
             if (session != null) {
                              session.close();
                         }
              if (sfactory != null) {
                             sfactory.close();

                    }
                 }
              }
           }
```
Testing

SELECT*FROM WORKSHOP;

09-06-2016

SessionFactory:

How do we need to create the object of a sessionFactory?
Where do we need to exactly create the object of SessionFactory?
How many SessionFactory objects will be created within an application?

Within an application we may have multiple Entity classes,and we will perform persistency operations for managing ,accessing and storing the data on these classes. we might write various different persistency logic for performing persistency operation on multiple Entity classes.

Lets say we have Entity classes like Customer and Workshop,If we wanted to store the data in these classes then those persistency operations will be scattered in two classes like CustomerDAO and WorkshopDAO.

```
Public Class CustomerDao{
   public void saveCustomer(){}
}
Public Class WorkShopDao{
   public void saveWorkshop(){}
   public int updateWorkshop(){}
}
```

If we wanted to save the Customer object we need Session,Inorder to create session we need SessionFactory,to create SessionFactory we need Configuration.

```
  Configuration configuration = null;
                      SessionFactory sfactory = null;
                      Workshop workshop = null;
                      Session session = null
try {
//object creations
configuration = new Configuration().configure();
 sfactory = configuration.buildSessionFactory();
  session = sfactory.openSession();
}finally{
      if(session!=null){
                        session.close();
                                }
                 if(sfactory!=null){
                       sfactory.close();

                        }
                    }
```

Within the Customer we may not have only one method,Lets say we have one more method as

```
Public Class CustomerDao{
    Public void saveCustomer(){}
    Pulic customer getCustomer(int id){}
}
```

Inorder to get the customer again we need to create the object for Session because For every atomic operation that we do, we need to create one connection,If two different operations are being managed in one connection then if we rollback() then both the operations will gets rollback.

We can re-use the session which we created in working with saveCustomer() for getCustomer() also whenever both are participating in the same transaction.But here both are different operations,so both will not use the same transaction that means a session will be created for Transaction or Atomic operation.

Do we need to write the logic for creating Configuration,SessionFactory ,Session in all the methods of our class?

SessionFactory is an object that holds the metadata information about our application,The data that is loaded within the sessionFactory object will never change.So If we create multiple SessionFactories also, then all the objects will holds the same metadata(read-only).So its enough to create one SessionFactory object inorder to create Session object.

Now Even though The SessionFactory is required for both the **Customer** and **Workshop** classes,we should not create the Session -Factory Object for each class,rather someone has to create SessionFactory object and should place it at some place,so that all the classes can use the same SessionFactory rather than re-creating it.So we need to write a separate class "**HibernateSessionFactory**" which will takes care of creating the SessionFactory object and stores with it.Whenever someone ask the SessionFactory then it returns the same object. HibernateSessionFactory is an util class,which will creates the one and only one SessionFactory and gives the reference of same SessionFactory object to any number of people,if anyone is calling getSessionFactory().

whenever some one calls getSessionFactory(); then first time it will creates the SessionFactory and store with it in a variable SessionFactory sessionfactory, and it should return the same object if the user is asking SessionFactory object twice.

```
if(sfactory=null){
    Configuration configuration=new Configuration.configure();
     SessionFactory sfactory=configuration.buildSessionFactory();
```

To call this method we need to create the object of HibernateSession Factory,this is singleton class because we dont want to allow it to create multiple objects.

Lets make the HibernateSessionFactory class as Singleton as below..

```
public class HibernateSessionFactory {
        private static HibernateSessionFactory instance;
public Synchronized static HibernateSessionFactory getInstance() {
//check wether instance is null or not?
//If instance=null
//instance=new HibernateSessionFactory();
return instance;
 }
public static void closeHibernateSessionFactory() {
               if (instance != null) {
                  HibernateSessionFactory.close();} }
```

Here to make the class as singleton we made the HibernateSession –Factory also as singleton.Instead of it we can create the SessionFactory itself as singleton as follows..
Lets say we have a class called HibernateUtil and within this class we declare the variable

```
public class HibernateUtil {
//as we required one object for the application so we will make it as static.
     private static SessionFactory sessionFactory;
//here static block will gets execited only once and one SessionFactory will gets created.
static {
    Configuration configuration = new Configuration().configure();
    sessionFactory = configuration.buildSessionFactory();
   }
//First it will creates one SessionFactory and should return it.
public static SessionFactory getSessionFactory() {
       return sessionFactory;

       }
//if SessionFactory is not null then we should close the SessionFactory.
public static void closeSessionFactory() {
      if (sessionFactory != null) {
              sessionFactory.close();
          }
  }
```

we create a SessionFactory in one class and keeps the reference of it,so that anyone who wants the SessionFactory can go and get the Session-Factory from this class.
Lets try to work on one Example on SessionFactory and we will see how it will works…

```
ManageSessionFactory
|-Src
    |-com.msf.dao
    |-com.msf.entities
    |-com.msf.test
    |-com.msf.util
|-hibernate.cfg.xml
|-lib(jars)
```

Lets try to take SERVICE_TYPES table in our database and we will start working on it to discuss about SessionFactory.
Inorder to access/store the data first and foremost we need to write the Entity class.

```
public class ServiceType {
        private int serviceTypeId;
        private String serviceName;
     //setters and getters.
}
```

once we have Entity class next we need to write the Mapping file(ServiceType.hbm.xml) otherwise Hibernate can not identify which attribute has to be persisted into which column of the table.

```
<hibernate-mapping package="com.msf.entities">
          <class name="ServiceType" table="SERVICE_TYPES">
                <id name="serviceTypeId" column="SERVICE_TYPE_ID" />
                <property name="serviceName" column="SERVICE_NM" />
         </class>
</hibernate-mapping>
```

Once we have the mapping file next we need to write the configuration file(hibernate.cfg.xml) to create SessionFactorya and it will be as..

```
<hibernate-configuration>
            <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
     <property name="show_sql">true</property>
     <mapping resource="com/msf/entities/ServiceType.hbm.xml" />
            </session-factory>
</hibernate-configuration>
```

Here Hiernate takes care of creating query internally as per our requirement,so whatever the SQL Query that is used generated by Hibernate For performing a specific operation can be displayed with

the help of **<property name="show_sql">true</property>**
If we configure the above property then we can debug the code easilly,if query is not working peoperly.

Now as we have the Entity class,Configuration file and mapping file and now to write the logic for storing the data we need to have a DAO.

```
public class ServiceTypeDao {
```

with in the Dao class we will have one method by passing Entity class as parameter.

```
        public void saveServiceType(ServiceType serviceType) {
```

If this Dao wants to store the data then it requires Session.To create the session it doesnt need to create Configuration class because multiple methods requires SessionFactory.

Is SessionFactory is mutable or immutable?

SessionFactory has been made as Immutable.
For creating the session we need to write logic in one place instead of writing it in several places.

```
public class HibernateUtil {
         private static SessionFactory sessionFactory;
static {
Configuration configuration = new Configuration().configure();
sessionFactory = configuration.buildSessionFactory();

        }

public static SessionFactory getSessionFactory() {

                return sessionFactory;

        }
public static void closeSessionFactory() {

                if (sessionFactory != null) {

                        sessionFactory.close();

                }

        }

}
```

During the startup of application we need to create SessionFactory and at the endof the application we need to close the sessionFactory.

If our Dao wants the SessionFactory then it can get the reference of SessionFactory from HibernateUtil class.

```
                SessionFactory sfactory = null;
                Transaction transaction = null;
                Session session = null;
```

and we need to take one variable as

```
                 boolean flag = false;
```

and we will write the code in try..finally blocks to manage the transactions.

```
try {
//creates the SessionFactory
                 sfactory = HibernateUtil.getSessionFactory();
//creates the session
                 session = sfactory.openSession();
//creates the Transaction
                 transaction = session.beginTransaction();
//saving serviceType data(class object)
                session.save(serviceType);
```

we will make the flag as true.

```
                 flag = true;
           } finally {
                          if (transaction != null) {
                             if (flag == true) {
                                 transaction.commit();
                               }
                            else {
                                 transaction.rollback();
                                       }
                         }
                            if (session != null) {
                                    session.close();
                              }
                         }
                     }
```

Every method should not close the SessionFactory.Lets say we have one more method as

```
public ServiceType getServiceType(int serviceTypeId) {
                ServiceType serviceType = null;
```

here also we need SessionFactory and Seesion but we need Transaction because we are not performing DML operation.

```
SessionFactory sfactory = null;
Session session = null;
```
But still we need to write try..finally to close the session properly even incase of a runtime exception,
```
try {
            sfactory = HibernateUtil.getSessionFactory();
            session = sfactory.openSession();
serviceType = (ServiceType)
session.get(ServiceType.class,serviceTypeId);
} finally {
              if (session != null) {
                           session.close();
                               }
                           }
//returns serviceType object.
                        return serviceType;
                  }
        }
```
Here we dont need to close the SessionFactory rather its enough to close session only.
After Dao we need to write the main method as follows..
```
public class MSFTest {
           public static void main(String[] args) {
//creates the object of Dao
                     ServiceTypeDao dao = null;
```
and we need to write the try..finaly blocks,
```
try {
//creates the object of ServiceType class
           ServiceType serviceType = new ServiceType();
//setting the data
           serviceType.setServiceTypeId(7);
           serviceType.setServiceName("Denting");
             dao = new ServiceTypeDao();
//calling the Dao method
             dao.saveServiceType(serviceType);
             System.out.println("saved service type");
//getting the serviceType
             serviceType = dao.getServiceType(6);
System.out.println("Service Name : " + serviceType.getServiceName());}
// at the end of the main method we need to close the SessionFactory.
```

```
finally {
 HibernateUtil.closeSessionFactory();
}
```
This is How we need to work on Typical Hibernate application and managing the sessionFactory within an application.

CompleteExample:

```
ManageSessionFactory
|-Src
    |-com.msf.entities
        |-ServiceType.java
        |-ServiceType.hbm.xml
    |-com.msf.util
        |-HibernateUtil
    |-com.msf.dao
        |-ServiceTypeDao.java
    |-com.msf.test
        |-MSFTest.java
|-hibernate.cfg.xml
|-lib(jars)
```

ServiceType.java

```
public class ServiceType {
        private int serviceTypeId;
        private String serviceName;
     //setters and getters.
}
```

ServiceType.hbm.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate
Mapping DTD 3.0//EN""http://hibernate.sourceforge.net/hibernate-
mapping-3.0.dtd">
<hibernate-mapping package="com.msf.entities">
          <class name="ServiceType" table="SERVICE_TYPES">
                <id name="serviceTypeId" column="SERVICE_TYPE_ID" />
                <property name="serviceName" column="SERVICE_NM" />
         </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC"-//Hibernate/Hibernate
Configuration DTD 3.0//EN""
http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
          <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
     <property name="show_sql">true</property>
    <mapping resource="com/msf/entities/ServiceType.hbm.xml" />
            </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
Configuration configuration = new Configuration().configure();
sessionFactory = configuration.buildSessionFactory();

      }
public static SessionFactory getSessionFactory() {

             return sessionFactory;

      }
public static void closeSessionFactory() {

             if (sessionFactory != null) {

                    sessionFactory.close();

             }

      }

}
```

ServiceTypeDao.java

```
public class ServiceTypeDao {
        public void saveServiceType(ServiceType serviceType) {
                SessionFactory sfactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;

                try {
                        sfactory = HibernateUtil.getSessionFactory();
                        session = sfactory.openSession();
                        transaction = session.beginTransaction();
                        session.save(serviceType);
                        flag = true;
                } finally {
                        if (transaction != null) {
                                if (flag == true) {
                                        transaction.commit();
                                } else {
                                        transaction.rollback();
                                }
                        }
                        if (session != null) {
                                session.close();
                        }
                }
        }
public ServiceType getServiceType(int serviceTypeId) {
                ServiceType serviceType = null;
                SessionFactory sfactory = null;
                Session session = null;

                try {
                        sfactory = HibernateUtil.getSessionFactory();
                        session = sfactory.openSession();
                        serviceType = (ServiceType)
session.get(ServiceType.class,
                                        serviceTypeId);
                } finally {
                        if (session != null) {
                                session.close();
                        }
                }
                return serviceType;
        }
}
```

MSFTest.java

```
public class MSFTest {
        public static void main(String[] args) {
                ServiceTypeDao dao = null;
                try {
                        ServiceType serviceType = new ServiceType();
                        serviceType.setServiceTypeId(7);
                        serviceType.setServiceName("Denting");
                        dao = new ServiceTypeDao();
                         dao.saveServiceType(serviceType);
                         System.out.println("saved service type");
                         serviceType = dao.getServiceType(6);
System.out.println("Service Name : " + serviceType.getServiceName());
                } finally {
                         HibernateUtil.closeSessionFactory();
                }
        }
}
```

10-06-2016

BootStraping in Hibernate

(1)In how many ways we can Bootstrap the Hibernate?
(2)How do we need to create the session object from SessionFactory?
(3)What do we mean by Configuration?How does the Configuration is going to read the hibernate.cfg.xml?
(4)In how many ways we can create the SessionFactory?

We have two types of API' Ways of BootStraping Hibernate and they are

(1) Legacy BootStrap mechanism

Earlier to Hibernate 3.x,there was one set of API is there for creating the object of SessionFactory and configuring our Hibernate,this is called legacy model.

Within the legacy approach again we will have three subtypes of BootStraping mechanism to provide hibernate configuration information and they are listed as follows..

→XMLConfigurationApproach
→PropertiesApproach
→Programmatic Approach

(2)Modern BootStrap mechanism

Where as from Hibernate 4.x onwards, A modern way of creating the SessionFactory has been introduced,and the API has been re-writtened using **Service** and **ServiceRegistry** API's,and this is called moderen way of BootStraping our Hibernate.

Here also we can work with the above said subtypes of BootStraping mechanism which we have used in legacy approach ,so

total we have **six** combinations of BootStraping Hibernate.
LegacyBootStrapMechanism:
Lets try to understand the legacy API in BootStraping Hibernate.
If we wanted to work with Hibernate and If we wanted to persist the data, we need Session Object.To create the Session object first we need to create SessionFactory with all the mapping and configuration information. So for every Entity class we need to write mapping details, unless and untill an Entity class is provided with mapping information ,Hibernate can not manage to perform the persistency operations.Apart from the mapping file we need configuration file as well.In general this file name would be hibernate.cfg.xml.
How many SessionFactories can be configured in one configuration file?
What is XMLconfiguration approach in Hibernate BootStrap mechanisms?
per one database we will configured one SessionFactory only,If we have two database then we need two configuration files.If we have two configuration files then both the file names should not be same.
To distinguish between these two configuration files ,to the Configuration class we need to pass the name of the configuration file with which we wanted to create the SessionFactory.But if we place any one of these configuration files in sub packages of src,then we need to pass the fully qualified name.It is the way how we customize the configuration file by changing its name and location,and we can call this procedure as Configuration approach in legacy BootStrap mechanism by passing configuration file.
What is properties approach in Hibernate BootStrap mechanisms?
Instead of writing configuration file in BootStraping Hibernate we have an alternative way in which we will write configuration details in “properties file”,for this we need to create **properties file** with the name must be as “**hibernate.properties**” file and it **must** and **should** be placed in under **classpath**(src) only.All the information related to hibernate configuration file wll be placed as keys and values in the propertied file and it will be as follows..

```
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=next_gen_auto_sys
hibernate.connection.password=welcome1
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
hibernate.show_sql=true
```

If we see this property file here every property is being prefixed with “hibernate”,this indicates that this property file will be used by Hibernate.

If we place another set of properties also in the same properties file,To distingusih between all those properties and hibernate required properties it is required to add a prefix as” hibernate” to every key-value.

can we write the mapping information as part of the properties file?

For each Entity class we will have one mapping file,and we should not place the mapping files in properties file,otherwise its a duplication process of keys,so we should not write the mapping file as part of the “hiberanate.properties”.

How to read the configuration details from properties file?

If we wanted make the hibernate to read the configuration details from a properties file,we need to create an empty Configuration object without calling configure(),if we call configure() it looks for hibernate.cfg.xml file.

If we have both hibernate.cfg.xml and hibernate.properties and when call Configuration configuration=new Configuration() then which file will be read?

Here as we didn’t call configure() so it will not read hibernate.cfg.xml, and hibernate.properties file only will read by configuration class.

If we have both hibernate.cfg.xml and hibernate.properties and when call Configuration configuration=new Configuration().configure then which file will be read?

In this case both the hibernate.cfg.xml and hibernate.properties files will be read by the Configuration class.why because while creating the object properties file will be read and after creating the object when we call configure() then configuration file will be read,so that the properties will be overlapped with each other.

11-06-2016

In properties Configuration approach as we didnt write mapping information along with the configuration information,But to the Creaed Configuration object we need to pass mapping information manually as Configuration configuration = new Configuration();
configuration.addResource("com/lb/entities/AnnualServicePackage.hbm.xml");
now the Configuration object contains both the Configuration information and mapping resources also,so we can create the SessionFactory as usaual..we can not add mapping after creating the

SessionFactory because SessionFactory is immutable.So whatever the information with we which we wanted create the SessionFactory has to be passed to the Configuration object.

What is programmatic approach in Hibernate BootStrap mechanisms?

Instead of going for properties approach we have an alternative approach called programmatic approach.In this approach we will not create hibernate.cfg.xml and hibernate.properties for BootStraping our Hibernate.

Here First We will create Configuration object,once we created it,generally it looks for the hibernate.propeties,if the file is not there it will not throw any Exception rather its jsut create empty configuration Object and to the created Empty object we need to set the properties and mapping also manually to create SessionFactory as follows..

```
Configuration configuration = new Configuration();
configuration.setProperty("hibernate.connection.driver_class",
                                 "oracle.jdbc.driver.OracleDriver");
configuration.setProperty("hibernate.connection.url",
                                 "jdbc:oracle:thin:@localhost:1521:xe");
configuration.setProperty("hibernate.connection.username",
                                 "next_gen_auto_sys");
configuration.setProperty("hibernate.connection.password",
"welcome1");
configuration.addResource("com/lb/entities/AnnualServicePackag
e.hbm.xml");
```

As we are Bootstraping Hibernate without using any Configuration approach, so we will call this approach as Programmatic approach.

Lets try to work on these aproaches by creating SessionFactory object using HibernateUtil class and within this class we will read one after another and we will see how they are BootStraping the Hibernate.

WorkingExample:

Here Lets Take ANNUAL_SERVICE_PKGS Table For creating the Entity class which is there in “next_gen_auto_sys” database.

```
LegacyBootStrapping
|-src
   |-com.lb.entites
   |-com.lb.util
   |-com.lb.dao
   |-com.lb.test|-com.lb.common
```

#1In order to access the data from the database for the corresponding table we need an Entity class.

```
public class AnnualServicePackage {
                protected int annualServicePackageNo;
                protected String packageName;
                protected float amount;
                protected int validityInDays;
//setters and getters
}
```

The mapping file for this Entity class will be as follows..

```
<hibernate-mapping package="com.lb.entities">
<class name="AnnualServicePackage"
table="ANNUAL_SERVICE_PKGS">
<id name="annualServicePackageNo"
column="ANNUAL_PKG_SERIVCE_NO" />
      <property name="packageName" column="PACKAGE_NM" />
      <property name="amount" column="AMOUNT" />
     <property name="validityInDays" column="VALIDITY_IN_DAYS" />
         </class>
</hibernate-mapping>
```

Now we can provide the configuration approach in three ways..lets see **XMLConfiguration** approach and the file name can be anything let it be as **next-gen-auto-sys.cfg.xml**

```
<hibernate-configuration>
        <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.show_sql">true</property>
<mapping resource="com/lb/entities/AnnualServicePackage.hbm.xml"
/>
</session-factory></hibernate-configuration>
```

And after ths we need a Dao class,to perfrom persistency operation to find the AnualServicePackage Informaton for a given package number. If we wanted to get the data from the database in hibernate we need to create Session,Inorder to create Session we need SessionFactory nd it should be single per application,so we will write the logic for creating session in a separate class HibernateUtil as here we have three approaches so we will write separate class for every approach.

```
public class XmlHibernateUtil {
                private static SessionFactory sessionFactory;
static {
          Configuration configuration = new
Configuration().configure("com/lb/common/next-gen-auto-
sys.cfg.xml");
sessionFactory = configuration.buildSessionFactory();

       }
public static SessionFactory getSessionFactory() {
           return sessionFactory;

       }
public static void closeSessionFactory() {
//To close the Sessionfactory at the nd of the application
if (sessionFactory != null) {
                            sessionFactory.close();
                      }
                 }
        }
```

Now within the Dao class we need to create the session,SessionFactory and we need to get the data and we will close the session.

```
public class ServiceTypeDao {
           public AnnualServicePackage getAnnualServicePackage(int
annualServicePackageNo) {
                   AnnualServicePackage asPackage = null;
                   SessionFactory sfactory = null;
                   Session session = null;
try {
                   sfactory = XmlHibernateUtil.getSessionFactory();
                   session = sfactory.openSession();
  asPackage =
(AnnualServicePackage)session.get(AnnualServicePackage.class,
annualServicePackageNo);
} finally {
```

```
            if (session != null) {
                        session.close();
                        }
             }
        return asPackage;
            }
}
```
now we can test wether it is working or not?
```
public class LBTest {
            public static void main(String[] args) {
try {
//creating the object of Dao
        ServiceTypeDao dao = new ServiceTypeDao();
AnnualServicePackage asPackage = dao.getAnnualServicePackage(1);
System.out.println("Package Name : " + asPackage.getPackageName());
} finally {
         XmlHibernateUtil.closeSessionFactory();

                }
            }
}
```
#2

To BootStrap Hibernate using properties approach,we need to create one properties file with the name "hibernate.properties" under classpath as follows.

```
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=next_gen_auto_sys
hibernate.connection.password=welcome1
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
hibernate.show_sql=true
```
and to create sessionFactory we need to create one more Util class as
```
 public class PropsHibernateUtil {
            private static SessionFactory sessionFactory;
static {
            Configuration configuration = new Configuration();
configuration.addResource("com/lb/entities/AnnualServicePackage.hbm.xml");
          sessionFactory = configuration.buildSessionFactory();
        }
public static SessionFactory getSessionFactory() {
          return sessionFactory;}
public static void closeSessionFactory() {
                    if (sessionFactory != null) {
                               sessionFactory.close();
                              }
                      }
        }
```
Now for this approach the Dao will looks as below ..
```
public class ServiceTypeDao {
           public AnnualServicePackage getAnnualServicePackage(int
annualServicePackageNo) {
                  AnnualServicePackage asPackage = null;
                  SessionFactory sfactory = null;
                  Session session = null;
try {
                  sfactory = PropsHibernateUtil.getSessionFactory();
                  session = sfactory.openSession();
  asPackage =
(AnnualServicePackage)session.get(AnnualServicePackage.class,
annualServicePackageNo);
} finally {
            if (session != null) {
                         session.close();
                          }
              }
       return asPackage;
           }
}
```
And the test class will looks as below..
```
 public class LBTest {
            public static void main(String[] args) {
try {
//creating the object of Dao
       ServiceTypeDao dao = new ServiceTypeDao();
AnnualServicePackage asPackage = dao.getAnnualServicePackage(1);
System.out.println("Package Name : " + asPackage.getPackageName());
} finally {
        PropsHibernateUtil.closeSessionFactory();      }
         }}
```

3#Now for the programatic approach we need to create one more util class as "ManualHibernateUtil" and it looks as below..

```
public class ManualHibernateUtil {
        private static SessionFactory sessionFactory;
static {
          Configuration configuration = new Configuration();
configuration.setProperty("hibernate.connection.driver_class",
 "oracle.jdbc.driver.OracleDriver");
configuration.setProperty("hibernate.connection.url",
"jdbc:oracle:thin:@localhost:1521:xe");
configuration.setProperty("hibernate.connection.username",
"next_gen_auto_sys");
configuration.setProperty("hibernate.connection.password",
"welcome1");
configuration.addResource("com/lb/entities/AnnualServicePackage.hb
m.xml");
sessionFactory = configuration.buildSessionFactory();
}
public static SessionFactory getSessionFactory() {
        return sessionFactory;
        }
public static void closeSessionFactory() {
            if (sessionFactory != null) {
                      sessionFactory.close();
                }
            }
}
```

and the Dao class will looks like as below..

```
public class ServiceTypeDao {
          public AnnualServicePackage getAnnualServicePackage(int
annualServicePackageNo) {
                    AnnualServicePackage asPackage = null;
                    SessionFactory sfactory = null;
                    Session session = null;
try {
                    sfactory = ManualHibernateUtil.getSessionFactory();
                    session = sfactory.openSession();
  asPackage =
(AnnualServicePackage)session.get(AnnualServicePackage.class,
annualServicePackageNo);
} finally {
            if (session != null) {
                          session.close();
                           }
             }
       return asPackage;
            }
}
```

And the test class will looks as below..

```
 public class LBTest {
            public static void main(String[] args) {
try {
//creating the object of Dao
       ServiceTypeDao dao = new ServiceTypeDao();
AnnualServicePackage asPackage = dao.getAnnualServicePackage(1);
System.out.println("Package Name : " + asPackage.getPackageName());
} finally {
        ManualHibernateUtil.closeSessionFactory();   }
         }
}
```

CompleteExample:

```
LegacyBootStrapping
|-src
    |-com.lb.entites
        |- AnnualServicePackage.java
        |- AnnualServicePackage.hbm
    |-com.lb.util
        |- ManualHibernateUtil.java
        |- PropsHibernateUtil.java
        |- XmlHibernateUtil.java
    |-com.lb.dao
        |- ServiceTypeDao.java
    |-com.lb.test
        |-LBTest.java
    |-com.lb.common
        |- next-gen-auto-sys.cfg
|-hibernate.properties
```

AnnualServicePackage.java

```
        public class AnnualServicePackage {
                 protected int annualServicePackageNo;
                 protected String packageName;
                 protected float amount;
                 protected int validityInDays;
//setters and getters
}
```

AnnualServicePackage.hbm

```
<hibernate-mapping package="com.lb.entities">
<class name="AnnualServicePackage"
table="ANNUAL_SERVICE_PKGS">
<id name="annualServicePackageNo"
column="ANNUAL_PKG_SERIVCE_NO" />
      <property name="packageName" column="PACKAGE_NM" />
      <property name="amount" column="AMOUNT" />
     <property name="validityInDays" column="VALIDITY_IN_DAYS" />
         </class>
</hibernate-mapping>
```

|-com.lb.util

ManualHibernateUtil.java

```
public class ManualHibernateUtil {
         private static SessionFactory sessionFactory;

static {
           Configuration configuration = new Configuration();
configuration.setProperty("hibernate.connection.driver_class",
 "oracle.jdbc.driver.OracleDriver");
configuration.setProperty("hibernate.connection.url",
"jdbc:oracle:thin:@localhost:1521:xe");
configuration.setProperty("hibernate.connection.username",
"next_gen_auto_sys");
configuration.setProperty("hibernate.connection.password",
"welcome1");
configuration.addResource("com/lb/entities/AnnualServicePackage.hb
m.xml");
sessionFactory = configuration.buildSessionFactory();
}
public static SessionFactory getSessionFactory() {
        return sessionFactory;
        }
public static void closeSessionFactory() {
            if (sessionFactory != null) {
                        sessionFactory.close();

                  }
            }
}
```

PropsHibernateUtil.java

```
public class PropsHibernateUtil {
           private static SessionFactory sessionFactory;
static {
           Configuration configuration = new Configuration();
configuration.addResource("com/lb/entities/AnnualServicePackage.hb
m.xml");
         sessionFactory = configuration.buildSessionFactory();
        }
public static SessionFactory getSessionFactory() {
          return sessionFactory;}

public static void closeSessionFactory() {
                    if (sessionFactory != null) {
                              sessionFactory.close();
                              }
                    }
        }
```

XmlHibernateUtil.java

```
public class XmlHibernateUtil {
               private static SessionFactory sessionFactory;
static {
          Configuration configuration = new
Configuration().configure("com/lb/common/next-gen-auto-
sys.cfg.xml");
sessionFactory = configuration.buildSessionFactory();

        }
public static SessionFactory getSessionFactory() {
           return sessionFactory;

        }
public static void closeSessionFactory() {
//To close the Sessionfactory at the nd of the application
if (sessionFactory != null) {
```

```
                        sessionFactory.close();
                  }
              }
       }
```

ServiceTypeDao.java

```
public class ServiceTypeDao {
          public AnnualServicePackage getAnnualServicePackage(int
annualServicePackageNo) {
                 AnnualServicePackage asPackage = null;
                 SessionFactory sfactory = null;
                 Session session = null;
try {
                 sfactory = ManualHibernateUtil.getSessionFactory();
                 session = sfactory.openSession();
  asPackage =
(AnnualServicePackage)session.get(AnnualServicePackage.class,
annualServicePackageNo);
} finally {
           if (session != null) {
                        session.close();
                        }
           }
      return asPackage;
           }
}
```

LBTest.java

```
public class LBTest {
           public static void main(String[] args) {
try {
//creating the object of Dao
      ServiceTypeDao dao = new ServiceTypeDao();
AnnualServicePackage asPackage = dao.getAnnualServicePackage(1);
System.out.println("Package Name : " + asPackage.getPackageName());
} finally {
       ManualHibernateUtil.closeSessionFactory();   }
        }
}
```

hibernate.properties

```
hibernate.connection.driver_class=oracle.jdbc.driver.OracleDriver
hibernate.connection.url=jdbc:oracle:thin:@localhost:1521:xe
hibernate.connection.username=next_gen_auto_sys
hibernate.connection.password=welcome1
hibernate.dialect=org.hibernate.dialect.Oracle10gDialect
hibernate.show_sql=true
```

next-gen-auto-sys.cfg

```
<hibernate-configuration>
        <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.show_sql">true</property>
<mapping resource="com/lb/entities/AnnualServicePackage.hbm.xml"
/>
      </session-factory>
</hibernate-configuration>
```

Why Hibernate has provided three wasy of Bootstrapping?
When to use XMLConfiguration Approach?

It is mostly used Approach in configuring/Bootstrapping a Hibernate.

Advantages:

[1].We can validate our Configuration Information,if we go for properties file we can not have validations support.

[2].It supports to work with multiple database,we can create more than one configuration files to create multiple SessionFactories.

[3].we can avoid manually configuration of mapping file.

Disadvantages:

[1].It is complicated to write and,it is mandatory to write DOCTYPE.

When to use PropertiesConfiguration Approach?
Advantages:
[1].Easy to write,No IDE support is required.
[2].Its Simple way of performing UnitTesting during development.
[3].Easy to write Junit test cases.
Disadvantage:
[1].No validation Support,so we can use it always.
[2].As there is no means of providing mapping information,those has to be managed manually due to which we need to hotcode the references of mapping files.
When to use ProgrammaticConfiguration Approach?
In General it is used in Integration.
Advantages
[1].we can read the configuration from external/proprietary configuration file.
[2].Writing configuration in external file In which the properties are encrypted ones, add more security.
Disadvantage:
[1].we need to program for Bootstrapping.

12-06-2016(No Class)& 13-06-2016

Modern BootStrapping(Service&ServiceRegistry)
Whenever we call Session session=sFactory.openSession() then Session -Factory immediately goes to the hibernate.cfg.xml and reads the database confiuration like driver_class-name,url,uname and password and SessionFactory takes care of creating the Connection, using that Connection the SessionFactory creates the Session Object.

Lets Say If we deploy our application in ApplicationSerer like WebLogic, then to create the connection we need to go for the datasource which is there in ApplicationServer for getting the connection from ConnectionPool that means here we should not use database details like driver_class_name,url,uname,password.It doesnt mean that Hibernate will not works with JavaEE environment But the problem here is SessionFactory can not switch between JavaEE and Non-JavaEE Environment.

Wether we are working with JavaEE environment or Non-JavaEE environment,depends on this,SessionFactory has to take the help of one more componenst(Helper Classes) to bring the connection from a specific Environment.So Hibernate takes the help of these helper classes for getting the additional functionalities that are plugged-in into Hibernate,these components are called "**Services**."

we will have one service called ConnectionProvider(I),whenever SessionFactory requires a Connection then SF will goes to the ConnectionProvider.ConnectionProvider will have multiple implementations like "JdbcConnectionProvider,JavaEE Connection -Provider "CustomConnectionPoolProvider" and etc..and these implementations are called as ServiceImplementors.

So now whenever we call Session session=sFactory.openSession() then SessionFactory immediately goes to the hibernate.cfg.xml and checks for user provided database configuration details and JNDI Name of the datasource,if the user has provided any JNDI Name of the datasource,it will quickly goes to the ConnectionProvider and then ask one of its Implementation like "DatasourceConnectionProvider". ConnectionProvider(I) is an Interface and it has multiple methods,two of its methods are getConnection() and closeConnection().So the DatasourceConnectionProvider" goes to the ApplicationServer in which it has been deployed and gets the datasource with that JNDI name and calls datasource.getConnection() and gives the connection itself directly to the SessionFactory.That means in JavaEE environement we have support for ConnectionPooling ,But not in the NonJavaEE Environment.

When it comes to Hibernate we have support for NonJavaEE environment ConnectionPooling implicitly.But it is not recomended to use Hibernate provided implicit connectionPooling support in production environment.In real application Environment we have to integrate with third party libraries when we are working in Non -JavaEE Environment.If it is JavaEE Environment we can take the benefit of J2EE provided ConnectionPooling support.
some of the third part connection pooling libraries are c3po, proxool ,dbcp and etc.So within the Hibernate.cfg.xml we will configure these libraries to provide the information to the Third party vendors.

Hibernate has support for the" services" from initially,but customization of connectionPooling will not be there in hibernate from 3,x,and this support has been added for adding/removing the services in the Hibernate 4.x.

Every service that is coming from Hibernate will implements from Service Interface,in this way Multiple services implementaors will be there.Here these services are dependant on each other and All the services has to be maintained in a place like container and In case Of Hibernate **ServiceRegistry** will plays the Container role which contains different service roles.

ServiceRegistry contains multiple service types,as we have multiple services which are dependant one on another services to manage the dependancies between these set of servics, we also have types in ServiceRegistry like StandardServiceRegistry and BootStrap -ServiceRegistry and etc.One ServiceRegistry will dependant on other.

14-06-2016

Hibernate 4.x has been enriched with array of Interface like Startable, ConfigurationService,Stoppable to manage Customizaions of the services.

If we want to take care of managing the life cycle of our service then the service class that we are writing should be implemented from Startable, ConfigurationService,Stoppable.

Who will call our Services?

ServiceRegistry will call our Services to start the service and close the service if our service class implemented from Startable,Stoppable.

How to pass configuration information to the Service?

Our service Requires some Configuration information and we can pass this information(hibernate.cfg.xml) as map to the service implementor class ,if we implement our class from Configurable interface.

How one service can get the another Service?

Inorder to get the one service for another service then injection will not be possible and supported by ServiceRegistry,the Service has to perform lookup or using AwareInterface.

Every Service will be binded with a speciifc ServiceRegistry through the ServiceBinding.ServieBinding maintains the metadata of the services.

Lets say we have Service like CustomConnectionProvider and it can be implementable from ConnectionProvider, Startable, ConfigurationService, and Stoppable,we need to override the methods like getConnection(), closeConnection().

```
public class CustomConnectionPoolProvider implements
ConnectionProvider,ConfigurationService, Startable, Stoppable {
@Override
        public Connection getConnection() throws SQLException {
     return con;
   }
@Override
  public void closeConnection(Connection con) throws SQLException {
con.close();
     }
```

whenever we call sFactory.openSession() then the SessionFactory should goes to StandardServiceRegistry and ask for a service(CustomConnectionProvider) of the specfic role. StandardServiceRegistry will checks for the corresponding Registered ServiceImplementor for the Specific Role,and picks up the Class and gives it to the SessionFactory.Now Session- Factory will goes to our class and call getConnection(),so that the connection will be returned to the SessionFactory,with that object SessionFactory can create the Session object.

As we have create one service(CustomConnectionprovider) and Now we need to register this service to StandardServiceRegistry.
StandardServiceRegistry.addService(ConnectionProvider.class,new CustomConnectionPoolProvider());
If we do this then the service will be created immediately,but we dont want this service to be available immediately and we wanted it to be available whenever hibernate required that service.

So hibernate has provider ServiceInitiator and inside the Initiator we can write the logic for Service.We will add the Standard ServiceRegistry to Initiator.Whenever Hibernate required the Service then it goes to StandardServiceRegistry and ask for the Service with speciifc role,then StandardServiceRegistry returns Initiator.Hibernate will checks wether StandardServiceRegistry returns a service object or Initiator object.If it is Initiator object Hibernate will ask it to give the service,where as if it is Service object,it starts counsuming it directly.

If we have hibernate.cfg.xml then to create the session,first we will create Configuration and then we will create SessionFactory as
Configuration configuration = new Configuration().configure();
sessionFactory configuration.buildSessionFactory();
If we do this then the SessionFactory will gets created by implicitly registering with the Services and ServicesRegistry,in this techinque we will not have support for Customization.Thatswhy Hibernate has depricated configuration.buildSessionFactory()
Why does hibernated removed configuration.buildSessionFactory() ?
In favour of configuring/customizing of creating ServiceRegistry Hibernate has depricated it.)

If we wants to create the SessionFactory then after the configuration information has been loaded into the Configuration object, we have to create the StandardServiceRegistry,and to the ServiceRegistry we need to pass the configuration,to let it know type of Connection,transaction and to know the environment related information to configure and manage the Services.

We should not permitted to create the ServiceRegistry as empty,rather we need to create the ServiceRegistry with Configuration,If we want 20 ServiceRegistry instances also then we can create but it is required to read the configration 20 times,so we need to take the help of Builder.So we will pass configuration object to the Builder instead of ServiceRegistry.Builder will creates ServiceRegistry i.e Every ServiceRegistry will asoociated with a Builder.

```
Configuration configuration = new Configuration().configure();
StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
```

now we can create any number of ServiceRegistry objects…and to the builder we need to add the Configuration as follows..

```
builder.applySettings(configuration.getProperties());
```

Now the Builder will know how to create ServiceRegistry with the services.

Before registering the Registry to the SessionFactory we can add customizations like add/modify services to registry.

```
builder.addService(ConnectionProvider.class,new
CustomConnectionPoolProvider());
StandardServiceRegistry registry = builder.build();
```

This StandardServiceRegistry is required for SessionFactory and Configuration can crete the SessionFactory,so we need to tell to the Configuration to SessionFactory by passing registry as reference.

```
sessionFactory = configuration.buildSessionFactory(registry);
```

Note:

Working Example:

```
ModernBootStrapping
|-src
   |-com.mb.entities
   |-com.mb.util
   |-com.mb.test
   |-com.mb.service
   |-com.mb.dao
|-hibernate.cfg.xml
```

Here Lets Take ANNUAL_SERVICE_PKGS Table For creating the Entity class which is there in "next_gen_auto_sys" database.

In order to access the data from the database for the corresponding table we need an Entity class.

```
public class AnnualServicePackage {
            protected int annualServicePackageNo;
            protected String packageName;
```

```
                protected float amount;
                protected int validityInDays;
//setters and getters
}
```
The mapping file for this Entity class will be as follows..
```
<hibernate-mapping package="com.lb.entities">
<class name="AnnualServicePackage"
table="ANNUAL_SERVICE_PKGS">
<id name="annualServicePackageNo"
column="ANNUAL_PKG_SERIVCE_NO" />
      <property name="packageName" column="PACKAGE_NM" />
      <property name="amount" column="AMOUNT" />
     <property name="validityInDays" column="VALIDITY_IN_DAYS" />
         </class>
</hibernate-mapping>
```
Now we can provide the configuration approach in three ways..lets see **XMLConfiguration** approach and the file name can be anything let it be as **hibernate.cfg.xml**
```
<hibernate-configuration>
        <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.show_sql">true</property>
<mapping
resource="com/mb/entities/AnnualServicePackage.hbm.xml" />
</session-factory></hibernate-configuration>
```
And after ths we need a Dao class,to perfrom persistency operation to find the AnualServicePackage Informaton for a given package number. If we wanted to get the data from the database in hibernate we need to create Session.

Now within the Dao class we need to create the session,SessionFactory and we need to get the data and we will close the session.

```
public class ServiceTypeDao {
          public AnnualServicePackage getAnnualServicePackage(int
annualServicePackageNo) {
                 AnnualServicePackage asPackage = null;
                 SessionFactory sfactory = null;
                 Session session = null;
try {
                 sfactory = XmlHibernateUtil.getSessionFactory();
                 session = sfactory.openSession();
  asPackage =
(AnnualServicePackage)session.get(AnnualServicePackage.class,
annualServicePackageNo);
} finally {
           if (session != null) {
                      session.close();
                      }
           }
      return asPackage;
           }
}
```

If the Dao requires SessionFactory then we need to have one more class as HibernateUtil to create SessionFactory.

```
public class HibernateUtil {
            private static SessionFactory sessionFactory;

static {
Configuration configuration = new Configuration().configure();
```

Once we have created the configuration object we need to create ServiceRegistry and we need add our services and with that registry as reference we can create SessionFactory.

```
StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
//all the properties in the configuration file will be applied.
builder.applySettings(configuration.getProperties());
builder.addService(ConnectionProvider.class,new
CustomConnectionPoolProvider());
StandardServiceRegistry registry = builder.build();
//while creating the sessionFactory we need to pass registry.
sessionFactory = configuration.buildSessionFactory(registry);
```

```
        }
public static SessionFactory getSessionFactory() {
               return sessionFactory;
        }
public static void closeSessionFactory() {
               if (sessionFactory != null) {
                       sessionFactory.close();
                  }
             }
}
```

Now we can see wether it is working or not?

```
public class MBTest {
             public static void main(String[] args) {
                  ServiceTypeDao dao = new ServiceTypeDao();
AnnualServicePackage asPackage = dao.getAnnualServicePackage(1);
System.out.println("Package  : " + asPackage.getPackageName());
        }
}
```

<u>How to Register our Own Registry in the ServiceRegistry.</u>

com.mb.service

|-CustomConnectionPoolProvider

If our Class has to acts as Service Class means,we need to categorize the service of this type,then only Hibernate can use it.Our class should implements from one of the ServiceRole Interface.

```
public class CustomConnectionPoolProvider implements
ConnectionProvider,ConfigurationService, Startable, Stoppable {
@Override
          public void closeConnection(Connection con) throws
SQLException {
           System.out.println("connection closed");
              con.close();
        }
@Override
public Connection getConnection() throws SQLException {
//Sopln is used to know wether our service is calling or not
 System.out.println("connection created");
try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
```

```
                } catch (ClassNotFoundException e) {
                          e.printStackTrace();

                }
Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"next_gen_auto_sys","welcome1");
                    con.setAutoCommit(false);
            return con;
   }
}
```

After creating the Registry we need to add this service to the Builder in the HibernateUtil class as follows..

```
builder.addService(ConnectionProvider.class,new
CustomConnectionPoolProvider());
```

whenever we call sFactory.openSession(),then it goes to Registry and Registry will give our Service and it call getConnection() on our class. Thatmeans here we are customizing the behaviour of Hibernate.

CompleteExample:

```
ModernBootStrapping
|-src
   |-com.mb.entites
      |- AnnualServicePackage.java
      |- AnnualServicePackage.hbm
   |-com.mb.util
      |- HibernateUtil.java
   |-com.mb.dao
      |- ServiceTypeDao.java
   |-com.mb.service
      |- CustomConnectionPoolProvider.java
   |-com.mb.test
      |-MBTest.java
|-hibernate.cfg.xml
```

AnnualServicePackage.java

```
      public class AnnualServicePackage {
              protected int annualServicePackageNo;
              protected String packageName;
              protected float amount;
              protected int validityInDays;
//setters and getters
}
```

AnnualServicePackage.hbm

```
<hibernate-mapping package="com.lb.entities">
<class name="AnnualServicePackage"
table="ANNUAL_SERVICE_PKGS">
<id name="annualServicePackageNo"
column="ANNUAL_PKG_SERIVCE_NO" />
      <property name="packageName" column="PACKAGE_NM" />
      <property name="amount" column="AMOUNT" />
     <property name="validityInDays" column="VALIDITY_IN_DAYS" />
         </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.show_sql">true</property>
<mapping
resource="com/mb/entities/AnnualServicePackage.hbm.xml" />
     </session-factory>
</hibernate-configuration>
```

CustomConnectionPoolProvider.java

```
public class CustomConnectionPoolProvider implements
ConnectionProvider,ConfigurationService, Startable, Stoppable {

@Override
          public boolean isUnwrappableAs(Class arg0) {
                   // TODO Auto-generated method stub
             return false;
    }
```

```
@Override
         public <T> T unwrap(Class<T> arg0) {
                        // TODO Auto-generated method stub

               return null;
 }
@Override
         public void stop() {
                   // TODO Auto-generated method stub

}
@Override
         public void start() {
                 // TODO Auto-generated method stub

}
@Override
            public <T> T cast(Class<T> arg0, Object arg1) {
                  // TODO Auto-generated method stub
            return null;
}
@Override
          public <T> T getSetting(String arg0, Class<T> arg1, T arg2) {
               // TODO Auto-generated method stub
            return null;
 }
@Override
       public <T> T getSetting(String arg0, Converter<T> arg1, T arg2) {
             // TODO Auto-generated method stub
             return null;
}
@Override
      public <T> T getSetting(String arg0, Converter<T> arg1) {
          // TODO Auto-generated method stub
             return null;
}
@Override
       public Map getSettings() {
           // TODO Auto-generated method stub
            return null;
}
```

```
@Override
    public void closeConnection(Connection con) throws SQLException {
            System.out.println("connection closed");
                con.close();
}
@Override
    public Connection getConnection() throws SQLException {
           System.out.println("connection created");
                    try {
Class.forName("oracle.jdbc.driver.OracleDriver");
                    } catch (ClassNotFoundException e) {
                    e.printStackTrace();
         }
Connection con = DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe",
"next_gen_auto_sys","welcome1");
                  con.setAutoCommit(false);
                   return con;
      }
@Override
         public boolean supportsAggressiveRelease() {
             // TODO Auto-generated method stub
                    return false;
             }}
```

HibernateUtil.java

```
public class HibernateUtil {
              private static SessionFactory sessionFactory;

static {
Configuration configuration = new Configuration().configure();
StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
             builder.applySettings(configuration.getProperties());
             builder.addService(ConnectionProvider.class,new
CustomConnectionPoolProvider());
StandardServiceRegistry registry = builder.build();
sessionFactory = configuration.buildSessionFactory(registry);
}
public static SessionFactory getSessionFactory() {
      return sessionFactory;
}
public static void closeSessionFactory() {
              if (sessionFactory != null) {
                          sessionFactory.close();
                      }
               }
    }
```

ServiceTypeDao.java

```
public class ServiceTypeDao {
            public AnnualServicePackage getAnnualServicePackage(int
annualServicePackageNo) {
                    AnnualServicePackage asPackage = null;
                    SessionFactory sfactory = null;
                    Session session = null;
try {
                    sfactory = ManualHibernateUtil.getSessionFactory();
                    session = sfactory.openSession();
  asPackage =
(AnnualServicePackage)session.get(AnnualServicePackage.class,
annualServicePackageNo);
} finally {
             if (session != null) {
                            session.close();
                            }
             }
        return asPackage;
            }
}
```

MBTest.java

```
public class LBTest {
             public static void main(String[] args) {
try {
//creating the object of Dao
        ServiceTypeDao dao = new ServiceTypeDao();
AnnualServicePackage asPackage = dao.getAnnualServicePackage(1);
System.out.println("Package Name : " + asPackage.getPackageName());
} finally {
        HibernateUtil.closeSessionFactory();        }
          }
}
```

15-06-2016

Get Vs **Load**

(1)when to use get?(2)when to use load?
(3)What is the difference between get() and load()?

Hibernate allows us to persist the data into the database and access the data from the Database with the help of Session Object.
If we wanted to access one record of data from the Database like ServicePackage information for the given servicePackage number, It is required to pass Entity class and and the corresponding primary key value(ID) to the Session by calling one of the methods.Thatswhy to fetch one record of data from the database into Entity class object,Hibernate has provided two methods and are get() and load().Here Both get() and load() will retunrs one object of data,But there is a difference between these two methods,before know about it lets see the rules in writing an Entity class.

Rules in writing Entity class:

[1]It is always recommended to write Entity class attributes as '**protected**' which helps us in achieving inheritance.Each and every attribute should has accessor methods.

[2]It is highly recommended to implement our Entity class from **Serializable** Interface.

[3]It is good to provide **Default Constructor** as part of the Entity class.

[4]Entity class can **not** be **abstract**.

[5] It is suggested to have our Entity classes to be **Non final** to perform optimization.

[6].It is recommended to **override equals()** and **hashcode()** to ensure identity of our object and database record as same.

These are some of the rules that we need to follow while writing Entity class in the Hibernate.
In Hibernate we can query the object of data from the database in two ways i.e either session.get() or session.load().

→ 😈Whenever we use session.get(),then the session class object will takes our Entity class as input,quickly goes to the SessionFactory and mapping meta –data and searches for Entity class related mapping information based on the canonical name of the class.Once it identifed that mapping then the Session object will creates a connection to the databaseand creates a statement and then queries the data by talking to the dialect.

Once the query has been executed,the session object will populate the Result-set object into Entity class object(by canonical name)so it immediately creates the object of Entity class using defaultconstructor. If we have provided a parameterizable constructor,in such cases the default constructor will not be available,so that the get() or load() can not instanitiate the object, because sometimes it is required for us to create the object of Entity class by Hibernate rather than we creating it manually.That means Session will access the data from the database directly whenever we call session.get().It is eager loading.

→ 😈 Whenever we call session.load(EntityClass.class, 1),then the load method immediately goes to the SessionFactory and tries to identify wether the mapping metadata for this class is available or not?If the Entity class is being mapped and if it is existed with the corresponding table information then it will not goes to the database to fetch the data rather it creates one proxy class object and then returns the proxy class object with empty data. That means load never access the data from the database directly whenever we call session.load().It is lazy loading.

why the does load() will not return the data by populating into Entity class object?
Why the load() is Creating proxy class object?
what do we mean by proxy?
Can we access the data using proxy?

16-06-2016

Lets say we have two classes Customer and Address as follows..

```
public class Customer {
            protected int customerNo;
            protected String firstName;
            protected String lastName;
            protected String email;
            protected String mobile;
//setters and getters
}
public class Address {
    protected String addressLine1;
    protected String addressLine2;
    protected String city;
    protected String state;
    protected int zip;
    protected String country;
```

```
        protected int Address_ID;
        protected int customerNo;(FK in Many side)
         //setters and getters.
}
```

one address belongs to One Customer,A Customer may have multiple addresses(One-To-Many),If we wanted to store the data of Address class,to persist the data into the Address Table we need to write the mapping file and we will create the object of Address as...
Address address=new Address();
address.setId(2)and similary we will set all the attributes,after that if we call session.save(address) then the data will not be persisted.because the class Address only contains the coulmns related to address but the primarykey column represented relational coulmn data is not being contained within the address Table.Thatmeans whenever we are trying to store the data then not only the fileds of address and but also the relationship coulns has to be persisted.So the Customer number also has to be associated.

So in Address class we should declare Customer as an attribute.

```
public class Address {
        protected String addressLine1;
        protected String addressLine2;
        protected String city;
        protected String state;
        protected int zip;
        protected String country;
        protected int Address_ID;
        protected int customerNo(PK);
private Customer customer;
         //setters and getters.
}
```

So that whenever we create an address,along with the address fileds we need to set the customer for which this address is being associated.

So here we dont need to create new Customer,its enough to get the one of the existed customer.

```
        Customer customer=session.get(Customer.class,1)
```

→now whenever we call session.save(address) then the hibernate will save the address associated with customer with the customer object that we have set.

If we wanted to get one customer from the database,then we need to call session.get(Customer.class,1).

Where we wil use get() or load()?
When we wanted to have an association between a child and parent,where we wanted to get the parent to be associated with the child,while persisting the child we can make use of get() or load().
where we will use load()?
As we are not using the data of Customer entity class,it is not required to get the data from the database,rather if we use load() instead of get() then the data will not be fetched from the database and a dummy customer object will be created with PK column value being populated as 1,so that we can easilly establish the association and we can persist it.Thatmeans when we wanted to have an entity class object,but we may not use the data, in such cases its better to use load() instead of get().

→ 😈Whenever we call session.load(EntityClass.class, 1) then it will not creates Entity class object,rather internally it creates one proxy class which extends from actual Entityclass object by populating the primary key coulmn value.So Thatmeans whenever we call session.load() then it will not creates Entity class object,rather on fly at runtime inside the JVM internally session object creates one more proxy class as EntityClass$proxy extends EntityClass like Customer$Proxy extends Customer with the help one library called **javaAssist** library.It is runtime bytecode generation library.
→Customer$Proxy contain primary key coulmn value and it contain methods of Customer which are being overriden here also.So now if we call customer.getMobile() then the Customer$Proxy class getMobile() will be called and this method will goes to the database and queries the Customer Table with primary key and returns the mobile whole customer of that primary key to us.
Note:Accessor methods will get the data from the database if they are not primarykey columns attributes of Entity class only.
A Proxy class not only contain the primary key value rather it contains the actual class objects aslo,so whenever we call first accessor methods then it will fetch all the data from the database,if we call second time then it will checks wether the data is loade or not?if it s loaded then it will return it,otherwise it will fetch from the database.
→**So** whenever call session.load(Customer.class,1) then the the session.load() will creates proxy class object which exactly resembles as actual class because the proxy willl extends from a actual class. Where the proxy contains the primary key value with which we wanted to fetch the data and also the original class object data will be there as part of proxy and it overrides all the getter&setters methods.

😟If the corresponding primary key value doesn't exist in the database ,then it returns null,if we tried fetching the data by calling session.get(EntityClass.class,3);
😄If the corresponding primary key value doesn't exist in the database and if we tried fetching the data by calling session.load (EntityClass.class,3);then it will not return null (or) it doesn't throw any exception,irrespective of the data existence it will create proxy. And while accessing the data from proxy only it throws an Exception.
Why load is not returning null?and why it is throwing Exception?
Note:Assume for primarykey '3',there is no data.
whenever call session.get(Customer.class,3) then Session returns Customer object either with pre-populated data or if the record is not found then it returns null.

Where as whenever we call customer.getEmail() then it will checks wether for the given customer_Id ,data is there or not? by going to the database.If the corresponding record is not there then Accessor method can return null also.
Note:Assume for primarykey '2', data is there .
whenever call session.get(Customer.class,2) then for this data exists and if we call customer.getEmail() then it goes to the database and queries the data with primarykey value as 2,and return email.Lets say if the Email column doesnt contain any data then also return null.So distinguish between these two null's is a complicated task,thats why if the record is not there the it throw Exception,where as if the record doesnt contain any data then it returns null.

Summary:
get()

- In case of get() always returns actual Entity object populated with data by querying from database,it is **eager loading**.
- If the record doesnt exist in the database for the primarykey value it retuens null.
- If we wish to access the data from the real object then only we will use get().
- Easy to determine wether the record is existed or not?(because imediately we will get if we perform simple check)

load()

- Load() will never access the data from the database,it always creates an proxy class object by populating primary key value.
- when we access the Accessor methods then only data will be accessed from database, that means it is **lazy loading**.
- If the record doesnt exists for the primary key then it doesnt throws any Exception still it creates proxy object with primarykey coulmn,but when we tried accessing the data using proxy,if the data is not there then only Exception will be reported.
- While persisting the child objects,to associate those child objects with the parent,we can fetch the parent using load() rather than using get(),so that the real data will not be fetched.

17-06-2016

How Proxy works Internally:
Proxy looks like as orignal class,talking to the Proxy and talking to the original both are same.proxy will never modify the underlying functionality of the original class.Proxy will exists between enduser and the real object to apply some additional functionaities on the actual class,proxy helpful in performing optimization by not fetching the data from database immediately.Hibernate uses **javaAssist** to create runtime proxy class object.Proxy Contains the primary key value,along with primaryKey proxy contains real object and for all the attributes all the corresponding accessor methods will be overriden as part of the proxy class.Hibernate will have one generic class called "GlobalDataLoadHandler" which contains one generic method public void loadData(),to this class we will pass class object and object id.If the data is not there then it will talk with the session with the help of ID and will get the data.Any of the proxy classes that are being created in Hibernate will calls "GlobalDataLoadHandler" by passing the real object and ID,where the class will takes care of fetching the data from the database and returns into the original proxy class objects which is requesting.
How to create proxy using javaAssist.
Requirement#1
<Buiding an application where we should be able to read the contents of a file and display the data to the user.>
To achieve this requirement we need two classes,one is for reading and the other is for displaying the data.

18-06-2016

Hibernate allows us to query one object of data from a Database in two ways,i.e using get() and load() methods ,Here we can query one object of data against primary key value.

In case of get() always returns actual Entity object populated with data by querying from database,it is **eager loading**.

Load() will never access the data from the database,it always creates an proxy class object by populating primary key value.when we access the Accessor methods then only data will be accessed from database, that means it is **lazy loading**.

Proxy looks like as orignal class,talking to the Proxy and talking to the original both are same. Proxy will exists between enduser and the real object to apply some additional functionaities on the actual class,proxy helpful in performing optimization by not fetching the data from database immediately.Hibernate uses javaAssist to create runtime proxy class object.

19-06-2016(No Class)& 20-06-2016

Creating Proxy:

Hibernate uses **javaAssist(**it is part of jdk no additional libraries required) runtime proxy library to create runtime proxy class object.If we go for runtime proxy then the number of subclasses are comparatively less.

Working Example:

```
javaAssistProxy
|-src
    |-com.jp.beans
        |- Reader.java
        |- FileReader.javass
    |-com.jp.handler
        |-BuferedDataHandler.java
    |-com.jp.test
        |-JPTest.java
```

Lets assume we have a FileReader class to read the data from a file and we wanted to return a text,so it will looks as follows..

Reader.java

```
public interface Reader {
      String getData(String fname);
}
```

Filereader.java

```
public class FileReader implements Reader {

        public String getData(String fname) {

                return "Good Morning!";
        }
}
```

we need to write one more class as BufferedDataHandler and it should be implements from InvocationHandler and it should override a method invoke().

BufferedDataHandler.java

```
public class BufferedDataHandler implements InvocationHandler {
private FileReader reader;

//we will not allow the handler to be created with real object
public BufferedDataHandler(FileReader reader) {
           this.reader = reader;
}
@Override
public Object invoke(Object proxy, Method method, Object[]
args)throws Throwable {
                String data = null;
                data = reader.getData((String) args[0]);
                data += " guys!";
                return data;

        }
}
```

TestClass

```
public class JPTest {
         public static void main(String[] args) {
                  Reader reader = null;
reader = (Reader)
Proxy.newProxyInstance(Reader.class.getClassLoader(),new Class[] {
Reader.class }, new BufferedDataHandler(new FileReader()));

                /*reader = new FileReader();*/
 String data = reader.getData("c:\\readme.txt");
                System.out.println(data);
          }
}
```

21-06-2016

Limitations of JavaAssist Library:
1.It can proxy only Interface,it can not proxy a direct class.
2.JavaAssist proxy will be apply for all the methods, we can not choose certain methods.
3.Target object will not be available in handler.
To overcome these limitations we can go for **cglib** third party library.
Working Example:

```
GetVsLoad
|-src
    |-com.gl.entities
    |-com.gl.util
    |-com.gl.test
|-hibernate.cfg.xml
```

The Entity class is ServiceType and it looks like as below.

```
 public final class ServiceType implements IServiceType {
            protected int serviceTypeId;
           protected String serviceName;
//setters and getters.

}
```

To work with this class we need mapping file and it will be as follows

```
<hibernate-mapping package="com.gl.entities">

        <class name="ServiceType" table="SERVICE_TYPES" >

                <id name="serviceTypeId" column="SERVICE_TYPE_ID" />

                <property name="serviceName" column="SERVICE_NM" />

        </class>

</hibernate-mapping>
```

after this we need to write hibernate.cfg.xml file as below..

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

```
name="connection.username">next_gen_auto_sys</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.show_sql">true</property>
<mapping resource="com/gl/entities/ServiceType.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

After this we need to create Session object for this we will write HibernateUtil class which contains the logic.

```
public class HibernateUtil {
            private static SessionFactory sessionFactory;

static {
Configuration configuration = new Configuration().configure();
StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
           builder.applySettings(configuration.getProperties());s
StandardServiceRegistry registry = builder.build();
sessionFactory = configuration.buildSessionFactory(registry);
}
public static SessionFactory getSessionFactory() {
     return sessionFactory;
}

public static void closeSessionFactory() {
            if (sessionFactory != null) {
                      sessionFactory.close();
                }
             }
   }
```

Here we avoiding Dao to test the working nature of get and load in test class easilly.

get() example:

```
public class GLTest {
      public static void main(String[] args) {
          SessionFactory sessionFactory = null;
               Session session = null;
try {
       sessionFactory = HibernateUtil.getSessionFactory();
```

```
        session = sessionFactory.openSession();
serviceType = (ServiceType) session.get(ServiceType.class, 1);
        } finally {
                        if (session != null) {
                                session.close();
                        }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

load() example:

```
public class GLTest {
       public static void main(String[] args) {
           SessionFactory sessionFactory = null;
            IServiceType serviceType = null;
                Session session = null;
try {
        sessionFactory = HibernateUtil.getSessionFactory();
        session = sessionFactory.openSession();
serviceType = (IServiceType) session.load(ServiceType.class, 1);
//To check which class is being called.
System.out.println(serviceType.getClass().getName());
//If we call accessor(get) methods then only it will get the data.
try {
System.out.println("Type Id : "+ serviceType.getServiceTypeId());
System.out.println("Service Name : "+ serviceType.getServiceName());
//If id is not there still it will return proxy object but when we call
non-accessor method primarykey value then it  throws exception.
} catch (ObjectNotFoundException e) {
        // supress the exception
               serviceType = null;
 }
if (serviceType == null) {
           System.out.println("Object not found");
                        }
}
finally {
                        if (session != null) {
                                session.close();}
HibernateUtil.closeSessionFactory();
                }
        }
}
```
How to eliminate in creating proxy,how to create actual object?
#when we specify the attribute as" lazy="false"proxy="IServiceType">" then even though if we use seesion.load then the proxy will not be created.

22-06-2016

Get() and load() are API methods which are part of Session,these are meant for accessing one single entity object data from the database against the primary key.
→When we use get() method to query then get() will immediately goes to the database and gets the data and populates it into the entity object and will return it,so it is called Eager loading.If the corresponding not found within the database then the get() will not throw an Exception rather it is going to return "null".
→when we use load() then it will not immediately goes to the database rather it creates Runtime Proxy,by populating ID as an input which we passed as an input to the Load() into the proxy object and returns the proxy to support lazy loading.It never checks the existence of record in the database rather its just create runtime proxy object and returns the reference to us.when we tried to calling any of the non primary key accessor methods then only it goes to the underlying database and queries the complete data of the entity object and returns as an value by populating into object.If the corresponding object doesnt exists witinin the database then it doesn't return null,still it returns empty object with primay key being populated.(It will not throw any Exception) but if we tried to calling any of the non primary key accessor methods then it throws ObjectNotFoundExceptiono that too if it is not there within the database only.The load can not be transfered across all the layers of the project,Under an Active session only th proxy will get the data from the database.So a necessary care has to be taken in working with load().

If we declare our Entity class as final,lazy=true and if we tried calling session.load() then will it throw any Exception?will it creates proxy?
If we want to work with load always it is not recommended to make our Entity class as Final only.When we call session.load(EntityClass,PK) then it will not throw exception rather it will creates original object.

Here we need to create one more Interface as IserviceType and it contains the setters and getters of ServiceType as follows..

```
public interface IServiceType {
        public int getServiceTypeId();
        public void setServiceTypeId(int serviceTypeId);
        public String getServiceName();
        public void setServiceName(String serviceName);
}
```

And here our ServiceType class should implement from this interface as public final class ServiceType implements IServiceType.
 So when we call session.load()   it is going to proxy the Interface and our Implementation class can not be received.So now it will create proxy implemting from interface,it doesnt create proxy extending from actual class.
The benefit of going for nterface is it makes us to easilly switch from get() and load() methods.
Note:Designed to inteface ,never designed to concrete classes.
when we call ServiceType = (IServiceType) session.load(ServiceType.class, 1); then session object takes the interface and goes to the SessionFactory and tries to find wether the interface is mapped or not?class will have mapping But Interface doesnt have mapping that means in the mapping file at the class level we need to specfiy as <class name="ServiceType" table="SERVICE_TYPES" lazy="true" proxy="IServiceType"> now the SessionObject will goes to the SessionFactory and tries to ask him about existence of IserviceType mapping information ,if the proxy name is available it will goes to the table and creates all the required things.
 Note:So by proxying the interfaces we can create the proxy even though i the class is final.

Complete Example:

```
GetVsLoad
|-src
    |-com.gl.entities
        |- IserviceType
        |- ServiceType.hbm
        |- ServiceType
    |-com.gl.util
        |- HibernateUtil
    |-com.gl.test
        |- GLTest
|-hibernate.cfg.xml
```

ServiceType.java

```
public final class ServiceType implements IServiceType {
             protected int serviceTypeId;
            protected String serviceName;
//setters and getters.

}
```

IServiceType.java

```
public interface IServiceType {
            public int getServiceTypeId();
            public void setServiceTypeId(int serviceTypeId);
            public String getServiceName();
            public void setServiceName(String serviceName);
}
```

ServiceType.hbm.xml

```
<hibernate-mapping package="com.gl.entities">

        <class name="ServiceType" table="SERVICE_TYPES" lazy="true"

                proxy="IServiceType">

                <id name="serviceTypeId" column="SERVICE_TYPE_ID" />

                <property name="serviceName" column="SERVICE_NM" />

        </class>

</hibernate-mapping>
```

After this we need to create Session object for this we will write HibernateUtil class which contains the logic.

```
public class HibernateUtil {
            private static SessionFactory sessionFactory;

static {
Configuration configuration = new Configuration().configure();
StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
           builder.applySettings(configuration.getProperties());s
StandardServiceRegistry registry = builder.build();
sessionFactory = configuration.buildSessionFactory(registry);
}
public static SessionFactory getSessionFactory() {
     return sessionFactory;
}

public static void closeSessionFactory() {
            if (sessionFactory != null) {
                      sessionFactory.close();
                   }
             }
   }
```

Here we avoiding Dao to test the working nature of get and load in test class easilly.

```
public class GLTest {

       public static void main(String[] args) {
       SessionFactory sessionFactory = null;
       IServiceType serviceType = null;
              Session session = null;

try {

                     sessionFactory = HibernateUtil.getSessionFactory()
                     session = sessionFactory.openSession();
       serviceType = (IServiceType) session.load(ServiceType.class, 1);
System.out.println(serviceType.getClass().getName());
              try {
System.out.println("Type Id : "+ serviceType.getServiceTypeId());
System.out.println("Service Name : "+ serviceType.getServiceName());
       } catch (ObjectNotFoundException e) {

                            // supress the exception
serviceType = null;    }
                        if (serviceType == null) {
System.out.println("Object not found");}
} finally {
               if (session != null) {
                          session.close();
}
                   HibernateUtil.closeSessionFactory();
               }
         }
}
```

<u>Hibernate Tools</u>

While working with Hibernate we may have to do several operations like Creating Schema,creating SQL Query Scripts to create Tables.

Why we need Hibernate Tools?

To derive the corresponding tables for the Entity classes and the mapping information associated with in the corresponding entity class even then also we can not work directly rather we need corresponding table with in the database ,so programmer has to write the SQL Scripts for creating those tables,which is a duplicate effort.So to avoid creating tables manually by the programmer,Hibernate has provided Certain Tools to work with both the mechanism like Tables to Entiites and Entities to Tables.Tools helps the developer to develop application with greater speed.

What are the various Tools provided by Hibernate?

Hibernate has provided certain Tools in multiple forms like.....

1.GraphicalUserInterface Tools
 |-Hibernate Configuraration console
 |-Hibernate Mapping console.
 |-Hibernate Query language Editor.
 |-DataBaseExploer Tool
 |-ReverseEngineering Tools

2.Command Line Based Tools
 |-SchemaExport Tool
 |-SchemaUpdate Tool
 |-SchemaValidate Tool.

3.Programmatic Tools and
4.Configuration Based Tools.

23-06-2016

For supporting rapid application development The Hiberanate has provided Ecclipse pugins.The GraphicalUserInterface Tools are not availble across all the platforms, Ecclipse has support for working with GraphicalUserInterface Tools,So Hibernate has provided one plugin called Hibernate Tool.Apart from the GraphicalUserInterface Tools hibernate also provided CommandLine Tools,which are bundled as part of the Core Distribution also.

GraphicalUserInterface Tools

To work with GraphicalUserInterface Tools we need Internet Connection.we can search for these tools in the market places of Ecclipse.

(------------- Ecclipse Related work--------------)

24-06-2016

CommandLine Tools:

Using CommandLine Tools we can automate Schema generation, updating schemas and validating schemas with the help of the rich set of tools provided by Hibernate and the tools are as below.

|-SchemaExport Tool
|-SchemaUpdate Tool
|-SchemaValidate Tool.

As part of the application we will write Entity classes,Lets say we created an entity class as Customer and it will be as follows.

```
public class Customer {
        private int id;
        private String firstName;
        private String lastName;
        private String mobile;
        private String email;
        private Date dob;
        private String gender;
//setters and getters
}
```

Along with this we need maaping information also and it will as below..

```
<hibernate-mapping>
  <class name="com.fh.entities.Customer" table="CUSTOMER">
        <id name="id" type="int">
              <column name="CUSTOMER_ID" length="5" />
```

```
      </id>
<property name="firstName" type="string">
<column name="FIRSTNAME" not-null="true" length="50" />
</property>
-------------------------
   </class>
</hibernate-mapping>
```

Along with the mapping file we also need configuration file also,and we also have a relavant table witin the database and its programmers responsibility to create the Table.In order to create the table we need SQL Scripts.If we are writing SQL Queries then theseSQl queies will not work across all the Environments.So we should have a mechnaism to make sure the database SQl queries that are going to generate the schema should be independent and are to be easy to create the schemas across all the environments.Irespective the Database Type To create the tables within the corresponding database based on the Entity classes that we have within our application, we can make use of the CommandLine Tools provided by Hibernate.

Is it mandatory to specify the "Type" within mapping file?

It is optional to specify the "type".In case if we failed to specify the type then Hibernate is going to introspect the attribute of the class and tries to find the java Type,and for this javaType it tries to identify the corresponding/equivaent hibernate type.database type and it will instruct the schema Export to generate the mappings.

25-06-2016(SAT)& 26-06-2016(SUN)& 27-06-2016

→ **SchemaExport:**

It will generate ddl scripts based on the Entity model within our application,so to the SchemaExport we need to provide Entity classes,configuration file(database info +dialect(mandatory)) and mapping information, within the mapping information we can specify the contraints and etc.

→**SchemaUpdate:**

If there exists a Database for the corresponding entity classes and If there is any change that is beng done within the Entity classes has to be reflected in the database schema we will use SchemaUpdate.

→ **SchemaValidate:**

using we can always cross check wether the Entity classes and the corresponding schema with which we wnted to run the application are valid or not ?

When we tried creating the mapping information in which we declared a property and for that property if we dont have the corresponding coulmn within the database,in such cases will Hibernate throws an Exception?
As SessionFactory never validates and verifies the mapping information against database table which is being loaded into it.
This information will be validated when we tried accessing the data using the relavant entity classes,if the data is not there then it will results a Runtime Exception.
Note:All the SchemaExport ,SchemaUpdate and SchemaValidate are the classes which were shipped as part of the Hibernate distribution itself.If we want to run the SchemaExport class then we need to set the classpath to the Hibernate jar and we need to run the class using the FullyQualifiedName of it.
The SchemaExport class is a part of **org.hibernate.tool.hbm2ddl. SchemaExport;**(ctrl+shift+T) and this tool is availble in the hibernate-core,jar,so first we need to set the classpath as java-cp c:\hibernate\lib\hibernate-core.jar org.hibernate.tool.hbm2ddl. SchemaExport;
If we have a group of commands which has to be repeatedly executed then instead of executing the all commands multiple times even though the order of exeuting the commandsis same,so it s better to create one **bat** file under on of the directory and we can execute the bat file only whenever we wanted to execute the set of commands.
Prasad:(How to run SchemaExport tool is a b it complicated and still need to be addressed.)

28-06-2016

Its all about working with CommandLine Tools in command driven approach.

29-06-2016

working with CommandLine Tools in programmatic approach.
<u>How to Work with SchemaExort</u>

```
Tools
|-src
   |-com.fh.entities
     |-Customer.java
     |-Customer.hbm.xml
   |-com.fh.tools
     |-SchemaExportTest
|-hibernate.cfg.xml
```

|-db-ddl.sql
CompleteExample:

Customer.java

```
public class Customer {
            private int id;
            private String firstName;
       private String lastName;
       private String mobile;
       private String email;
       private Date dob;
       private String gender;
//setters and getters
}
```

Customer.hbm.xml

```
<hibernate-mapping>
  <class name="com.fh.entities.Customer" table="CUSTOMER">
        <id name="id" type="int">
             <column name="CUSTOMER_ID" length="5" />
      </id>
<property name="firstName" type="string">
<column name="FIRSTNAME" not-null="true" length="50" />
</property>
<property name="lastName" type="string">

                  <column name="LAST NAME" length="50" />

</property>

<property name="mobile" type="string">

                      <column name="MOBILE" not-null="true"
unique="true" length="13" />

</property>

<property name="email" type="string">

                   <column name="EMAIL" not-null="true" unique="true"
length="100" />

               </property>

<!-- <property name="dob" type="date">

                      <column name="DATE_OF_BIRTH" />

</property> -->
```

```
            <property name="gender" type="string">
                <column name="GENDER" />
            </property>
        </class>
</hibernate-mapping>
```

Hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
<!-- Database configuration -->
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.hbm2ddl.auto">validate</property>
<mapping resource="com/fh/entities/Customer.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

Test Class:

```
import org.hibernate.tool.hbm2ddl.SchemaExport;
public class SchemaExportTest {
        public static void main(String[] args) {
                Configuration configuration = new
Configuration().configure();
                /*
                 * SchemaExport schemaExport = new
SchemaExport(configuration);
                 * schemaExport.execute(false, true, false, true);
                 */
                StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                SessionFactory factory =
configuration.buildSessionFactory(registry);
                factory.close();
        }
}
```

properties approach:

db-ddl.sql

```
drop table CUSTOMER cascade constraints
create table CUSTOMER (CUSTOMER_ID number(10,0) not null, FIRSTNAME
varchar2(50 char) not null, LASTNAME varchar2(50 char), MOBILE
varchar2(13 char) not null, EMAIL varchar2(100 char) not null,
primary key (CUSTOMER_ID))
alter table CUSTOMER add constraint UK_rlhhgj4x81jlr1c4rw9f4s3s5
unique (MOBILE)
alter table CUSTOMER add constraint UK_mk3cgvpjoy0vr5tbjwp5g13i1
unique (EMAIL)
```

hibernate.cfg.xml:

```
<hibernate-configuration>
   <session-factory>
         <!-- Database configuration -->
<property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name="hibernate.hbm2ddl.auto">validate</property>
<mapping resource="com/fh/entities/Customer.hbm.xml" />
     </session-factory>
</hibernate-configuration>
```

Points to Remember:
1.During the development we will use SchemaUpdate.
2.Durig the production we will use SchemaValidate.
3.During the test cases we will use create and drop(SchemaExport).

30-06-2016

Working with Annotations:

If we wanted to persist the data that is there within the object of Entity class then we can not pass the object directly to the Hibernate and Hibernate can not able to performing persitency operations.Hibernate can not identify the corresponding table and coulmn into which it has to persist the data automatically.Unless and untill we provide the relavant mapping information as an input to the Hibernate,it will not persist the data.

So we need to provide both the configuration and mapping information to make our components persistable with the help of Hibernate.

Hibernate is one of the frameworks which we will depends on meta data information that is being passed interms of XML configuration file.But it is difficult to work with Xml because the amount of configuration could be more and one has to remember the tags in writing Xml due to this Programmers will ended up in writing the loads and loads of configuration files only.

To avoid the burden to the programmer from writing more amount of configuration file,Hibernate people though an alternative which eliminates efforts in writing configuration files and they brought of the annotation based configuration.With the Annotations approach the developer will be least bothered about working with Hibernate in terms of writing mapping and configuration approach.

Problems with XML Based Configuration/mapping :

(1) It is not Easy because One has to remeber the several Tags(like DocType tags and etc..) in writing XML configuration files.

(2) As mapping information has been scattered across multiple files,one has to consalidate all such information to derive the relationship between the Entity classes,otherwise we can not easilly understand.

(3)XML based mapping will not gets validated at the stratup of our application,rather it gets validated during its usage during runtime,which results in Runtime Exception.

Considering all the problems we can go for Annotations to eliminate all such type of problems.Annotations are called as Source code metadata becuase those will be configured at source code level.

Benefits of Annotation Based Mapping:
(1) Annotations are alternatives of XML Configuration approach, Annotations are short in nature and these will not scattered in multiple words /charcacters,as those are short we can remember them easilly.
(2)we can easilly derive the mapping information about Entity classes also if we use annotations.
(3) Annotations will gets validated during the compile time itself.
Considering all the benefits Hibernate started supporting to work with Annotations along with XML Configuration.
Note:The Configuration mechanism still has to be written in hibernate.cfg.xml,because it can not annotatable.
→**H**ibernate has added the support for working with annotation from Hibernate 3.0.
From the Hibernate 3.0,it is being served in two ways as follows...
[1]Hibernate Core
[2]JPA Api
So we will have two sets of annotations,one is provided by Hibernate and the other one by JPA,both resembles the same(same annotations).
which annotations are recommended to use?
It is recommended to use JPA annotations.
How many types of annotations are there in Hibernate?
Annotations wihtin the Hibernate are broadly classified into two groups and they are..

- **Logical Annotations**

#1-Anything that are describing related to the structure of Entity class are called Logical Annotations.
#2-These are optional,even though we didnt write these Annotations our Entity classes will be persistable.

- **Physical Annotations**

#1-Anything that makes our Entity class mapped to the underlying schema model are physical annotations.
#2-These are mandatory unless and untill we write these annotations the Entity classes will not be persistable.

→How do we need to work with Annotations:
Lets say we have an Entiy class Customer as follows with the mentioned attributes and accessor methods.

```
public class Customer {
	private int id;
	private String firstName;
	private String lastName;
	private String mobile;
	private String email;
	private Date dob;
	private String gender;
//setters and getters
}
```

If we wanted to persist the data that is contained within the object of this class we need mapping file,so we need to write the mapping information as part of the Entity class with the help of Annotations.

@Entity

It will be used to map a class As Entiy class,it is mandatory annotation because without it hibernate can not identify the class to persist the data.It is logical Annotation.

@Table(name = "Customer")

By default theEntity class name will be considered as Table name because Hibernate follows convention over configuration.But if we wanted to change the table then we need to use this annotaton.
It is physical annotation,it is optional.

@Id

This annotation is used to annotate the attribute of the Entity class which is representing as primary key.It is mandatory annotations.

@Column(name = "Customer_ID", nullable = false)

By default the attribute names will be considered as coulmn names,incase if the attribute names and coulmn names are not matching then we can use this annotation.We can even specify it as not null.

@Basic

It is used for specifying the fetch type is "Lazy" or "Eager" for the speciifc attribute.It will applicable for relational objects.It will acts as default annotation if we didnt provide any annotations for the attributes.

In how many places we can write the annotatons?

→we can write the annotations at both the attribute and Accessor method level.

Note:As per the EJB speciifcation we should not write annotation at setter methods,rather we should write them at getter methods only.

Is it entertained to write the annotations at both the levels for a Entity class?
No it is not entertained to write the annotatons at both the levels of the entity classes,because Hibernate will not read the both level annotations rather we should follow consistency in writing them at one place for one Entity classes.

Writing annotaton at which level is recommended?
we can write the annotation at one of the either of two levels available. Hibernate determines the access type(attribute,method level) depends on the place where we have written the @Id.
Note:In the absence of writing any of the annotations on any of the attributes,The Hibernate internally thinks that the default annotation that will be annotated with for those attribute is @Basic.
How to make a column name as unique within the database using annotation based configuration?
If we wanted to make one of the coumn within the database table as unique then we can write the following annotations at class level.
uniqueConstraints = { @UniqueConstraint(columnNames = {"MOBILE", "DOB" }) })

01-07-2016

@Transient:
if we have a speciifc attribute as are of the Entity class,which we dont want to persistable rather it is a calculated value that will gets populated through the programme.In such case such kind of attributes will be annotated with @transient annotation.
@UniqueConstraint(columnNames = {"column1", "Coulmn2" }) })
Using this we can specify the UniqueConstraints at table level as and we need to specify the coulmn names, here as we are specify the attribute names themselves as column names as follows..
@Table(name = "EVENT", uniqueConstraints = {
@UniqueConstraint(columnNames = {"EVENT_DT", "PLACE" }) })
Working Example:
```
FirstAnnotationHibernate
|-src
   |-com.fha.entities
   |-com.fha.util
   |-com.fha.test
|-hibernate.cfg.xml
```
Lets take an An Entity class **Event** which can be used to capture the information about Event,Event will have the properties like int id,

String description,Date eventDate,String place, int priority,boolean remainder.The Entity class will looks as below and we need to annotate tis class with the required annotations to perssit into the database rather than writing Xml based maping file s below..

```
@Entity
//Unique constraints will be described at Table level.
@Table(name = "EVENT", uniqueConstraints = {
@UniqueConstraint(columnNames = {"EVENT_DT", "PLACE" }) })
public class Event {
  @Id
        //Here for the Id we dont need to specify Nullabale constraint.
        @Column(name = "EVENT_ID")
       private int id;
 @Column(name = "DESCR", nullable = false)
        private String description;
        @Column(name = "EVENT_DT", nullable = false)
        private Date eventDate;
        @Column(name = "PLACE", nullable = false)
        private String place;
        @Transient
//if we annotate the attributes with @transient then the column for the particular attribute will not be created at Table level,the data will not gets persisted.
        private int priority;
//As here we are not specifying any annotation,so the default annotation is @Basic only which we dont need to specify it expilicitly.
        private boolean reminder;
//seters&getters
}
```

Once we have the mapping information interms of annotations next we need to have configuration information as follows..

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

```
            <property
name="connection.password">welcome1</property>
            <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
        <property name="hibernate.show_sql">true</property>
//as we dont have the Event table within the hib_tools database and
we wanted to create the Table by hibernate it self so we configured it as
auto.
        <property name="hibernate.hbm2ddl.auto">update</property>
//as the mapping informatin has been written at class level we need to
pass the Entity class name as mapping class to read the annotations.
        <mapping class="com.fa.entities.Event" />
        </session-factory>
</hibernate-configuration>
```

Once we have the Configuration file next we need to write the HibernateUtil class as follows

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
//we want to make the SessionFactory as singleton so we will place
the Static block.
static {
//we can use Configuration class only to read the annotations of
Entity class.
        Configuration configuration = new Configuration().configure();
//To create StandardServiceRegistry we need Builder.
                StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);

        }

        public static SessionFactory getSessionFactory() {
                return sessionFactory;

        }
        public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
```

```
                    }
            }
    }
now we can write the test class as follow..
public static void main(String[] args) {
        SessionFactory sfactory = null;
//we  need transactions because we are manaing DML operations.
        Transaction transaction = null;
        Session session = null;
        boolean flag = false;
try{
 sfactory = HibernateUtil.getSessionFactory();
        session = sfactory.openSession();
       transaction = session.beginTransaction();
   Event event = new Event();
                event.setId(1);
                event.setDescription("Web Service Sessions");
                event.setEventDate(new Date());
                event.setPlace("Mithrivanam");
                event.setPriority(1);
//if remainder is required we can make it true.
                event.setReminder(false);
                session.save(event);
//after performing the perssitency the last statement within the try
should be  making flag  as true.
        flag = true;
}finally {
                if (transaction != null) {
                        if (flag == false) {
                                transaction.rollback();
                        } else {
                                transaction.commit();
                                        }
                        }
                if (session != null) {
                        session.close();
                                }
                HibernateUtil.closeSessionFactory();

                }
        }
```

CompleteExample:

```
FirstAnnotationHibernate
|-src
   |-com.fha.entities
      |-Event.java
   |-com.fha.util
      |-HibernateUtil.java
   |-com.fha.test
      |-FATest.java
|-hibernate.cfg.xml
|-lib(jars)
```

Event.java

```
@Entity
@Table(name = "EVENT", uniqueConstraints = {
@UniqueConstraint(columnNames =  {"EVENT_DT", "PLACE" }) })
public class Event {
  @Id
  @Column(name = "EVENT_ID")
       private int id;
  @Column(name = "DESCR", nullable = false)
        private String description;
  @Column(name = "EVENT_DT", nullable = false)
        private Date eventDate;
  @Column(name = "PLACE", nullable = false)
        private String place;
   @Transient.
        private int priority;
        private boolean reminder;
//seters&getters
}
```

hibernate.cfg.xml

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

```
		<property
name="connection.password">welcome1</property>
		<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
	<property name="hibernate.show_sql">true</property>
	<property name="hibernate.hbm2ddl.auto">update</property>
	<mapping class="com.fa.entities.Event" />
	</session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
	private static SessionFactory sessionFactory;
static {
	Configuration configuration = new Configuration().configure();
	StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
		builder.applySettings(configuration.getProperties());
		StandardServiceRegistry registry = builder.build();
		sessionFactory =
configuration.buildSessionFactory(registry);

	}
	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}
	public static void closeSessionFactory() {
	if (sessionFactory != null) {
		sessionFactory.close();
		  }
	}
}
```

FATest.java

```
public class FATest {
public static void main(String[] args) {
	SessionFactory sfactory = null;
	Transaction transaction = null;
	Session session = null;
	boolean flag = false;
try{
 sfactory = HibernateUtil.getSessionFactory();
        session = sfactory.openSession();
       transaction = session.beginTransaction();
   Event event = new Event();
               event.setId(1);
               event.setDescription("Web Service Sessions");
               event.setEventDate(new Date());
               event.setPlace("Mithrivanam");
               event.setPriority(1);
               event.setReminder(false);
               session.save(event);
               flag = true;
}finally {
               if (transaction != null) {
                       if (flag == false) {
                               transaction.rollback();
                       } else {
                               transaction.commit();
                               }
                       }
               if (session != null) {
                       session.close();
                               }
               HibernateUtil.closeSessionFactory();

               }
       }
```

Note:when we run this program 'priority' will be missed because,we annotated with @Transient.

FirstLevelCache

What is Cache?
why do we need to use cache?
Where(Under which circumstances) can we use cache?
How does cache works internally within Hibernate?
If we have some informaton which we wanted to repeatedly access then instead of frequently accessing such data from the underlying database,we wanted to access and store within the cache.So that we can reduce the round trips to the database with which eventually we can improves the performance of the application.

UseCase🌏

Online Shopping Applications(Myntra,FlipKart,SnapDeal and etc)
Lets say we have a **browse.jsp** page where products displayed as part of the page and we can browse for the list of products for a speciifc category. Whenever The user clicked on the selected product then it should able to get the details of the product and it should display to the user. As it is a shopping application multiple users can interact with it and they can try to fetch the same data(repeatedly access).

Probelms with repeatedly accessing the data(without cache):
1.Un necessary CPU cycles will be wasted in re-fetching the same data from the database.
2.Database Round trips will be more even though the users are querying the same data.
3.Memory will be wasted by creating multiple objects for multiple requests though they requires same data.
4.Network bandwindth will be increased if it is remote application.
So to avoid above said problems aand we should store the data temporarily in a place from where the user can get the data instead of going to the database,for this we can bring up the cache concept.
5.JVM memory will be occupied with same object snapshots.

Rules in creating a Cache:
1.The Cache Class should be Singleton.

How to make a Cache class as Singleton:

✓If any Class which allows us to create only one object then that class can be called as Singleton.

✓By default the scope of any java class is non-singleton only.

✓To make a class as singleton we shoud make the **constructor** as **private,**so that only the methods of the class can access the constructor.

```
private Cache() {
```

as we need to create the object of dataMap once only, so its best place to create the object within the private constructor.

```
dataMap = new ConcurrentHashMap<String, Object>();}
```

✓The Singleton classes will have static factory methods which contains the logic for creating the objects as follows.

```
public synchronized static Cache getInstance() {}
```

✓within the cache for every piece of information that we wanted to store we need to to attach one unique identifier to access the data from the cache.Data will be stored in Key and values pairs within cache.

```
private Map<String, Object> dataMap;
```

✓To store the data within map we need some methods as below..

public void put(String key, Object value) public Object get(String key) {
public boolean containsKey(String key)

so the Cache class will looks like as below..

```
public class Cache {
        private static Cache instance;
        private Map<String, Object> dataMap;
        private Cache() {
                dataMap = new ConcurrentHashMap<String, Object>();
        }
        public synchronized static Cache getInstance() {
        if (instance == null) {
                instance = new Cache();
                }
        return instance;
        }
        public void put(String key, Object value) {
             dataMap.put(key, value);
                }
         public Object get(String key) {
                 return dataMap.get(key);
                }
          public boolean containsKey(String key) {
                return dataMap.containsKey(key);
     }
}
```

03-07-2016(Sun No class)&04-07-2016

When will(circumstances)we use cache?

✓If the volumes of requests who are trying to access the data within the time are high then we can use cache,but we should employ techniques to identify the stale(outdated data within cache) data.

✓when we use the cache we should never modify the data in the underlying source system directly if such kind of requirement is not there(read-only) then we can use cache.

→Between the classes,database and cache we will build one intelligent agent called “persistence-manager”.It acts as a mediator between the cache and database.persistnece-manager holds connection and will takes care of reflecting the changes in both the database and Hibernate cache.

05-07-2016(No class)& 06-07-2016

Cache:

Cache is used to store the data temporarily within the application/ JVM memory to avoid repeatedly accessing the data from the underlying source.

Advantages of cache:

1.we can avoid huge amount of data transfer between the database and application server,so that the bandwidth space of network will be saved.
2.The traffic conversation can be reduced.
3.The performance will be more when we use cache.
4.It eliminates in creating same objects in JVM memory.

when we will use cache:

1.when we use the cache we should never modify the data in the underlying source system directly if such kind of requirement is not there(read-only) then we can use cache.
2.Even if the underlying source is changing the voume sof users who are accessing the data are high within the time hour,to save the performance we can go for cache.

when we will not use cache:

1.If the data that is there within the source system is being directly modified without the interaction with the application,as the application never aware of such kind of changes at data source level,so the data that is being stored in the cache will become stale data.In such cases we should never use case.

→Always while performing any of the DML operations across the components within our application,we should not only make sure in performing the persistency operations but also it is required to write the cache managment logic also in all the classes within our application.

Here Database is actual persistence store and cache is virtual perssitence store ,so we need to ensure the changes should be reflected in both these persistency stores so we need to write the logic for reflecting changes in both the source systems across all the clasees within our application.But it is problem to write logic,So we should not dump the whole data in the cache.here we can have two types of data where one will changes frequently where as the other will not?out of these two types of data data which is getting changed frequently is major(70%) one.

FirstLevelCache:

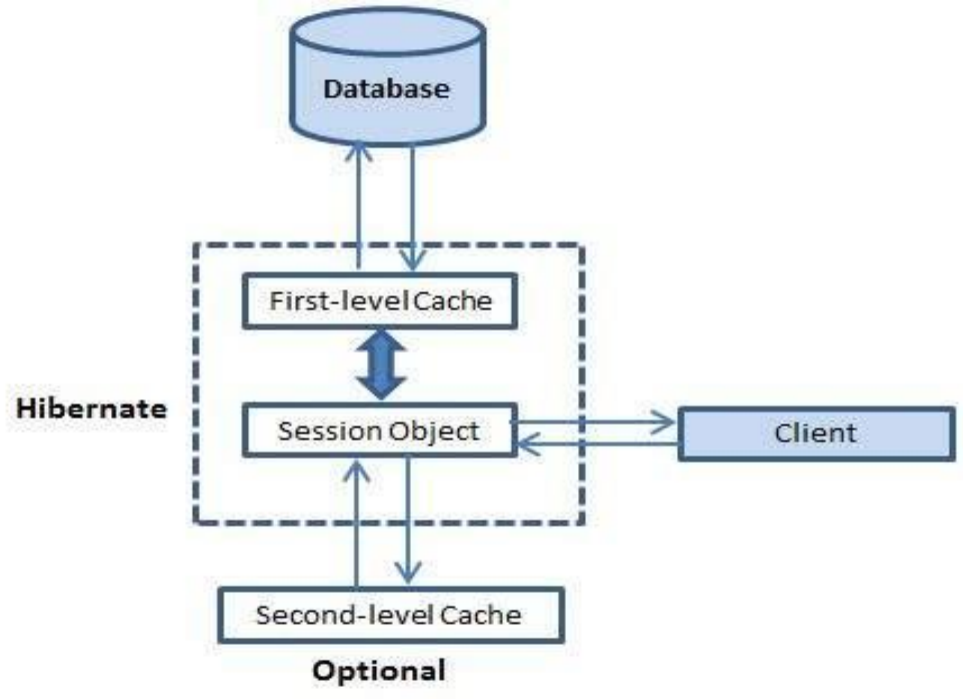
If we wanted to access the data from the database using Hibernate then we can use either get() or load() methods which are there with Session,so every user will uses Session object to query the data from the database.Whenever we called session.get(Cutomer.class,1) the the Session object should go to the database and should query the data from the database and should return it.

Where as within the same session if we tried again calling Session.get(Customer.class,1) any number of times then the Hibernate should not go the database for querying the data from the data.because within the interaction of using the session the chances of getting the changes within the underlying database will be less.In such case if we create more number of objects for accessing the data then the objects will gets bloated up within the JVM Memory.So Hibernate also has cache at interaction level,As Session will interact with database and session performs operations on the database that means Session only can manage the cache.So every time when we create a session **Hibernate** will **creates** one **cache by default**.As cache is being creating by hibernate at **session level**,the persistence manager role will be played session itself,it will takes care of keeping the changes or making the changes within the database or cahce as well.

So **we will call this cache as FirstLevelCache.**

07-07-2016(No Class)&08-07-2016

The Following diagram depicts How Hibernate Cache looks like...



How it works:
when the application classes ask Session for getting the data then Session will looks for the data in the First-Level cache which is a mandatory cache provided by hibernate by default,As Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache if the data is not there within the First-Level cache.even in the Second-levelCache also if there is any miss then it goes to the data-base and gets the data and store in the first-level cache and returns to the application classes.

what is FirstLevelCache?

The cache that is being maintained at Session level wihtin the Hibernate is called As First-Level cache.Any access to the data or any access to the entity classes from the database can be performed through the FirstLevel cache.

what is the purpose of FirstLevelCache?

Hibernate will uses first level cache to avoid performance issues and to eliminate repeated access to the database by maintning the data temporarily at JVM Memory.

who creates the FirstLevelCache?how does it gets created?
Whenever the user creates the Session object then the First-Level cache will gets created.
can you turn off the Session-level cache(First-level cache)?
No we can not,because as it is not a global cache never the data is going to gets stored at an application level rather it is at interaction level.If the frequent access to the repeated data within the usual interaction has to be cached to save performance and memory Hibernate will not permits us to disable the FirstlevelCache.
why does the Hibernate mandates the user to cofigure Id property?
unless and untill every entity is being associated with unique Identifier the data can not be managed within cache.So every Entity class should have Id attribute.
Can you mange Entiy classes persistency operations without Id property using Hibernate?
No we can not manage,it must be there .
Working Example:

```
FirstlevelCache
|-src
  |-com.flc.entites
  |-com.flc.test
|-hibernate.cfg.xml
```

Lets say we have an Entity class with which we wanted store the information about mobile.The Mobile class will be as follows..

```
public class Mobile {
        private int id;
        private String modelName;
        private String description;
        private String manufacturer;
        private String type;
        private String networkType;
        private float amount;
//setters and getters
}
```

The Mobile class corresponding mapping file will looks as below..

```
<hibernate-mapping package="com.flc.entities">
        <class name="Mobile" table="MOBILE" dynamic-update="true">
        <id name="id" column="MOBILE_ID" type="int" />
```

```
         <property name="modelName" column="MODEL_NAME"
type="string" length="50" not-null="true" />
        <property name="description" column="DESCR" type="string"
length="200" not-null="false" />
        <property name="manufacturer" column="MANUFACTURER"
type="string"length="100" />
         <property name="type" column="MOBILE_TYPE" length="1"
type="string" />
         <property name="networkType" column="NETWORK_TYPE"
type="string"length="4" />
<property name="amount" column="AMOUNT" type="float" />
      </class>
</hibernate-mapping>
```

To interact with the database we need hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
       <property name="connection.username">hib_tools</property>
      <property name="connection.password">welcome1</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
         <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <mapping resource="com/flc/entities/Mobile.hbm.xml" />
     </session-factory>
</hibernate-configuration>
```

After this we need to write the util class as below...

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
        StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
        StandardServiceRegistry registry = builder.build();
        sessionFactory = configuration.buildSessionFactory(registry);

        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;

        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
        }
 }
```

Now we can write test class to check wether it is working or not?

#Inserting The data:

```
public class FLCTest {
        public static void main(String[] args) {
                SessionFactory sfactory = null;
                Transaction transaction = null;
                 Session session = null;
                 Mobile mobile = null;
                 boolean flag = false;
mobile = new Mobile();
                mobile.setId(6);
                mobile.setModelName("OnePlus Three");
                mobile.setDescription("OnePlus Three HD");
                mobile.setManufacturer("OnePlus");
                mobile.setType("s");
                mobile.setNetworkType("GSM");
                mobile.setAmount(32000);
try {
      sfactory = HibernateUtil.getSessionFactory();
      session = sfactory.openSession();
       transaction = session.beginTransaction();
        session.save(mobile);
} finally {
if (transaction != null) {
                        if (flag) {
                              transaction.commit();
} else {
transaction.rollback();
```

```
    }
 }
if (session != null) {
            session.close();
                        }
            HibernateUtil.closeSessionFactory();
             }
        }
}
```

#Accessing the Mobile Data:

Now lets try to get the mobile data.

Here we dont need transactions because it is not DML operation.

```
try {
      sfactory = HibernateUtil.getSessionFactory();
      session = sfactory.openSession();
       transaction = session.beginTransaction();
       session.save(mobile);
}
System.out.println(mobile);
```

To print the mobile we need to override the toString() in the Mobile entity class as below

```
@Override
public String toString() {
            return "Mobile [id=" + id + ", modelName=" + modelName
+ ", description=" + description + ", manufacturer="+ manufacturer + ",
type=" + type + ", networkType="+ networkType + ", amount=" +
amount + "]";
}
```

#Re-Accessing the Same Data:

```
mobile = (Mobile) session.get(Mobile.class, 1);
System.out.println(mobile);
```

Now the data will be fetched from the database only once.because the data will be queried from the FirstLevelCache.

#checking both are same objects or not?

```
System.out .println("mobile 1 == mobile 2 ?: " + (mobile ==mobile1));
```

Now it will prints **true** because data is being cached from FirstlevelCache.

#Accessing the data using Load()

#Checking the cache is at SessionLevel or not?

```
mobile = (Mobile) session.get(Mobile.class, 1);
```

```
 System.out.println(mobile);
 session.close();
```
It will execute the above query two times that means firstlevelcache will be associated at session level.

Note:if we call session.save(mobile) the the mobile object will not gets persisted rather it will be stored in the cache,when we issue commit then only the data will be persisted in the database.

#How Save will work:
```
transaction = session.beginTransaction();
session.save(mobile);
System.out.println("mobile saved...");
```
🌏Session.flush()vs 🌏transaction.commit()

Flush() will takes the changes that has been reflected within the cache and will gets executed onto the underlying database.flush will writes the changes into the database.Commit() will internally call flush.
```
transaction = session.beginTransaction();
     session.save(mobile);
 System.out.println("mobile saved...");
     session.flush();
System.out.println("session flushed...");
```
CompleteExample:
```
FirstlevelCache
|-src
  |-com.flc.entites
       |-mobile.java
       |-Mobile,hbm.xml
  |-com.flc.test
       |-FLCTest.java
  |-com.flc.util
        |-HibernateUtil.java
|-hibernate.cfg.xml
```
Mobile.java
```
public class Mobile {
        private int id;
        private String modelName;
        private String description;
        private String manufacturer;
        private String type;
        private String networkType;
        private float amount;
//setters and getters
//override toString()
}
```

Mobile.hbm.xml

```
<hibernate-mapping package="com.flc.entities">
        <class name="Mobile" table="MOBILE" dynamic-update="true">
        <id name="id" column="MOBILE_ID" type="int" />
          <property name="modelName" column="MODEL_NAME"
type="string" length="50" not-null="true" />
        <property name="description" column="DESCR" type="string"
length="200" not-null="false" />
        <property name="manufacturer" column="MANUFACTURER"
type="string"length="100" />
          <property name="type" column="MOBILE_TYPE" length="1"
type="string" />
          <property name="networkType" column="NETWORK_TYPE"
type="string"length="4" />
<property name="amount" column="AMOUNT" type="float" />
      </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
        <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
       <property name="connection.username">hib_tools</property>
      <property name="connection.password">welcome1</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
         <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <mapping resource="com/flc/entities/Mobile.hbm.xml" />
     </session-factory>
</hibernate-configuration>
```

HibernateUtil.java
```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
        StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
        StandardServiceRegistry registry = builder.build();
        sessionFactory = configuration.buildSessionFactory(registry);

        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;

        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
        }
 }
```
FLCTest.java
```
public class FLCTest {
   public static void main(String[] args) {
        SessionFactory sfactory = null;
                Transaction transaction = null;
                Session session = null;
                Mobile mobile = null;
                boolean flag = false;
mobile = new Mobile();
                mobile.setId(6);
                mobile.setModelName("OnePlus Three");
                mobile.setDescription("OnePlus Three HD");
                mobile.setManufacturer("OnePlus");
                mobile.setType("s");
                mobile.setNetworkType("GSM");
                mobile.setAmount(32000);
        try {

             sfactory = HibernateUtil.getSessionFactory();
                session = sfactory.openSession();
        mobile = (Mobile) session.get(Mobile.class, 1);
        System.out.println(mobile);
                        //session.evict(mobile);
                        session.clear();
                        mobile = (Mobile) session.get(Mobile.class, 1);
                        System.out.println(mobile);

                        /*
                         * transaction = session.beginTransaction();
                           session.save(mobile);
                         * System.out.println("mobile saved...");
                           session.flush();
                         * System.out.println("session flushed...");
                         */flag = false;
                        /*
                    * mobile = (Mobile) session.get(Mobile.class, 1);
                         * System.out.println(mobile);
                         session.close();
                          session =sfactory.openSession();
                           Mobile mobile1 = (Mobile) session.get(Mobile.class, 1);
                          System.out.println(mobile1);
 * System.out .println("mobile 1 == mobile 2 ?: " + (mobile == *
mobile1)); */
} finally {
                        if (transaction != null) {
                            if (flag) {

                                transaction.commit();

                            } else {
                                transaction.rollback();
                        }

                        }
 if (session != null) {

                            session.close();
                    }
                HibernateUtil.closeSessionFactory();

                }
        }
}
```

09-10-2017

seesion.flush()

Whenever we are poerforming DML operations on the database using Hibernate,then the Hibernate will not write changes happened on the Entity object into the database directly.But in some cases we wanted to ensure the Hibernate to persist the changes into the database.Then in such cases flush() will be used to write the current state of the entity object into the database and will never commits.

transaction.commit()

when we use transaction.commit() then it calls internally flush(),all the changes that are there in the entity object that are being wriiten onto the database and finally issues commit on those changes into the database.That menas commit not only flushes the changes it writes the changes to the database permanantly.

Mehtods to control the Cache at FirstLevel:

session.evict(Entity object);

It is a method that is used for removing one object from the session level cache. If we dont want to get the second object from the cache,rather from database we will use evict().

```
mobile = (Mobile) session.get(Mobile.class, 1);
System.out.println(mobile);
session.evict(mobile);
```

session.clear()

if we wanted to clear the all objects with in the cache then we can use session.clear().once this method loads the entire objects loaded within the session from then onwards every access to the entity object will comes from the database only.

```
mobile = (Mobile) session.get(Mobile.class, 1);
System.out.println(mobile);
session.clear();
```

IdentityGenerators

What are Identity Generators?
What is the purpose of Identiy generators?
Which Identity generators you are using as part of your project?
Which Identity generator has to be used at which time?

Every application contains a database to store and manage the data,To establish the relation between different parts of the database or across the tables we relay on PrimaryKeys.Unless and untill a Table contains primary key we can not relate one piece of information with another piece of information by establishing an foreign key. foriegn key has to be created for Primary key of another table.

Is it mandatory to contain PK?

No it is not mandatory.

Note:It is not encouraged to mainatin compositePk as part of the database because it is hard to manage them and they undergo some changes.

Why IdentityGenerators?

Every application and every database table contains the surrogate keys the so Programmer has to write complicated logic to determine and generate these keys uniquely within the system,with which the persistency has to be performed.As it is a tedious job for the developer to write the mapping for generating keys,Hibernate has provided a strategy called Identity Generators.

List of IdentityGenerators:

Hibernate contains the following IdentityGenerators.

1.Assigned
2.Native
3.Increment
4.Identity
5. hilo
6.sequence
7.seqhilo
8.GUID
9.UUID
10.foreign
11.select

- <generator /> is one of main element we are using in the hibernate framework [in the mapping file], let us see the concept behind this generators.
  Up to now in our hibernate mapping file, we used to write <generator /> in the id element scope, actually this is default like whether you write this assigned generator or not hibernate will takes automatically
- In fact this assigned means hibernate will understand that, while saving any object hibernate is not responsible to create any primary key value for the current inserting object, user has to take the response
- The thing is, while saving an object into the database, the generator informs to the hibernate that, how the primary key value for the new record is going to generate
- hibernate using different primary key generator algorithms, for each algorithm internally a class is created by hibernate for its implementation.
- hibernate provided different primary key generator classes and all these classes are implemented from org.hibernate.id.IdentifierGeneratar Interface

- while configuring <generator /> element in mapping file, we need to pass parameters if that generator class need any parameters, actually one sub element of <generator /> element is <param />, will talk more about this

10-07-2016(No Class)&11-07-2016(Missed class)
12-07-2016

1.Assigned:

✓This is the default generator class used by the hibernate, if we do not specify <generator –> element under id element then hibernate by default assumes it as "assigned"

✓If generator class is assigned, then the programmer is responsible for assigning the primary key value to object which is going to save into the database.

✓when we have not specified any Generator then the default Generator is "assigned" which means when we configure the IdentityGenerator as assigned then Hibernate will not generates the PK value for that attribute of the Entity Class during the persistency,rather programmer has to assign it manually.
If we dont want "assigned "Id Generator and if we want to have our own Idgenerator,for this within the Id we have one child tag called

<generator class="com.ig.generators.PolicyIDGenerator">
<u>Compatibility</u>:
✓It works across all the databases.
2.Increment:
✓It queries the max Id of the value table and increments it by one. During the first time while persisting the related entity of that table it queries and holds the value at sessionFactory level and for the successive times it will increment the value at SeesonFactory.
✓This generator is used for generating the id value for the new record by using the formulaMax of id value in Database + 1
✓if we manually assigned the value for primary key for an object, then hibernate doesn't considers that value and uses **max value of id in database + 1** concept only 😃
If there is no record initially in the database, then for the first time this will saves primary key value as 1, as…
✓max of id value in database + 1
0 + 1
result -> 1
<u>advantages</u>:
✓It works across all the databases.
<u>drawbacks</u>:
✓It is suitable for non-clustered Environment only.
✓It works only for Int short or long datatypes.
3.Identity:
Neither the programmer nor the Hibernate is going to generate the Identity or Id value,rather the Underlying database will generates the Id for the PK coulmn.
✓It works only for Int short or long datatypes because those only can be increased.
✓This identity generator doesn't needs any parameters to pass
✓ This identity generator is similar to increment generator, but the difference was increment generator is database independent and hibernate uses a select operation for selecting max of id before inserting new record
✓ But in case of identity, no select operation will be generated in order to insert an id value for new record by the hibernate
<u>Drawbacks</u>:
✓It is specfic to database,it will not be portable.
✓It will not works for oracle.

Compatiability:
✓It works in few databases like MySql and MsSql Server.
Environment:
✓It works all platforms including multithreaded and is suitable for clustered and non-clustered Environment as well,because the key is generated at database level.
4.native:
✓It generates the Id Depends on the database over which we works.
✓In case of MsSqlServer it acts as Identity generator.
✓In case of oracle it acts as Sequence.
✓when we use this generator class, it first checks whether the database supports identity or not, if not checks for sequence and if not, then hilo will be used finally the order will be.. identity,sequence,hilo.
✓It supports int,short and long datatypes.
Drawbacks:
✓To determine how the Id is generated it again depends on database.
Compatability:
✓ As it chooses the strategy of generating the Id depends on the database which seems to be compatiable arcross all databases.
5.Sequence
✓It queries the sequence to get the next value with which the entity value will be persisited. while inserting a new record in a database, hibernate gets next value from the sequence under assigns that value for the new record
✓If programmer has created a sequence in the database then that sequence name should be passed as the generator as

```
id name="productId" column="pid">
<generator>
<param name="sequence">MySequence</param>
</genetator></id>
```

✓If the programmer has not passed any sequence name, then hibernate creates its own sequence with name “**Hibernate-Sequence**” and gets next value from that sequence, and then assigns that id value for new record.
✓But remember, if hibernate want’s to create its own sequence, in hibernate configuration file, **hbm2ddl.auto** property must be set enabled

Drawbacks:
✓It works with only Oracle.
Compatiabilitiy
✓It will not compatiable across all the database.
Environment:
✓It works across all the Environments.

13-07-2016

6.hilo

✓It is designed based on high and low algorithm to generate the id of type short, int and long.
✓It works on mathematical formula (max_lo*max_high)+next_high.
✓while persisting batch of records to avoid re-computing the Id for every entity by going to the database hilo can generate batch of Id's at one shot.

✓<id name="Id" column="IdCoulmn">
<generator class="hilo">
<param name="table">your table name</param>(optional)
<param name="column">your column name </param>(optional)
</generator>
</id>

✓In case if we didnt specify any Table name then bydefault it creates Hibernate unique Key with coulmn as next_high and defaults to value as '0'.
Compataibility:
✓It works with all the databases.
Environments:
✓It works only in Global Transactions.

7.Seqhilo

✓It works same as hilo but instead of getting the next_high value from table,it fetches the value from Sequence which we have specified.

✓<id name="Id" column="IdCoulmn">
<generator class="hilo">
<param name="sequence">next_high_Sequence</param>(mandatory)
<param name="max_low">bagsize </param>(optional)</generator></id>

✓It uses high and low algorithm on the specified sequence name.
✓It returns the Id of Int,short,long datatypes.
Environments:
✓It works in all the Environments.
Drawbacks:
✓It is database speciifc,it works with oracle only.

14-07-2016

8.UUID('**Universally Unique Identifier**')

✓The UUID is represented in hexadecimal digits, 32 in length.

✓It uses 128-bit UUID algorithm to generate the id,computed by the application.

✓The returned id is of type String, unique within a network (because IP is used).

Drawbacks:

✓The generated key is not human readable.

✓It can not be useful for the enduser to interact with the system.

✓Huge amount of storage space will be consumed.

Compatibility:

✓It works with all the databases.

Environments:

✓It is suggestable to use across all the environments where the systems can keep track of the generated Id.

9.GUID(**Globally Unique Identifier**')

✓The GUID is represented in hexadecimal digits, 32 in length.

✓It uses 128-bit UUID algorithm to generate the id,computed by the database of type string.

Drawbacks:

✓The generated key is not human readable

✓It can not be useful for the enduser to interact with the system.

✓Huge amount of storage space will be consumed.

✓It is database speciifc.

Compatability:

✓It works on MS SQL Server and MySQL.

Environments:

✓It works seamlessly across all the Environments like clustered,non-clustered and multithreaded environments.

10.foreign

we will see about this generator in one-to-one relationship, else you may not understand.

11.select

✓It works with legacy applications.

✓If the Id is being generated by a database trigger which we wanted to use it as part of Hibernate then we can use Hibernate.

✓When the Database trigger has generated the Id and persisted the record

Then The Select generator in Hibernate goes to the Database and fetches the Id got Generated with which record is persisted by querying it using candiadate key.

```
✓<id name="Id" column="IdCoulmn">
<generator class=”select”>
<param name="key">
<coulmn name=”policy property name”</generator>
</id>
```

Compatibility:

✓It works with all the databases.

Advantage:

✓It is used for fecilitating the migration efforts rather than see it as a drawback because it bridges the gap between the legacy application and database.

Environments:

✓It works seamlessly across all the Environments like clustered,non-clustered and multithreaded environments.

CompleteExample:

15-07-2016

ID**G**enerators Using HibernateJPA Annotations

Hibernate has added the support for working with annotation from Hibernate 3.0.

From the Hibernate 3.0,it is being served in two ways as follows...

[1]Hibernate Core

[2]JPA Api

So we will have two ways of working with IdGenerators using two sets of annotations,one is provided by Hibernate and the other one by JPA,both resembles the same(same annotations).

JPA provided IDGenerators:

We have FOUR identity Generators provided by JPA and are..

1.auto

2.Sequence

3.Table

4.Identity

working with Sequence.

Lets say we have an class with some attributes and accessor methods. If we wanted to make this class as Entity class then we need to write the corresponding mapping file or we can annotate with JPA Annotation to make that class as Entity class and we can persistable.

Example:
Generation of primary key values is a very important functionality of relational database management systems. The main idea is to let RDBMS automatically calculate and assign primary key value to the row being inserted into the database table. This not only simplifies the source code of the application using database but also makes the application more robust.JPA provide a way to automatically generate primary key values using"sequence" .
Oracle Database and PostgreSQL use explicit sequence type to generate unique primary key numbers. For example, in Oracle Database the table and its sequence can be defined like this:

```
CREATE TABLE JPAGEN_ADDRESS
(
   ID NUMBER PRIMARY KEY,
   CITY VARCHAR(255) NOT NULL,
   STREET VARCHAR(255) NOT NULL
);
CREATE SEQUENCE JPAGEN_ADDRESS_SEQ START WITH 100;
```

When using sequence for given entity, the *SEQUENCE* strategy must be defined on its ID column and additionally the name of the sequence must be specified using *@SequenceGenerator* annotation:

```
@Entity
@Table(name = "JPAGEN_ADDRESS")
public class Address implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE,
         generator = "addressGen")
   @SequenceGenerator(name = "addressGen",
         sequenceName = "JPAGEN_ADDRESS_SEQ")
   private long id;
 // other fields and methods are omitted.
}
```

When a new *Address* entity is persisted, JPA implementation will obtain the next value of the sequence and use it to insert a new row into the database table.
Working Example On Sequence:

```
11.AnnotationIdGenerators
|-src
  |-com.aig.entites
    |-plan.java
  |-com.aig.util
      |-HiberanateUtil.java
  |-com.aig.test
     |-AIGTest.java
|-hibernate.cfg.xml
```

Lets say we have an Entity class plan as folows..

```
@Entity
@Table(name = "MOBILE_PLANS")
public class Plan {
@Id
@Column(name = "PLAN_ID")
@GeneratedValue(strategy = GenerationType.SEQUENCE,

          generator = " planIdSequence ")
//Sequence can not be configured at either attribute level.So the
SequenceName will be providing as reference to use it multiple times.
@SequenceGenerator(name = "PlanIdSequence",
          sequenceName = "Plan_Id_Generator_Seq")
private int id;
     @Column(name = "PLAN_NM", nullable = false, length = 50)
private String planName;
      @Column(name = "PLAN_TYPE", nullable = false, length = 50)
private String type;
      @Column(name = "DESCR", nullable = true, length = 500)
private String description;
private float amount;
@Column(name = "VALIDITY_IN_DAYS", nullable = false, length = 3)
private int validityInDays;
//setters and getters.
}
```

We have Both the Entity class and Mapping file next we need configuration file as hibernate.cfg.xml as follows..

```
<hibernate-configuration>
      <session-factory>
            <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
            <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
            <property
name="connection.username">hib_tools</property>
            <property
name="connection.password">welcome1</property>
```

```
              <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty> -->
             <property name="show_sql">true</property>
              <property name="hbm2ddl.auto">update</property>
              <mapping class="com.aig.entities.Plan" />
       </session-factory>
</hibernate-configuration
```

After this we need to create HibernateUtil class as follows..

```
public class HibernateUtil {
       private static SessionFactory sessionFactory;
static {
       Configuration configuration = new Configuration().configure();
       StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
              builder.applySettings(configuration.getProperties());
              StandardServiceRegistry registry = builder.build();
              sessionFactory =
configuration.buildSessionFactory(registry);

       }

       public static SessionFactory getSessionFactory() {
              return sessionFactory;

       }
       public static void closeSessionFactory() {
       if (sessionFactory != null) {
              sessionFactory.close();
                }
       }
}
```

Now we can write the Test class,but before that we need to create the PlanIdSequence in the database as

```
CREATE SEQUENCE Plan_Id_Generator_Seq  START WITH 100 and
INCREMENTED BY 1;
public class AIGTest {
    public static void main(String[] args) {
              SessionFactory sessionFactory = null;
              Transaction transaction = null;
              Session session = null;
              Plan plan = null;
```

```
                boolean flag = false;
        try {
            plan = new Plan();
//Id will be set by the generator.
            plan.setPlanName("Night116CPlan");
            plan.setType("prepaid");
            plan.setDescription("Client Plan");
            plan.setAmount(99f);
            plan.setValidityInDays(30);

                    sessionFactory = HibernateUtil.getSessionFactory();
                    session = sessionFactory.openSession();
                    transaction = session.beginTransaction();
                    session.save(plan);
                    System.out.println("plan id : " + plan.getId());
            flag = true;
                } finally {
                if (transaction != null) {
                    if (flag) {
                            transaction.commit();
                    } else {
                        transaction.rollback();
                }
                  }
                HibernateUtil.closeSessionFactory();
                }
            }
}
```

16-07-2016

2.working with **table** gnerator:

Example:

The third way to generate primary keys is to have a separate **table** which stores in a single row the sequence name along with the next number to use for the primary key. For performance reasons the next value is not increased by one, whenever JPA implementation needs a new value for the primary key but by a much higher number (e.g. 50). Once the range of values becomes reserved, JPA implementation can assign primary keys without accessing the database. When every value in the range becomes used,JPA implementation reserves a new range and the cycle continues.

Additionally, in such table we can have multiple rows with each row serving different entity. The only requirement is to use unique sequence names for each entity.

The table strategy is the most complicated one but it is the only strategy that is really portable across different databases. If you are developing an application which can use multiple RDBMS or there may be a need to port to a new RDBMS in the future, using table strategy is the most viable option.
The table to store the ***Person*** entity and the table to generate sequences can be created like this:

```
→CREATE TABLE JPAGEN_PERSON

(
   ID NUMBER PRIMARY KEY,
   NAME VARCHAR(255) NOT NULL,
   ADDRESS_ID NUMBER
);
→CREATE TABLE JPAGEN_GENERATORS
(
   NAME VARCHAR(255) PRIMARY KEY,
   VALUE NUMBER

);
```

The table with sequences must have two columns. The first one should contain the name of the sequence and the second one should be of numeric type. JPA implementation will automatically update the second column when reserving a new range of values.
Additionally, JPA must be instructed to use *TABLE* generation strategy:

```
@Entity
@Table(name = "JPAGEN_PERSON")
public class Person implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.TABLE,
         generator = "personGen")
   @TableGenerator(name = "personGen",
         table = "JPAGEN_GENERATORS",
         pkColumnName = "NAME",
         pkColumnValue = "JPAGEN_PERSON_GEN",
         valueColumnName = "VALUE")
   private long id;
  // other fields and methods are omitted
}
```

We also have to add *@TableGenerator* annotation to inform JPA about the name of the generator table (*table* element), the names of its both columns (*pkColumnName* and *valueColumnName* elements) and also the name of the sequence (*pkColumnValue* element).

Advantages:

It works across all the databases.

drawbacks:

✓It is not recommneded to use unless and untill we are working with local Transactions.

✓It will not works with all the Datatypes.

Working Example

step1.Lets create a Table with the Name "GENERATED_KEYS" in the database.

step2.Writing Entity class and remaing are same.

```
@Entity
@Table(name = "MOBILE_PLANS")
public class Plan {
@Id
@Column(name = "PLAN_ID")
@GeneratedValue(strategy = GenerationType.TABLE,
          generator = "Plan_Id-generator")
    @TableGenerator(name = "Plan_Id_Generator",
          table = "GENERATED_KEYS",
          pkColumnName = "KeyCoumnName",
          pkColumnValue = "Plan_Id_Generator",

          valueColumnName = "VALUE")
private int id;
      @Column(name = "PLAN_NM", nullable = false, length = 50)
private String planName;
       @Column(name = "PLAN_TYPE", nullable = false, length = 50)
private String type;
       @Column(name = "DESCR", nullable = true, length = 500)
private String description;
private float amount;
@Column(name = "VALIDITY_IN_DAYS", nullable = false, length = 3)
private int validityInDays;
//setters and getters.
}
```

3.Working with Auto-Increment/**Identity**

Example:

MySQL and Microsoft SQL Server provide a functionality to automatically generate a unique number for the primary key when a new row is added into a table. For example, in MySQL it is done by marking primary key column with *AUTO_INCREMENT* keyword:

```
1CREATE TABLE JPAGEN_ADDRESS
2(
3  ID INT PRIMARY KEY AUTO_INCREMENT,
4  CITY VARCHAR(255) NOT NULL,
5  STREET VARCHAR(255) NOT NULL
6);
```

and in Microsoft SQL Server with *IDENTITY* keyword:

```
CREATE TABLE JPAGEN_ADDRESS
(
   ID INT IDENTITY(1,1) PRIMARY KEY,
   CITY VARCHAR(255) NOT NULL,
   STREET VARCHAR(255) NOT NULL
);
```

JPA must be informed to use auto-increment/identity for primary key by specifying *IDENTITY* generation strategy on the ID column:

```
@Entity
@Table(name = "JPAGEN_ADDRESS")
public class Address implements Serializable {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private long id;

   // other fields and methods are omitted
}
```

Now, when a new *Address* entity is persisted, the value of the primary key column will be automatically generated by the database.

Working With Hibernate Id-generators Using Annotations.

The plan Entity class will looks as below..

```
@Entity
@Table(name = "MOBILE_PLANS")
public class Plan {
        @Id
        @Column(name = "PLAN_ID")
        @GenericGenerator(name = "hilo_generator", strategy = "hilo",
parameters = { @Parameter(name = "max_lo", value = "10") })
        @GeneratedValue(generator = "hilo_generator")
 private int id;
//ommit the rest of the attributes for clarity.
}
```

Similarly for all the id generators the synatax will be as follows..
@GenericGenerator(name = "generator name", strategy = "hibernate generator" parameters={@Parameter(name = "-", value = "-") })

Custom IdGenerators

If the IdGenerators provided by both the Hibernate and JPA fails to meet our Requirements then we can create our Own IdGenerator which are nothing but CustomIdentity Generators.
→Lets say we have an Plan Entity class and for this plan we wanted to generate the planId as "P followed by month and year"(P-07-2016)for every plan which is being entered into the database.

Working Example:
The Entity class will looks like as below…

```
public class Policy {
        private String id;
        private String policyName;
        private String description;
        private String type;
        private int minTenure;
        private int maxTenure;
        private int eligibleAge;
//Setters and getters
}
```

The mapping will will be as below..

```
<hibernate-mapping package="com.ig.entities">
          <class name="Policy" table="POLICY">
     <id name="id" column="POLICY_ID" length="32">
  <generator class="com.ig.generators.PolicyIDGenerator">
            <param name="prefix">P</param>
                  </generator>
              </id>
<property name="policyName" column="POLICY_NM" length="100"
           not-null="true" />
<property name="description" column="DESCR" length="500"
       not-null="false" />
<property name="type" column="POLICY_TYPE" length="20"
       not-null="true" />
<property name="minTenure" column="MIN_TENURE" length="3"
        not-null="true" />
<property name="maxTenure" column="MAX_TENURE" length="3"
        not-null="true" />
<property name="eligibleAge" column="MAX_ELIGIBLE_AGE"
        length="3" not-null="true" />
        </class>
</hibernate-mapping>
```

After that we need to Writing our own IdGenerator class,and our class should implement from Interfaces like IdentifierGenerator and if our Id has to be perform some more operations then our class should be implemented from Configurable interface.

```
public class PolicyIDGenerator implements IdentifierGenerator,
Configurable {
        private Properties props;
//Here parameters will be passed as properties
@Override
public void configure(Type type, Properties props, Dialect dialect
        throws MappingException {
        this.props = props;
        }
//to get the connection from Session we will pass the implementor
class of Session object and we will also pass the Object into for whoom
we wanted  to generate the Idand it should return the policyId.
@Override
public Serializable generate(SessionImplementor session, Object obj)
throws HibernateException {
              int month = 0;
              int year = 0;
              int policySeq = 0;
String prefix = null;
String policyId = null;
//we need to get the policySeq from database so we need connection.
Connection con = null;
Calendar calendar = null;
//we need to create the calendar object.
calendar = Calendar.getInstance();
//month start from 0 so we will add 1.
month = calendar.get(Calendar.MONTH) + 1;
year = calendar.get(Calendar.YEAR);
prefix = props.getProperty("prefix");
try {
con = session.getJdbcConnectionAccess().obtainConnection();
//once we get the statement then we need to create the Statemnt
       stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery("SELECT
POLICY_ID_SEQUENCE.NEXTVAL FROM DUAL");
if (rs.next()) {
               policySeq = rs.getInt(1);
        }
policyId = prefix + policySeq + "-" + month + "-" + year;
    } catch (SQLException e) {
   e.printStackTrace();

            }
     return policyId;
             }
}
```

17-07-2016(No Class)& 18-07-2016

Xml Based CompositePrimaryKey:

If we have two columns as part of the PK of a Table then that Table has Composite Primarykey.But we should avoid most of the times to create PrimaryKey because never allow business data to be key coulmn.

Even though Hibernate will not have directly encouraging to work with Composite PrimaryKeys but for the sake ensuring the application that is being already designed and if it is trying to migrate then Hibernate has to provide a mechanism of handling Composite PrimaryKey as part of the application.

Why we have no IdGenerators for CompositePrimaryKey?

Because The CompositePrimaryKeys represents Business data and those are not surrogate columns to generate.

Working Example:

```
CompositeKeys
|-src
    |-com.cpk.entities
        |-RationCard.java
        |-RationCardPk.java
        |-RationCard.hbm.xml
   |-com.cpk.util
       |-HibernateUtil.java
   |-com.cpk.test
       |-CPKTest.java
|-hiberanate.cfg.xml
```

While working with Comosite primaryKeys we should identify the key cloumns that are contributing to the primary key and we should separate them and we need to create a separate class by declaring them as attributes.

Lets say we have RationCard Entity class with the two PK columns rationCardNo and wardNo.So we need to create a separate class for these two attributes and this class should implements from Serializable interface.

RationCardPK.java

```
public class RationCardPK implements Serializable {
private int rationCardNo;
private int wardNo;
//setters and getters
}
```

we have one more class with Non PK columns and will be as below...

RationCard.java

```
public class RationCard implements Serializable {
        private String cardHolderName;
        private String gender;
        private Date dateOfBirth;
//setters and getters
}
```

For these Two Entity class the mapping file will looks as follows.

RationCard.hbm.xml

```
<hibernate-mapping package="com.ck.entities">
   <class name="RationCard" table="RATION_CARD">
        <composite-id name="id">
<key-property name="rationCardNo" column="RATION_CARD_NO" />
<key-property name="wardNo" column="WARD_NO" />
        </composite-id>
<property name="cardHolderName" column="CARD_HOLDER_NM" />
<property name="gender" column="GENDER" />
<property name="dateOfBirth" column="DATE_OF_BIRTH" />
</class>
</hibernate-mapping>
```

Once we have Entity classes and mapping files next we need configuration files.

hibernate.cfg.xml:

```
<hibernate-configuration>
<session-factory>
<property
name=”connection.driver_class”>oracle.jdbc.driver.OracleDriver</prop
erty>
<property
name=”connection.url”>jdbc:oracle:thin:@localhost:1521:xe</property
>
<property name=’connection.username”>hib_tools</property>
<property name=”connection.password”>welcome1</property>
<property
```

```
name=”hibernate.dialect”>org.hibernate.dialect.Oracle10gDialect</pro
perty>
<property name=”show_sql”>true</property>
<property name=”hbm2ddl.auto”>update</property>
<mapping resource=”com/ck/entities.RationCard.hbm.xml”/>
<!--<mapping class=”com.ck.entites.RationCard/>-->
</session-factory>
</hibernate-configuration>
```

And the HibernateUtil class will be as follows..

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);

        }

        public static SessionFactory getSessionFactory() {
                return sessionFactory;

        }
        public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                   }
        }
 }
```

Now lets try to persist the entity into database

CKTest.java

```
public class CKTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                RationCard card = null;
                boolean flag = false;
```

```
     try {
                card = new RationCard();
                        card.setRationCardNo(1);
                        card.setWardNo(34);
                        card.setCardHolderName("John");
                        card.setGender("Male");
                        card.setDateOfBirth(new Date());
sessionFactory = HibernateUtil.getSessionFactory();
session = sessionFactory.openSession();
transaction = session.beginTransaction();
               session.save(card);
} finally {
        if (transaction != null) {
                           if (flag) {
                          transaction.commit();
      } else {
                           transaction.rollback();
                             }
                  }
        if (session != null) {
               session.close();
               }
       HibernateUtil.closeSessionFactory();
           }
     }
}
```

Fetching the data from the RationCard Table.

For this we need to create one more RationCard class object.

```
RationCardPK pk = new RationCardPK();
                        pk.setRationCardNo(1);
                        pk.setWardNo(34);
card = (RationCard) session.get(RationCard.class, pk);
System.out.println("name : "+ card.getCardHolderName());
```

19-07-2016

Annotation Based CompositePrimaryKey:

RationCard.java

```
@Entity
@Table(name = "RATION_CARD")
public class RationCard implements Serializable {
//As this class is embeded in other class.
@EmbeddedId
        private RationCardPK id;
        @Column(name = "CARD_HOLDER_NM")
        private String cardHolderName;
        @Column(name = "GENDER")
        private String gender;
        @Column(name = "DATE_OF_BIRTH")
        private Date dateOfBirth;
//setters and getters
}
```

RationCardPk.java

```
@Embeddablepublic class RationCardPK implements Serializable
{@Column(name = "RATION_CARD_NO")
private int rationCardNo;
@Column(name = "WARD_NO")
private int wardNo;
//setters and getters
}
```

Note:Hibernate.cfg.xml is same without mapping resources and test class and util classes are same.

save(),persist(),saveOrUpdate()

Lets say we have an Entity class with the name "person" as below..

```
public class Person {
                   private int id;
                   private String firstName;
                   private String lastName;
                   private String gender;
                   private int age;
//Setters and Getters
}
```

After Mapping this Entity class If we wanted to save this Entity Object into the database we have a method called **Session.save()** which is provided by the Hibernate as part of the Session Interface.save() is useful to create a new entity object into the database.Apart from save()

we have one more method called **persist()** to creates a new Entity object into the database.

Difference between save() and persist()

Actually the difference between hibernate save() and persist() methods is depends on generator class we are using.

Session.save()

Whenever we save the person Entity then the save() will not just execute the corresponding SQL Query to store the data into the database,Once The Entity object is stored then save() will fetch the ID that is being generated by the IdGenerator while persisting the data and it quickly reflected the Id into the Entity object and place within the cache before calling the flush() itself by reloading that Entity.

✓return type of save is Serializable object

✓Both methods make a transient instance persistent. However, save() method guarantee that the identifier value will be assigned to the persistent instance immediately.

✓save() method doesnt guarantees that it will not execute an INSERT statement if it is called outside transaction boundaries. it returns an identifier, and if an INSERT has to be executed to get the identifier (e.g. "identity" generator), this INSERT happens immediately, no matter if you are inside or outside of a transaction.

✓save method is not good in a long-running conversation with an extended Session context

Session.persist()

Whenever we call session.persist() then it stores the Entity object into the underlying database but the Id for that particular Entity object will not be generated immediately.It may reflects or may not reflects.Untill and unless we call flush() then only the Id will be generated into the object.

✓return type of persist is void

✓persist() method doesn't guarantee that the identifier value will be assigned to the persistent instance immediately, the assignment might happen at flush time.

✓persist() method guarantees that it will not execute an INSERT statement if it is called outside transaction boundaries.

✓its useful in long-running conversations with an extended Session context

Working Example

```
SaveOrPersistOrSaveUpdate
|-src
   |-com.spt.entities
       |-person.java
       |-person.hbm.xml
 |-com.spt.util
      |-hibernateUtil.java
|-com.spt.test
    |-SPTTest.java
|-hibernate.cfg.xml
```

Person Entity class will be as follows with the annotations.

```
@Entity
@DynamicUpdate(true)
@DynamicInsert(true)
@Table(name = "PERSON")
public class Person {
        @Id
        @Column(name = "PERSON_ID")
@GenericGenerator(name = "hilo_generator", strategy = "hilo",
parameters = { @Parameter(name = "max_lo", value = "10") })
@GeneratedValue(generator = "hilo_generator")
   private int id;
        @Column(name = "FIRST_NM")
    private String firstName;
        @Column(name = "LAST_NM", nullable = true)
    private String lastName;
    private String gender;
    private int age;
//setters and getters
}
```

Hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
                <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</prope
rty>
        <property name="connection.username">root</property>
        <property name="connection.password">root</property>
<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</proper
ty>
```

```
     <property name="show_sql">true</property>
     <property name="hbm2ddl.auto">update</property>
    <!-- <mapping class="com.sp.entities.Person" /> -->
    <mapping resource="com/sp/entities/Person.hbm.xml" />
     </session-factory>
</hibernate-configuration>
```

and the Util class will looks as below..

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
        }

        public static SessionFactory getSessionFactory() {
                return sessionFactory;

        }
        public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
        }
 }
```

Now we can check wether it is working or not?

```
public class SPTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                Person person = null;
                boolean flag = false;
        try {
                sessionFactory = HibernateUtil.getSessionFactory();
                session = sessionFactory.openSession();
                transaction = session.beginTransaction();
        person = new Person();
//if we didnt specify any Id generator then we need to manually
configure Id.
```

```
                    //person.setId(1);
                     person.setFirstName("Rod");
                     person.setLastName("Andrew");
                     person.setGender("Male");
                     person.setAge(10);
                     session.save(person);
                    //session.persist(person);
                    //session.saveOrUpdate(person);
        flag = true;
} finally {
                     if (transaction != null) {
                          if (flag) {
                               System.out.println("commited...");
                     transaction.commit();
                 } else {
                          transaction.rollback();
                     }
                   }
         HibernateUtil.closeSessionFactory();
             }
         }
  }
```

Test Cases

#1-without Id generator.
#2-with id Genrator(by commenting)

Differences between save() and saveOrUpdate()

session.save()

✔save() method saves records into database by INSERT SQL query, Generates a new identifier and return the Serializable identifier back.

session.saveOrUpdate()

✔It works based on one attribute called “unsaved-value="0",this attribte will be declared at id tag level as follows..

```
<id name="id" column="PERSON_ID" unsaved-value="0">
<generator class="increment"/></id>
```

Note:unsaved-value for primitive integer is “0” and for string it is “null”

✔saveOrUpdate() method either INSERT or UPDATE based upon existence of object in database. If persistence object already exists(unsaved-value!=0 in database then UPDATE SQL will execute and if there is no corresponding(unsaved-value=0) object in database than INSERT will run based on.

✔Instead of writing two methods save(),update() we can use saveOrUpdate().

session.delete()

✔To deleting the object(1 row) form the database we need to call delete method in the session.

✔In the hibernate we have only one method to delete an object from the database that is what i have shown you here...

```
Configuration cfg = new Configuration().configure("hibernate.cfg.xml");
SessionFactory factory = cfg.buildSessionFactory();
Session session = factory.openSession();
Object obj=session.load(Person.class,new Integer(103));
 Person p=(person)obj;
Transaction tx = session.beginTransaction();
session.delete(p);
System.out.println("Object Deleted successfully.....!!");
 tx.commit();
session.close();
factory.close();
}
```

When we create the SessionFactory for every Entity object to whoom we provide the mapping information ByDefault Hibernate will generate two queries INSERT and UPDATE to avoid performance issues while performing DML operation.

DynamicUpdate

✔The dynamic-update attribute tells Hibernate whether to include unmodified properties in the SQL UPDATE statement.

✔The default value of dynamic-update is false, which means **include unmodified properties** in the Hibernate's SQL update statement.

✔Hibernate will update all the unmodified columnsIf set the dynamic-insert to true, which means **exclude unmodified properties** in the Hibernate's SQL update statement.

✔In a large table with many columns (legacy design) or contains large data volumes, update some unmodified columns are absolutely unnecessary and great impact on the system performance.

✔**dynamic-update**" tweak will definitely increase your system performance, and highly recommended to do it.

✔we can configure "dynamic-update" properties via annotation or XML mapping file.

Xml Conigiuration:

```
<class name="Person" table="PERSON" dynamic-update="true">
```

Annotation Configuration:

```
@Entity
@DynamicUpdate(true)
@Table(name = "PERSON)
public class Person {
-------------------}
```

Dynamic-Insert

✓The dynamic-insert attribute tells Hibernate whether to include null properties in the SQL INSERT statement.
✓The default value of dynamic-insert is false, which means **include null properties** in the Hibernate's SQL INSERT statement.
✓**Hibernate will generate the unnecessary columns** for the insertion, If set the dynamic-insert to true, which means **exclude null property values** in the Hibernate's SQL INSERT statement then **Hibernate will generate only the necessary columns** for the insertion.
✓**dynamic-insert**" tweak may increase your system performance, and highly recommends to do it
✓we can configure the dynamic-insert properties value through annotation or XML mapping file.

Xml Coniguration:

```
<class name="Person" table="PERSON" dynamic-insert="true">
```

Annotation Configuration:

```
@Entity
@DynamicInsert(true)
@Table(name = "PERSON")
public class Person {
------------------------}
```

session.merge()

Lets say we have an Entity class with the name "AdEnquiry" which is used to provide the informaton about the advertisement.

```
public class AdEnquiry {
        private int id;
        private String description;
        private int duration;
        private String email;
        private String mobile;
        private String place;
        private boolean onTv;
        private boolean internetMedia;
        private boolean newsPaper;
        private boolean hordings;
//setters and getters
}
```

Inorder to make this class persistable we need to write mapping file as follows..

```
<class name="AdEnquiry" table="AD_ENQUIRY">
  <id name="id" column="AD_ENQUIRY_ID" unsaved-value="0">
                        <generator class="increment" />
                </id>
                <property name="description" column="DESCR" />
```

```
<property name="duration" column="DURATION" not-null="false" />
                <property name="email" />
                <property name="mobile" />
                <property name="place" />
<property name="onTv"><column name="ON_TV" default="0" not-
null="false" />
                </property>
<property name="internetMedia"><column name="INTERNET_MEDIA"
default="0" not-null="false" />
                </property>
<property name="newsPaper"><column name="NEWSPAPER"
default="0" not-null="false" />
                </property>
<property name="hordings"><column name="HORDINGS" default="0"
not-null="false" />
                </property>
        </class>
</hibernate-mapping>
```

Lets create AdEnquiry object in the Test class as follows..

```
public class WMTest {
        public static void main(String[] args) {
                SessionFactory sfactory = null;
                Transaction transaction = null;
                Session session = null;
try{
                AdEnquiry enquiry = null;
                boolean flag = false;
        enquiry = new AdEnquiry();
        enquiry.setDescription("Surya Steels");
        enquiry.setDuration(60);
        enquiry.setEmail("surya@gmail.com");
        enquiry.setMobile("13393839384");
        enquiry.setPlace("Ameerpet");
 enquiry.setOnTv(true); enquiry.setInternetMedia(true);
              sfactory = HibernateUtil.getSessionFactory();
              session = sfactory.openSession();
              transaction = session.beginTransaction();
              session.save(enquiry);
} finally {
----------}
```

Now the enquiry will gets persisted with an Id and the Id will be associated with that enquiry.

→Lets create one more Enquiry,if it is first one then it will be persisted with id as=1.

```
AdEnquiry enquiry = new AdEnquiry();
                    enquiry.setId(2);
                    enquiry.setDuration(100);
                    session.udate(enquiry);
```

As we already saved one Enquiry Entity object,immediately while under the same active Session if we tried creating one more Enquiry object and if we call update() then Hibenrate throws **NonUinque-Exception** because the update should not be allowed because the object is alredy associated with the session and the earlier unchanged changes will be lost if we allow the new Entity object to update.

Note:At anytime within the session for the same ID there should be only one Entity object to be allowed,it will not allow to exist multiple Entity objects with the same ID to be binded to the Session Object.

→Instead of the above if do as follows..

```
  AdEnquiry enquiry1 = new AdEnquiry();
                    enquiry1.setId(2);
                    enquiry1.setDuration(100);
                    session.update(enquiry1);
```

Here as we are fetching the same object and we are overriding,so there is no problem But our old data will be overriden by latest copy and Hibernate will not allows us to store two Entity Objects cache witht the same ID.

→Where as when we use merge then the current object will be merged with the associated object that is already there in session.

```
AdEnquiry enquiry1 = new AdEnquiry();
                    enquiry1.setId(2);
                    enquiry1.setDuration(100);
                    session.merge(enquiry1);
```

Note:It is not recommended to use.

CompleteExample:

```
WorkingWithMerge
|-src
   |-com.wm.entities
      |-AdEnquiry.java
      |-AdEnquiry,hbm.xml
   |-com.wm.util
      |-HibernateUtil.java
   |-com.wm.test
      |-WMTest.java
|-hibernate.cfg.xml
```

AdEnquiry.java

```
public class AdEnquiry {
        private int id;
        private String description;
        private int duration;
        private String email;
        private String mobile;
        private String place;
        private boolean onTv;
        private boolean internetMedia;
        private boolean newsPaper;
        private boolean hordings;
//setters and getters
}
```

AdEnquiry.hbm.xml

```
<hibernate-mapping>
<class name="AdEnquiry" table="AD_ENQUIRY">
  <id name="id" column="AD_ENQUIRY_ID" unsaved-value="0">
                        <generator class="increment" />
                </id>
                <property name="description" column="DESCR" />
<property name="duration" column="DURATION" not-null="false" />
                <property name="email" />
                <property name="mobile" />
                <property name="place" />
<property name="onTv"><column name="ON_TV" default="0" not-
null="false" />
                </property>
<property name="internetMedia"><column name="INTERNET_MEDIA"
default="0" not-null="false" />
                </property>
<property name="newsPaper"><column name="NEWSPAPER"
default="0" not-null="false" />
                </property>
<property name="hordings"><column name="HORDINGS" default="0"
not-null="false" />
                </property>
        </class>
</hibernate-mapping>
```

Hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
                <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
```

```
		<property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</prope
rty>
	<property name="connection.username">root</property>
	<property name="connection.password">root</property>
<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</proper
ty>
    <property name="show_sql">true</property>
    <property name="hbm2ddl.auto">update</property>
<mapping resource="com/wm/entities/AdEnquiry.hbm.xml" />
    </session-factory>
</hibernate-configuration>
```

and the Util class will looks as below..

```
public class **HibernateUtil** {
	private static SessionFactory sessionFactory;
static {
	Configuration configuration = new Configuration().configure();
	StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
		builder.applySettings(configuration.getProperties());
		StandardServiceRegistry registry = builder.build();
		sessionFactory =
configuration.buildSessionFactory(registry);
	}

	public static SessionFactory getSessionFactory() {
		return sessionFactory;

	}
	public static void closeSessionFactory() {
	if (sessionFactory != null) {
		sessionFactory.close();
		  }
	}
}
```

WMTest.java

```
public class WMTest {
	public static void main(String[] args) {
		SessionFactory sfactory = null;
		Transaction transaction = null;
		Session session = null;
		AdEnquiry enquiry = null;
		boolean flag = false;
try{
       /* enquiry = new AdEnquiry();
        enquiry.setDescription("Surya Steels");
        enquiry.setDuration(60);
        enquiry.setEmail("surya@gmail.com");
        enquiry.setMobile("13393839384");
        enquiry.setPlace("Ameerpet");
 enquiry.setOnTv(true);
 enquiry.setInternetMedia(true);
*/
              sfactory = HibernateUtil.getSessionFactory();
              session = sfactory.openSession();
              transaction = session.beginTransaction();
              /*session.save(enquiry); */
/*
  * AdEnquiry enquiry1 = new AdEnquiry();
  * enquiry1.setId(2);
  * enquiry1.setDuration(100);
    session.merge(enquiry1);
 */
        enquiry = (AdEnquiry) session.get(AdEnquiry.class, 1);
        enquiry.setDescription("Surya Teja Steels");
        enquiry.setDuration(360);
        flag = true;
} finally {
                    if (transaction != null) {
                        if (flag) {
                            System.out.println("commited...");
                    transaction.commit();
                } else {
                        transaction.rollback();
                    }
                }
          HibernateUtil.closeSessionFactory();
             }
        }
  }
```

Difference Between update() and merge()

Both update() and merge() methods in hibernate are used to convert the object which is in detached state into persistence state. But there is little difference. Let us see which method will be used in what situation.

```
1  ------
2  -----
3  SessionFactory factory = cfg.buildSessionFactory();
4  Session session1 = factory.openSession();
5
6  Student s1 = null;
7  Object o = session1.get(Student.class, new Integer(101));
8  s1 = (Student)o;
9  session1.close();
10
11 s1.setMarks(97);
12
13 Session session2 = factory.openSession();
14 Student s2 = null;
15 Object o1 = session2.get(Student.class, new Integer(101));
16 s2 = (Student)o1;
17 Transaction tx=session2.beginTransaction();
18
19 session2.merge(s1);
```

Explanation

- See from line numbers 6 – 9, we just loaded one object s1 into session1 cache and closed session1 at line number 9, so now object s1 in the session1 cache will be destroyed as session1 cache will expires when ever we say session1.close()
- Now s1 object will be in some RAM location, not in the session1 cache
- here s1 is in detached state, and at line number 11 we modified that detached object s1, now if we call update() method then hibernate will throws an error, because we can update the object in the session only
- So we opened another session [session2] at line number 13, and again loaded the same student object from the database, but with name s2

- so in this session2, we called **session2.merge(s1)**; now into s2 object s1 changes will be merged and saved into the database

Hope you are clear…, actually update and merge methods will come into picture when ever we loaded the same object again and again into the database, like above.

22-07-2016

Hibernate Entity Object LifeCycle

Hibernate prefers your objects to be in a certain "state", and there are four different states that exist inside of the hibernate life cycle. They are-

1.Transient
2.Persistent
3.Detached
4.Removed

1.Transient:

✓When ever an object of a pojo class is Created(instantiated) using the new operator then it will be in the Transient state; this object is not associated with any Hibernate Session.

✓This object don't have any association with any database table row. In other words any modification in data of transient state object doesn't have any impact on the database table.

✓Object is neither associated with Session nor present in Database.

Example:

EMP ID	EMP NAME	EMP SALARY
1001	prasad	50000
1002	anusha	40000

```
Employee emp = new Employee();
emp.setEmpId(1003);
emp.setEmpName("Sreenath");
emp.setEmpSalaray("30000");
```

✓Here **emp** object is not associated with session and there is no matching record in the Employee table. So **emp** object is in **TRANSIENT**

✓In this state, object is non-transactional. That means object is not synchronized with table record in database.

✓In this state, changes made to Objects don't save into the database.

2.Persistent

✓when the object is in persistent state means it represents one row of database.

✓It is associated with the Unique Session.

✓Hibernate will detect any changes made to an object in persistent state and synchronze the with the database when the unit of work completes.

✓We can create Persistent objects via **TWO** ways.

ways to save an object:

1.save() 2.saveOrUpdate() 3.persist()

ways to load an object:

1.get() 2.load() 3.list()

EMP ID	EMP NAME	EMP SALARY
1001	prasad	50000
1002	anusha	40000

```
Employee emp = (Employee) session.get(Employee.class,1001);
```

✓Here **emp** object is associated with session and there is matching record in **EMPLOYEE.** So **emp** object is in **PERSISTENET** state.

✓Here object is transactional. That means object is synchronized with table record in database. Here modifications which are done to the entity, doesn't save into the database.

✓Changes made to objects are automatically saved into the database without invoking session persistence methods.

3.Detached

✓If we wanted to move an object from persistent to detached then we need to cloase the session or clear the cache of the session.

✓Here the reference to the object is still valid and detached instance can be modified in this state.

✓A detached instance can be re-attached to a new session at later point in time,making it(and all modifications)persist again.

✓The re-attaching of object from detached state to persistent state can be done by calling following methods.

1.update() 2.merge() 3.saveOrUpdate() 4.look().

✓
```
Employee emp = new Employee();
    emp.setEmpId(1003);
    emp.setEmpName("Sreenath");
    emp.setEmpSalaray("30000");
  //Here emp object is in Transient state.
```

```
 session.save(emp);
//Here emp object is in Persistent state.
 session.close();
//Here emp object is in Detached state.
```

✓Here **emp** object becomes Detached state from persistent state after calling session.close().

✓ Changes made to object will not be stored into database, Since session is closed (garbage collected)

✓ Here object is non-transactional, that means object is not synch with database. So changes made to detached objects are not saved into the database.

4.Removed

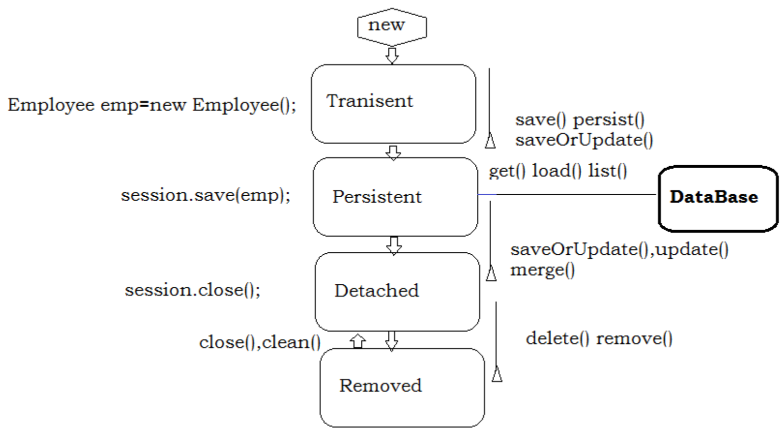
✓A persistent object is considered to be Removed state when a delete() or remove() operation is called on it.

✓Once we have deleted an object and moved to the "removed "state then we should no longer use that particular object for any reason.

Conclusion:

Lifecycle/State	Is object associated with the session?	Is object present in database?
Transient	NO	NO
Persistent	YES	YES
Detached	NO	YES

The following diagram depicts how Hibernate life cycle looks like..



Hibernate Entity Object Life Cycle

23-07-2016

Contextual Sessions

24-07-2016(No class)& 25-07-2016
26-07-2016(No class)& 27-07-2016
28-07-2016(No class)& 29-07-2016

What are contextual Session?
Why we need Contextual Session?
when we are working within an application,whlile we are managaing the transactions across all the classes,we wanted to share the same session object for the entire business operations which falls under one single transactions.In such cases usually we manully pass the same session object between all the Dao classes within our application but it is not a recommended solution to follow.

As part of the Business Operation that we wanted to perform there could be multiple classes,which we need to bring all such classes under one single transaction .In such cases we should be able to share the same session object to multiple classes which are participating under the same business operation.

As part of the Hibernate 3.0,it has provided the support for working with Contextual Session.Hibernate has provided three dfferent levels at which we can manage the session object.
Values for the **current_session_context_class** will be,

1. thread(local transaction)
2. jta(global transaction)
3. managed

1. Thread

✓session bound to a thread, for single thread execution if you call a getCurrentSession() then it will return you a same session object.
✓For every Thread of execution the Hibernate SesionFactory is going to create one Session object.
✓To tell the Hibernate to manage the sessionFactory at **thread level** we need to call **sessionFactory.getCurrentSession()** whenever we call this method then sessionFactory will goes to the **hibernate.cfg.xml** and tries to find the CurrentSession Context class(Thread),then it will not creates new Session object rather it goes to the ThreadLocalContextSession class and ask him the existence of session.If the session is not there then it wil ask the SessionFactoyImplementor to create a new session and then stores in

the ThreadLocal and returns the session.

✓Session is a light weight and a non-threadsafe object (No, you cannot share it between threads) that represents a single unit-of-work with the database. Sessions are opened by a SessionFactory and then are closed when all work is complete. Session is the primary interface for the persistence service. A session obtains a database connection lazily (i.e. only when required). To avoid creating too many sessions ThreadLocal class can be used as shown below to get the current session no matter how many times you make call to the currentSession() method.

```
public class HibernateUtil {
public static final ThreadLocal local = new ThreadLocal();
public static Session currentSession() throws HibernateException

{
        Session session = (Session) local.get();
        //open a new session if this thread has no session
        if(session == null)
        {
session = sessionFactory.openSession();
local.set(session);
        }
        return session;
        }
}
```

2. managed - this provided for managing session from some external entity, something like intercepter. maintaining session in a ThreadLocal same as a "thread" type, but having a different methods like bind(), unbind(), hasbind() to be used by external entity to maintain session in a context.

3. jta - session will be bound to a transaction, as we says a transaction then we need application server for using this context class.

Difference Between openSession() and getCurrentSession()

In **hibernate**, there are **two ways** to get the org.hibernate.**Session** object.

1. **Using openSession() method**
2. **Using getCurrentSession() method**

1. openSession():

Session openSession( ) throws HibernateException

It will return a new session object on every call, which is actually a instance of **org.hibernate.impl.SessionImpl**.

```
private SessionImpl openSession(
   Connection connection,
       boolean autoClose,
       long timestamp,
       Interceptor sessionLocalInterceptor
   ) {
   return new SessionImpl(
      connection,
      this,
      autoClose,
      timestamp,
      sessionLocalInterceptor == null ? interceptor :
sessionLocalInterceptor,
      settings.getDefaultEntityMode(),
      settings.isFlushBeforeCompletionEnabled(),
      settings.isAutoCloseSessionEnabled(),
      settings.getConnectionReleaseMode()
  );
}
```

- We can use this method when we decided to manage the Session our self.
- It does not try to store or pull the session from the current context.
- If we use this method, we need to **flush() and close()** the session. It does not flush and close() automatically.

Example:

```
 Transaction transaction = null;
      Session session = HibernateUtil.getSessionFactory().openSession();
      try {
          transaction = session.beginTransaction();
          // do Some work
           session.flush(); // extra work
          transaction.commit();
      } catch(Exception ex) {
          if(transaction != null) {
              transaction.rollback();
```

```
        }
         ex.printStackTrace();
      } finally {
         if(session != null) {
             session.close(); // extra work
         }
      }
```

2. getCurrentSession():

- Session getCurrentSession() throws HibernateException
- Obtains the **current session**. The "**current session**" refers to a Hibernate Session bound by Hibernate behind the scenes, to the transaction scope.
- A session is opened whenever getCurrentSession() is called for the first time and closed when the transaction ends. This creates a brand new session if one does not exist or uses an existing one if one already exists. It automatically configured with both auto-flush and auto-close attributes as true means Session will be automatically flushed and closed.
- We can use getCurrentSession() method when our ***transaction runs long time.*** To use this method we need to configure one property in **hibernate.cfg.xml** file. The property is <property name=”hibernate.current_session_context_class”>thread </property>
- **public org.hibernate.classic.Session** getCurrentSession() throws **HibernateException** {

```
public org.hibernate.classic.Session getCurrentSession() throws
HibernateException {
    if ( currentSessionContext == null ) {
       throw new HibernateException( "No CurrentSessionContext
configured!" );
    }
    return currentSessionContext.currentSession();
}
```

Example:

```
        Transaction transaction = null;
        Session session =
HibernateUtil.getSessionFactory().getCurrentSession();
        try {
            transaction = session.beginTransaction();
            // do Some work
            // session.flush(); // no need
            transaction.commit();
        } catch(Exception ex) {
            if(transaction != null) {
                transaction.rollback();
            }
            ex.printStackTrace();
        } finally {
            if(session != null) {
                // session.close(); // no need
            }
        }
```

Application.java

```
public class Application {
    public static void main(String[] args) {
        SessionFactory sessionFactory1 =
HibernateUtil.getSessionFactory();
        Session session1 = null, session2 = null;

        session1 = sessionFactory1.openSession();
        session2 = sessionFactory1.openSession();

// comparing the session
        // Note1
        if (session1 == session2) {
            System.out.println("Both seesions are equal by using
openSession()");
        } else {
            System.out.println("Both seesions are not equal by using
openSession()");
        }
        session1 = sessionFactory1.getCurrentSession();
        session2 = sessionFactory1.getCurrentSession();
        // Note2
        if (session1 == session2) {
            System.out.println("Both seesions are equal by using
getCurrentSession()");
        } else {
            System.out.println("Both seesions are not equal by using
getCurrentSession()");
        }
    }
}
```

Output:
Both seesions are not equal by using openSession()
Both seesions are equal by using getCurrentSession()
Working Example:

```
ContextualSession
|-src
    |-com.cs.entites
        |-AdEnquiry.java
        |-AdEnquiry.hbm.xml
   |-com.cs.delegate
        |-EnquiryDelegate.java
   |-com.cs.vo
        |-EnquiryVo.java
   |-com.cs.dao
        |-EnquireDao.java
   |-com.cs.util

        |-HibernateUtil.java
        |-GlobalSessionContext.java
  |-com.cs.test
        |-CStest.java
|-hibernate.cfg.xml
```

AdEnquiry.java

```
public class AdEnquiry {
        private int id;
        private String description;
        private String place;
        private int duration;
        private String customer;
//setters and getters
}
```

AdEnquiry.hbm.xml

```
<hibernate-mapping package="com.cs.entities">
        <class name="AdEnquiry" table="AD_ENQUIRY">
                <id name="id" column="AD_ENQUIRY_ID">
                        <generator class="increment" />
                </id>
                <property name="description" column="DESCR" />
                <property name="place" column="PLACE" />
                <property name="duration" column="DURATION" />
                <property name="customer" column="CUSTOMER_NM" />
        </class>
```

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

hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
                <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</prop
erty>
                  <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property
>
                <property
name="connection.username">hib_tools</property>
                <property
name="connection.password">welcome1</property>
                <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</pro
perty>
                <property name="hibernate.show_sql">true</property>
                <property name="hbm2ddl.auto">update</property>
                <property
name="current_session_context_class">com.cs.util.GlobalSessionC
ontext</property>
             <mapping resource="com/cs/entities/AdEnquiry.hbm.xml" />
        </session-factory>
</hibernate-configuration>
```

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;

        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
        }
 }
```
EmployeeVo.java
```
public class EnquiryVo {
        private int id;
        private String description;
        private String place;
        private int duration;
        private String customer;
//setters and getters
}
```
EnquiryDelegate.java
```
public class EnquiryDelegate {
        public int storeEnquiry(EnquiryVo vo) {
                boolean flag = false;
                int enquiryId = 0;
                AdEnquiry adEnquiry = null;
                EnquiryDao enquiryDao = null;
                Transaction transaction = null;

                try {
                        adEnquiry = new AdEnquiry();
                        adEnquiry.setDescription(vo.getDescription());
                        adEnquiry.setPlace(vo.getPlace());
                        adEnquiry.setDuration(vo.getDuration());
                        adEnquiry.setCustomer(vo.getCustomer());

                        enquiryDao = new EnquiryDao();
                        /*Session session =
HibernateUtil.getSessionFactory().openSession();
                        ManagedSessionContext.bind(session);*/
transaction = HibernateUtil.getSessionFactory().getCurrentSession()
                                .beginTransaction();
                        enquiryId = enquiryDao.saveEnquiry(adEnquiry);
flag = true;
                } finally {
                        if (transaction != null) {
                                if (flag) {
                                        transaction.commit();
                                } else {
                                        transaction.rollback();
                                }
                        }
        /*ManagedSessionContext.unbind(HibernateUtil.getSessionFactor
y());*/
                }
                return enquiryId;
        }
}
```
EnquiryDao.java
```
public class EnquiryDao {
        public int saveEnquiry(AdEnquiry enquiry) {
                SessionFactory sfactory = null;
                Session session = null;
                int id = 0;

                sfactory = HibernateUtil.getSessionFactory();
                session = sfactory.getCurrentSession();
                id = (Integer) session.save(enquiry);
                return id;
        }
}
```
CSTest.java
```
public class CSTest {
        public static void main(String[] args) {
                EnquiryVo vo = null;
                EnquiryDelegate delegate = null;

                try {
                        vo = new EnquiryVo();
                        vo.setDescription("Surya Kiran General Stores
Advertisement");
                        vo.setPlace("ameerpet");
                        vo.setDuration(60);
                        vo.setCustomer("surya");

                        delegate = new EnquiryDelegate();
                        int id = delegate.storeEnquiry(vo);
                        System.out.println("id: " + id);
                } finally {
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Configuring our own SessionContext:

```
public class GlobalSessionContext extends
AbstractCurrentSessionContext {
        private Session session;
public GlobalSessionContext(SessionFactoryImplementor factory) {
                super(factory);
        }
@Override
        public Session currentSession() throws HibernateException {
                System.out.println("currentSession()");
                if (session == null || (session != null && session.isOpen() ==
false)) {
                        session = super.factory().openSession();
                }
                return session;
        }
}
```

Part-B

Mappings

1.Introduction:

Within an Application we may have multiple objects,where one Object will be related with another object using two ways and they are:

- Using Inheritance.
- Using Association.
  (✓Association✓Aggregation✓Composition)

Lets say we have two Entity classes Person and Address,where Address contained within the Person as follows..

```
→public class person{
          private int personId;
          private String firstName;
          private String lastName;
        -------------------------------
   priavte Address address;
}
```

```
→Public class Address{
            private String addressLine1;
            private String addressLine2;
            private String city;
            private String state;
      --------------------------------
}
```

Whenever we created Person person=new Person(); we populated the data including address and whenever we call session.save(person); Then the Hibernate will store **not only** the **person** data **but also** the **address** contained within the person.why because every piece of information that is there will be placed interms of objects.so when we have the person containing the address,where we will expect Address also has to be persisted along with person.

As Here Hibernate is persisting the data within database, so to the Hiberanate we need to tell the information about how to map the Entity objects with the corresponding relations which is nothing but **Hibernate Relationship Mapping**.

Lets say we have a class called **Courier** as follows..

```
public class Courier{
              private int id;
              private String description;
              private String source;
              private String destination;
              private Date courierDate;
              private float amount:
              //setters and getters
      }
```

Lets say we have one more class **OpenCourier** as follows..

```
Public class OpenCourier{
                        private int id;
                        private String description;
                        private String source;
                        private String destination;
                        private Date courierDate;
                        private float amount;
                        private float estimatedGoodsValue;
   }
```

Here almost all the properties which are there in the courier are also there in the OpenCourier.But it is **not** recommended to **re-declare** the same attributes in both the classes.Instead of that we can establish **IS-A** relationship between these two classes.i,e **OpenCourier** should **extends** from **Courier**.If we do like this then **all** the **attributes** within the Courier will be **inherited** to OpenCourier.

Whenever we tried persisting the OpenCourier object as **session.save(openCourier);** then Then along with the Parent class attributes,child class attributes should also persistable.But there will be inheritance or association relationship containing one Table inside another Table,The only way of expressing the Relationship between the Tables is by using **PrimaryKeys** and **foreignKeys**.In such cases how to map the Data that is there within the Entity objects based on their relationships into the corresponding RelationalModel, which is nothing but **Mappings**.

So we have two types of Mappings and they are as below....

🌏How to map inheritance relationship between the classes and Relational-Model"

🌏How to map association relationship between the classes and Relational - Model".

2.Impedance-mismatch:

When we are trying to map Object model to the relational model to perform persitency we are going to encounter lot of **problems** and we call this beaviour as 'Impedance-mismatch".It is the bottom line of ORM Technology.

What is ORM?

The **Expert group** of people provided certain **solutions** for the problems (impedance-mismatch) which are going to encounter While we are trying to map Object model to the relational model to perform persitency.

Solving the impedance problems,which will encounter While we are trying to map Object model to the relational model to perform persitency can be called as ORM.

Similary Hibernate also provided the solutions for the Impedance mismatch by using" **inheritance mappings"**& **"association mappings".**

30-07-2016

3.Impedance-mismatch problems:

✓Granularity mismatch
✓SubTypes
✓Identity
✓Association
✓Navigation

1.Granularity mismatch:

→The number of classes and the Number of Tables into which we wanted to persist the data may not be same.

→Lets say we have one PERSON Table which holds the details of a person including the address.

Id	F_NM	L_NM	GEN	DOB	AGE	A_Line1	A_Line2	CITY	STATE	ZIP

The Person Entity class will looks like as below..

```
public class Person{
            private int id;
            private String firstName;
            private String lastName;
            private String gender;
            private string age;
priavte Address address;
//setters and getters
}
```

Lets say addressLine1; addressLine2; city; state are the common attributes of of the several entity classes like person.So rather than re-declaring those attributes in every entity class,it is recommended to create a separate Entity class for those attributes as below...

```
Public class Address{
              private String addressLine1;
              private String addressLine2;
              private String city;
              private String state;
       //setters and getters
}
```

→If we wanted to persist the data that is there within the Person object,But here the Person object also contain another class also as an attribute within it. So To perssit the **Person** with the **Address** we need **Two tables** but within the database we have **only one** Table as **PERSON**.So this problem is called as "**granularity-mismatch**".

2.SubTypes:

→A class extending from another class is called SubTypes.

→Lets say we have a class called TOY as follows..

```
public Toy{
       private int id;
       private String description;
       private float amount;
      //setters and getters
}
```

A Toy can be of multiple Types so we have one more class called "RemoteToy" and will be as follows..

```
public RemoteToy extends Toy{
       private int iRFrequency;
       private String sensorType;
      //setters and getters
}
```

→while we are persistig the **Toy** we should be able to store the data wihtin the Table,where as while we are perssiting the **RemoteToy** we should **able** to **persist** the **Toy data** also **along** with **RemoteToy**.
→How do we need to create the database tables that supports us to store both the Toy data and RemoteToy data objects,so in such cases creating **Tables** and establisingh the **inheritance relationship between** them is **not possible**.
→In such cases how do we need to support to persist the data that is there wihtin the classes which has inheritance relatonship with each other is one more problem and is called as **subTypes**.

3.Identity

→Lets say we have a record in the **database**,to identify the uniquely **record** within the database we need **PrimaryKey**.So from database point of view PK is acting **as Identiy**.
→when it comes to Entity class,In the database table To find wether two rows are same or not?we can comparae their PK's.
where as when it comes to **Entity class** If we wanted to find wether **two objects** are equally **same or not**?we can use either **hashCode()** and **equals().**
→So when it comes to the database identity of the record means "**one{PK}**" where as when it comes to the classes identity of the object is "**two**{hashCode() and equals()}.so this mismatch is called **identiy mismatch**.
remedy:In Hibernate First Level cache resolves this problem.

4.Association

→A class will have association with another class either via **composition** or **navigation** or **association** or **dependancy**.But we can not create one Table inside another Table.

5.Navigation

→Lets say we have Two **Tables OWNER** and **CAR** .If we want to **find** the **owner** of the car we need to **navigate** between the tables and we need to **compare** the **primary keys.**

→Lets say we have a class called Owner as follows..

```
Pubic class Owner{
        priavte int Id;
        priavate String name;
        priavate String email;
        priavate String mobile;
//setters and getters
    }
```

and we have one more class Car as follows..

```
public class Car{
          private int registrationNo;
          private String model:
          private String manufacturer:
     priavte Owner owner;
// setters and getters
}
```

→Where as when it comes to Enttiy class we can find the owner of the car as follows..

```
        Car c=new Car();
              c.getOwner();
```

So here we are navigating between the objects using (.) **indirection operator**. But to find the related records in the database it is not possible to traverse using indirection operator.so this is called **navigation mis-match**.

01-08-2016

3.Types of Inheritance Mapping in Hibernate:

There are three types of inheritance mapping in hibernate

1. Table per class hierarchy

2. Table per concrete class Hierarchy

3. Table per subclass hierarchy

Lets we have an Entity Class **InsurancePlan** and it will looks like as follows..

```
public class InsurancePlan {
        protected int uPlanNo;
        protected String planName;
        protected String description;
        protected String planProvider;
   //setters and getters
 }
```

The InsurancePlan has the **LifeInsurancePlan** and **AccidentalInsurancePlan** as **SubClasses**.These are as follows..

```
public class LifeInsurancePlan extends InsurancePlan {
                  protected int eligibleAge;
                  protected int planPeriod;
                  protected boolean preMedicalCheckup;
     //setters and getters
}
public class AccidentalInsurancePlan extends InsurancePlan {
                 protected int coveragePeriod;
                 protected boolean partialDisability;
                 protected boolean deathCoverage;
   //setters and getters
}
```

point to remember:

1.If We tried persisting **InsurancePlan** then the the data related to the **InsurancePlan** has to be perssited.

2.If we tried perssiting **LifeInsurancePlan** then Both **InsurancePlan** the **LifeInsurancePlan** has to be persisted.

3.If we tried perssiting **AccidentalInsurancePlan** then Both **InsurancePlan** the **AccidentalInsurancePlan** has to be persisted.

So we should comeUp with a Table which supports to store Any of the objects of this Relationship or Hierarchy.

1. Table per class hierarchy

Case#1(One Table)

Lets consider one Single MASTER_TABLE where we can declare all the attributes as COLUMNS and we can store any data of any of these classes.

INSURANCE_PLANS(Table for hierarchy)
  |-U_PLAN_NO
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER
  |-ELIGIBLE_AGE
  |-PLAN_PERIOD
  |-PRE_MEDICAL_CHECKUP_REQUIRED
  |-COVERAGE_PERIOD
  |-PARTIAL_DISABILITY
  |-DEATH_COVERAGE

1.Now If we tried persisting the InsurancePlan object into INSURANCE_PLANS Then it will perssit uPlanNo; planName;description;planProvider attributes.
2.Now If we tried persisting LifeInsurancePlan the object into INSURANCE_PLANS Then it will perssit uPlanNo; planName;description; planProvider & eligibleAge;planPeriod;preMedicalCheckup; attributes
3.Now If we tried persisting the AccidentalInsurancePlan object into INSURANCE_PLANS then it will persist uPlanNo; planName; description; planProvider & coveragePeriod; partialDisability; deathCoverage attributes.
→So here per this hierarchy of classes, we created ONE Table,so we will call this strategy as **Table per class hierarchy**.

2. Table per concrete class Hierarchy

Case#2(Three Tables)
→INSURANCE_PLANS (separate Table per concrete class)
  |-U_PLAN_NO
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER
→LIFE_INSURANCE_PLANS(separate Table per concrete class)
  |-LIFE_INSURANCE_UPLAN_NO
  |-ELIGIBLE_AGE
  |-PLAN_PERIOD
  |-PRE_MEDICAL_CHECKUP_REQUIRED
→ACCIDENTAL_INSURANCE_PLAN(separate Table per concrete class)
  |- ACCIDENTAL_INSURANCE_PLAN_UPLAN_NO
  |-COVERAGE_PERIOD
  |-PARTIAL_DISABILITY
  |-DEATH_COVERAGE

→Here per Every Individual class we created one Table,so we will call this strategy as **TablePerConcreteClass hierarchy**.

3. Table per SubClass hierarchy

Case#3(Three Tables)

→INSURANCE_PLANS(One Table per Super class)
  |-U_PLAN_NO
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER

→LIFE_INSURANCE_PLANS(One Table per Sub Class)
  |-**U_PLAN_NO**
  |-ELIGIBLE_AGE
  |-PLAN_PERIOD
  |-PRE_MEDICAL_CHECKUP_REQUIRED

similarly for AccidentalInsurancePlan we need to create one more Table.

→ACCIDENTAL_INSURANCE_PLAN(One Table Per Sub Class)
  |-**U_PLAN_NO**
  |-COVERAGE_PERIOD
  |-PARTIAL_DISABILITY
  |-DEATH_COVERAGE

→So here we created a separate Table per Every Subclass so we will call this strategy as **TablePerSubClass hierarchy**.

So these are the **Three** possible wasy to design tables in the database to store the entity object when it comes to **inheritance** mapping mechanism.

02-08-2016&03-08-2016

1. Table per class hierarchy

Working with TablePerClass(**Xml Mapping**)

```
InsurancePolicyTablePerClass
|-src
  |-com.tpc.entities
     |-InsurancePlan.java
     |- InsurancePlan.hbm.xml
     |-LifeInsurancePlan.java
     |-LifeInsurancePlan.hbm.xml
     |-AccidentalInsurancePolicy.java
     |-AccidentalInsurancePolicy.hbm.xml
  |-com.tpc.util
     |-Hibernateutil.java
  |-com.tpc.test
    |-TPCTest.java
|-hibernate.cfg.xml
```

If we wanted to persist the InsurancePlan details into the database,The Entity class will looks as below.

InsurancePlan.java

```
→public class InsurancePlan {
        protected int uPlanNo;
        protected String planName;
        protected String description;
        protected String planProvider;
    //setters and getters
//@toString
 }
```

To store this Entity class object within the corresponding record in the database then we need mapping file for the Entity class as below.

InsurancePlan.hbm.xml

```
<hibernate-mapping package="com.tpc.entities">
<class name="InsurancePlan" table="INSURANCE_PLAN"  discriminator-
value="InsurancePlan">
<id name="uPlanNo" column="U_PLAN_NO">
<generator class="increment" />
</id>
  <discriminator column="PLAN_TYPE" />
<property name="planName" column="PLAN_NM" not-null="true" />
<property name="description" column="DESCR" not-null="false" />
<property name="planProvider" column="PLAN_PROVIDER"not-null="true" />
      </class>
</hibernate-mapping>
```

//For this class (or) for any of the sub classes To identify which record belongs to which class, we need one more additional column **<discriminator column>** in the super class.It is used for distinguishing which record belongs to which class.

//Now This class not only persist the related properties and it should also insert "PLAN_TYPE" also.so we need to declare that attribute as **discriminator-value="InsurancePlan**.

→We have LifeInsurancePlan Entity class also as follows..

LifeInsurancePlan.java

```
public class LifeInsurancePlan extends InsurancePlan {
```

```
                protected int eligibleAge;
                protected int planPeriod;
                protected boolean preMedicalCheckup;
      //setters and getters
}
```

As this class is sub class to the “InsurancePlan”.lets see how to configuring it tin the mapping file.

LifeInsurancePlan.hbm.xml

```
<hibernate-mapping package="com.tpc.entities">
<subclass name="LifeInsurancePlan" extends="InsurancePlan"   discriminator-
value="LifeInsurancePlan">
<property name="eligibleAge" column="MAX_ELIGIBLE_AGE" />
  <property name="planPeriod" column="MIN_PLAN_PERIOD" />
<property name="preMedicalCheckup"
column="PRE_MEDICAL_CHECKUP_REQ" />
       </subclass>
</hibernate-mapping>
```

Similarly we have one more Entity class as follows..

AccidentalInsurancePolicy.java

```
→public class AccidentalInsurancePlan extends InsurancePlan {
                protected int coveragePeriod;
                protected boolean partialDisability;
                protected boolean deathCoverage;
    //setters and getters
}
```

As it also a sub class for the InsurancePlan class we need to write the mapping file as follows..

AccidentalInsurancePolicy.hbm.xml

```
<hibernate-mapping package="com.tpc.entities">
<subclass name="AccidentalInsurancePlan" extends="InsurancePlan"
discriminator-value="AccidentalInsurancePlan">
<property name="coveragePeriod" column="COVERAGE_PERIOD" />
<property name="partialDisability" column="PARTIAL_DISABILITY_COVERAGE"
/>
<property name="deathCoverage" column="DEATH_COVERAGE" />
</subclass>
</hibernate-mapping>
```

→Once we have the Entity class,its sub classes and their corresponding mapping files next we need configuration details to interact with the database.

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
      <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/tpc/entities/InsurancePlan.hbm.xml" />
<mapping resource="com/tpc/entities/LifeInsurancePlan.hbm.xml" />
<mapping resource="com/tpc/entities/AccidentalInsurancePlan.hbm.xml" />
         </session-factory>
</hibernate-configuration>
```

→Once we have the hibernate.cfg.xml then we need to create the SessionFactoty as follows..

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

Now we can Test wether it is working properly or not?

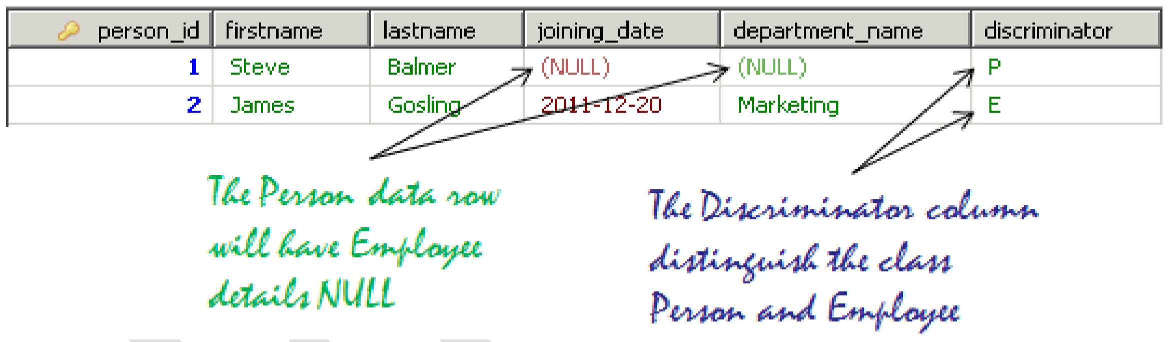
TPCTest.java

```
public class TPCTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
        // entities
                InsurancePlan ip = null;
                LifeInsurancePlan lip = null;
                AccidentalInsurancePlan aip = null;
        try {
              ip = new InsurancePlan();
                         ip.setPlanName("Jeevan Anand");
                         ip.setDescription("Risk/Life coverage plan");
                         ip.setPlanProvider("Uhg");
                lip = new LifeInsurancePlan();
                         lip.setPlanName("Jeevan Abhay");
                         lip.setDescription("Life Coverage Plan");

                        lip.setPlanProvider("Uhg");
                        lip.setEligibleAge(60);
                        lip.setPlanPeriod(120);
                        lip.setPreMedicalCheckup(true);
        aip = new AccidentalInsurancePlan();
                        aip.setPlanName("Risk Coverage");
                        aip.setDescription("Accidental Coverage Plan");
                        aip.setPlanProvider("aithena");
                        aip.setCoveragePeriod(60);
                        aip.setPartialDisability(true);
                        aip.setDeathCoverage(true);
sessionFactory = HibernateUtil.getSessionFactory();
session = sessionFactory.getCurrentSession();
transaction = session.beginTransaction();
                         session.save(ip);
                        session.save(lip);
                        session.save(aip);
 flag = true;
                } finally {
                        if (transaction != null) {
                                if (flag) {
                                transaction.commit();
                        } else {
                                transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test(sample output related to other classes)

Note:the attributes which are not there with Super class returns null.



person_id	firstname	lastname	joining_date	department_name	discriminator
1	Steve	Balmer	(NULL)	(NULL)	P
2	James	Gosling	2011-12-20	Marketing	E

The Person data row will have Employee details NULL
The Discriminator column distinguish the class Person and Employee

<u>**Test**:(Accessing the data from the database.)</u>

<u>permutations &combinations</u>

→InsurancePlan plan = (InsurancePlan) session.get(InsurancePlan.class, **2**);
System.out.println(plan);*/.

where as if we genearate the toString and hashCode() in the InsurancePlan then we wil get the data of bothe the InsurancePlan and LifeInsurancePlan.

```
@Override
        public boolean equals(Object obj) {
                if (this == obj)
                        return true;
                if (obj == null)
                        return false;
                if (getClass() != obj.getClass())
                        return false;
                InsurancePlan other = (InsurancePlan) obj;
                if (planName == null) {
                if (other.planName != null)
                                return false;
        } else if (!planName.equals(other.planName))
                return false;
        if (planProvider == null) {
                if (other.planProvider != null)
                        return false;
        } else if (!planProvider.equals(other.planProvider))
                return false;
                return true;
}
```

→/*lip = (LifeInsurancePlan) session.get(LifeInsurancePlan.class, **2**);
System.out.println(lip);*/

o/p:only the LifeInsurancePlan will be generated

→/*lip = (LifeInsurancePlan) session.get(LifeInsurancePlan.class, **3**);
System.out.println(lip);*/

o/p:not found.

-----------------XML Mapping completed---------------------

Working with TablePerClass(**Annotations Mapping**)

InsurancePlan.java

```
@Entity
@Table(name = "INSURANCE_PLAN")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "PLAN_TYPE")
@DiscriminatorValue("InsurancePlan")
public class InsurancePlan {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
protected int uPlanNo;
        @Column(name = "U_PLAN_NM")
protected String planName;
        @Column(name = "DESCR")
protected String description;
        @Column(name = "PLAN_PROVIDER")
protected String planProvider;
//setters and getters
//@hashCode
//@toString
}
```

LifeInsurancePlan.java

```
@Entity
@DiscriminatorValue("LifeInsurancePlan")
public class LifeInsurancePlan extends InsurancePlan {
@Column(name = "MAX_ELIGIBLE_AGE")
protected int eligibleAge;
@Column(name = "PLAN_PERIOD")
protected int planPeriod;
@Column(name = "PRE_MEDICAL_CHECKUP_REQ")
protected boolean preMedicalCheckup;
//setters and getters
//@toString
}
```

AccidentalInsurancePlan.java

```
@Entity
@DiscriminatorValue("AccidentalInsurancePlan")
public class AccidentalInsurancePlan extends InsurancePlan {
        @Column(name = "COVERAGE_PERIOD")
        protected int coveragePeriod;
        @Column(name = "PARTIAL_DISABILITY_COVERAGE")
        protected boolean partialDisability;
        @Column(name = "DEATH_COVERAGE")
        protected boolean deathCoverage;
//setters and getters
//@toString
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</property>
                <property name="connection.username">root</property>
                <property name="connection.password">root</property>
```

```
<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
                <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property
name="hibernate.current_session_context_class">thread</property>
<mapping class="com.tpc.entities.InsurancePlan"/>
        <mapping class="com.tpc.entities.LifeInsurancePlan"/>
        <mapping class="com.tpc.entities.AccidentalInsurancePlan"/>
        </session-factory>
</hibernate-configuration>
```

Note:Remaining Util and Test classes are same as XML mapping only.

04-08-2016&05-08-2016

3. Table per SubClass hierarchy

WorkingWithTablePerSubClass(**XML Mapping**)

Here we have Three Entity classes "InsurancePlan", and the sub classes LifeInsurancePlan", "AccidentalInsurancePlan".

Here for the above Entity classes we can create the tables as follows..

→INSURANCE_PLANS(**One** Table per **Super** class)
  |-U_PLAN_NO(**PK**)
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER

→LIFE_INSURANCE_PLANS(**One** Table per **Sub** Class)
  |-**U_PLAN_NO(FK)**
  |-ELIGIBLE_AGE
  |-PLAN_PERIOD
  |-PRE_MEDICAL_CHECKUP_REQUIRED

similarly for AccidentalInsurancePlan we need to create one more Table.

→ACCIDENTAL_INSURANCE_PLAN(**One** Table Per **Sub** Class)
  |-**U_PLAN_NO(FK)**
  |-COVERAGE_PERIOD
  |-PARTIAL_DISABILITY
  |-DEATH_COVERAGE

```
InsurancePolicyTablePerSubClass
|-src
  |-com.tpsc.entities
    |-InsurancePlan.java
```

```
        |-InsurancePlan.hbm.xml
        |-LifeInsurancePlan.java
        |-LifeInsurancePlan.hbm.xml
        |-AccidentalInsurancePolicy.java
        |-AccidentalInsurancePolicy.hbm.xml
    |-com.tpsc.util
        |-Hibernateutil.java
   |-com.tpsc.test
        |-TPSCTest.java
|-hibernate.cfg.xml
```

InsurancePlan.java

Lets say we have an Entity Class **InsurancePlan** and it will looks like as follows..

```
public class InsurancePlan {
        protected int uPlanNo;
        protected String planName;
        protected String description;
        protected String planProvider;
    //setters and getters//@equals,@hashcode,@toString
 }
```

The InsurancePlan has the **LifeInsurancePlan** and **AccidentalInsurancePlan** as **SubClasses**.These are as follows..

LifeInsurancePlan.java

```
public class LifeInsurancePlan extends InsurancePlan {
                protected int eligibleAge;
                protected int planPeriod;
                protected boolean preMedicalCheckup;
      //setters and getters,@toString
}
```

AccidentalInsurancePlan.java

```
public class AccidentalInsurancePlan extends InsurancePlan {
                protected int coveragePeriod;
                protected boolean partialDisability;
                protected boolean deathCoverage;
    //setters and getters,@toString}
```

InsurancePlan.hbm.xml

In order to let the hibernate to store these Entity classes into the database we need to write the corresponding mapping files.

```
<hibernate-mapping package="com.tpsc.entities">
        <class name="InsurancePlan" table="INSURANCE_PLAN">
```

```
            <id name="uPlanNo" column="U_PLAN_NO">
                <generator class="increment" />
            </id>
            <property name="planName" column="PLAN_NM" />
            <property name="description" column="DESCR" />
            <property name="planProvider" column="PLAN_PROVIDER" />
        </class>
</hibernate-mapping>
```

LifeInsurancePlan.hbm.xml

```
 <hibernate-mapping package="com.tpsc.entities">
        <joined-subclass name="LifeInsurancePlan"
table="LIFE_INSURANCE_PLAN" extends="InsurancePlan">
            <key column="U_PLAN_NO" />
        <property name="eligibleAge" column="MAX_ELIGIBLE_AGE"
        not-null="true" />
<property name="planPeriod" column="PLAN_PERIOD" not-null="true" />
            <property name="preMedicalCheckup"
column="PRE_MEDICAL_CHECKUP_REQ"not-null="true" />
        </joined-subclass>
</hibernate-mapping>
```

AccidentalInsurancePlan.hbm.xml

```
<hibernate-mapping=”com.tpsc.entites”>
  <joined-subclass name="AccidentalInsurancePlan"
table="ACCIDENTAL_INSURANCE_PLAN" extends="InsurancePlan">
        <!-- foreign key column of my table -->
        <key column="U_PLAN_NO"/>
           <property name="coveragePeriod" column="COVERAGE_PERIOD"
not-null="true"/>
           <property name="partialDisability"
column="PARTIAL_DISABILITY_COVERAGE" not-null="true"/>
          <property name="deathCoverage" column="DEATH_COVERAGE"
not-null="true" />
        </joined-subclass>
</hibernate-mapping>
```

hibernate.cfg.xml

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

```
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/tpsc/entities/InsurancePlan.hbm.xml" />
<mapping resource="com/tpsc/entities/LifeInsurancePlan.hbm.xml" />
<mapping resource="com/tpsc/entities/AccidentalInsurancePlan.hbm.xml" />
           </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                   }
```

Now we can Test wether it is working properly or not?

```
public class TPSCTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;

                InsurancePlan ip = null;
                LifeInsurancePlan lip = null;
                AccidentalInsurancePlan aip = null;

                try {
                sessionFactory = HibernateUtil.getSessionFactory();
                session = sessionFactory.getCurrentSession();
                transaction = session.beginTransaction();
/*ip = new InsurancePlan();
                        ip.setPlanName("Jeevan Arogya");
                        ip.setDescription("Arogyam plan");
                        ip.setPlanProvider("icici lamboard");
    session.save(ip);

lip = new LifeInsurancePlan();
                        lip.setPlanName("Jeevan Abhaya");
                        lip.setDescription("Life coverage policy");
                        lip.setPlanProvider("lic");
                        lip.setEligibleAge(60);
                        lip.setPlanPeriod(360);
                        lip.setPreMedicalCheckup(true);
     session.save(lip);

aip = new AccidentalInsurancePlan();
                        aip.setPlanName("Jeevan Anand");
                        aip.setDescription("Risk coverage policy");
                        aip.setPlanProvider("Bajaj Allianz");
                        aip.setCoveragePeriod(365);
                        aip.setPartialDisability(false);
                        aip.setDeathCoverage(true);
     session.save(aip);*/
                        flag = true;
  } finally {
                        if (transaction != null) {

                                if (flag) {
                                transaction.commit();
```

```
                    } else {
                            transaction.rollback();
                    }
            }
                    HibernateUtil.closeSessionFactory();
            }
    }
}
```

<u>**Test**☹(Inserting the data)</u>

case#1-when we tried call session.save(ip) then the data will be stored into the INSURANCE_PLAN.

case#2-when we tried call session.save(lip) then the data will be stored into both the INSURANCE_PLAN and LIFE_INSURANCE_PLAN.

case#3-when we tried call session.save(aip) then the data will be stored into both the INSURANCE-PLAN& ACCIDENTAL_INSURANCE_PLAN.

Note:1)Now INSURANCE_PLAN will contains THREE records.
  2)Now LIFE_INSURANCE_PLAN will contains TWO records.
  3)Now ACCIDENTAL_INSURANCE_PLAN will contains TWO records.

<u>**Test**☹(Accessing the data)</u>

1) lip = (LifeInsurancePlan) session.get(LifeInsurancePlan.class, 2);
System.out.println(lip);

o/p:Now the LifeInsuranceClass object will come by joining INSURANCE_PLAN and LIFE_INSURANCE_PLAN Tables.

2)) lip = (LifeInsurancePlan) session.get(LifeInsurancePlan.class, 2);
System.out.println(lip);

o/p: It will returns null because joining these Tables with 3 is not there even though 3 exists.

3) ip = (InsurancePlan) session.get(InsurancePlan.class, 3);
System.out.println(ip);

o/p: The Now it will return InsurancePlan object by joining all the three tables.

WorkingWithTablePerSubClass(**Annotation Mapping**)

InsurancePlan.java

```
@Entity
@Table(name = "INSURANCE_PLAN")
@Inheritance(strategy=InheritanceType.JOINED)
public class InsurancePlan {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        protected int uPlanNo;
        @Column(name = "U_PLAN_NM")
        protected String planName;
        @Column(name = "DESCR")
        protected String description;
        @Column(name = "PLAN_PROVIDER")
        protected String planProvider;
//setters and getters
//@toString()
//@equals()
//@hashCode()
}
```

LifeInsurancePlan.java

```
@Entity
@Table(name="LIFE_INSURANCE_PLAN")
@PrimaryKeyJoinColumn(name="U_PLAN_NO")
public class LifeInsurancePlan extends InsurancePlan {
        @Column(name = "MAX_ELIGIBLE_AGE")
        protected int eligibleAge;
        @Column(name = "PLAN_PERIOD")
        protected int planPeriod;
        @Column(name = "PRE_MEDICAL_CHECKUP_REQ")
        protected boolean preMedicalCheckup;
//setters and getters
//@toString()
}
```

AccidentalInsuracePlan.java

```
@Entity
@Table(name = "ACCIDENTAL_INSURANCE_PLAN")
@PrimaryKeyJoinColumn(name = "U_PLAN_NO")
public class AccidentalInsurancePlan extends InsurancePlan {
	@Column(name = "COVERAGE_PERIOD")
	protected int coveragePeriod;
	@Column(name = "PARTIAL_DISABILITY_COVERAGE")
	protected boolean partialDisability;
	@Column(name = "DEATH_COVERAGE")
	protected boolean deathCoverage;
//setters and getters
//@toString()
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
	<session-factory>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
		<property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</property>
		<property name="connection.username">root</property>
		<property name="connection.password">root</property>
	<property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
		<property name="show_sql">true</property>
	<property name="hibernate.hbm2ddl.auto">update</property>
	<property
name="hibernate.current_session_context_class">thread</property>
<mapping class="com.tpsc.entities.InsurancePlan"/>
	<mapping class="com.tpsc.entities.LifeInsurancePlan"/>
	<mapping class="com.tpsc.entities.AccidentalInsurancePlan"/>
	</session-factory>
</hibernate-configuration>
```

Note:Remaining Util and Test classes are same as XML mapping only.

Test:

case#1-persisting the data.

case#2-accessing the data.

06-08-2016

3. TablePerConcrete class Hierarchy

Working with TablePerConcreteClass (**Xml Mapping**)

Here we have Three Entity classes "InsurancePlan", and it has the sub classes LifeInsurancePlan", "AccidentalInsurancePlan".Here for the above Entity classes we can create the tables as follows.....

→INSURANCE_PLANS (separate Table per concrete class)
  |-U_PLAN_NO
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER
→LIFE_INSURANCE_PLANS(separate Table per concrete class)
  |-U_PLAN_NO (INSURANCE_PLAN_U_PLAN_NO)
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER
  |-ELIGIBLE_AGE
  |-PLAN_PERIOD
  |-PRE_MEDICAL_CHECKUP_REQUIRED
→ACCIDENTAL_INSURANCE_PLAN(separate Table per concrete class)
  |-U_PLAN_NO(ACCIDENTAL_INSURANCE_PLAN_U_PLAN_NO)
  |-PLAN_NAME
  |-DESCRIPTION
  |-PLAN_PROVIDER
  |-COVERAGE_PERIOD
  |-PARTIAL_DISABILITY
  |-DEATH_COVERAGE

points to remember:

DML

1.As Every class has a Table which consists all the columns representing both the super class and sub class attributes,so storing the data is easy task.because based on the object with the table we can store the data.

DQL

1.As Every sub class has a separate Table, we can direectly querying the data from sub class represented Table.

2.Unless the ID is unique,we can not find the class type of the record it means it does not allows to support polymorphism indirectly.

InsurancePlan.java

Lets say we have an Entity Class **InsurancePlan** and it will looks like as follows..

```
public class InsurancePlan {
        protected int uPlanNo;
        protected String planName;
        protected String description;
        protected String planProvider;
    //setters and getters
    //@equals,@hashcode,
    //@toString
 }
```

The InsurancePlan has the **LifeInsurancePlan** and **AccidentalInsurancePlan** as **SubClasses**.These are as follows..

LifeInsurancePlan.java

```
public class LifeInsurancePlan extends InsurancePlan {
                protected int eligibleAge;
                protected int planPeriod;
                protected boolean preMedicalCheckup;
     //setters and getters.
     //@toString
}
```

AccidentalInsurancePlan.java

```
public class AccidentalInsurancePlan extends InsurancePlan {
                protected int coveragePeriod;
                protected boolean partialDisability;
                protected boolean deathCoverage;
    //setters and getters,
    //@toString
}
```

The corresponding mapping files will be as follows..

InsurancePlan.hbm.xml

```
<hibernate-mapping package="com.tpcc.entities">
        <class name="InsurancePlan" table="INSURANCE_PLAN">
                <id name="uPlanNo" column="U_PLAN_NO">
                        <generator class="sequence" />
                </id>
                <property name="planName" column="PLAN_NM" />
                <property name="description" column="DESCR" />
                <property name="planProvider" column="PLAN_PROIVDER" />
        </class>
</hibernate-mapping>
```
LifeInsurancePlan.hbm.xml
```
<hibernate-mapping package="com.tpcc.entities">
        <union-subclass name="LifeInsurancePlan"
table="LIFE_INSURANCE_PLAN" extends="InsurancePlan">
        <property name="eligibleAge" column="MAX_ELIGIBLE_AGE" />
        <property name="planPeriod" column="PLAN_PERIOD" />
        <property name="preMedicalCheckup"
column="PRE_MEDICAL_CHECKUP_REQ" />
        </union-subclass>
</hibernate-mapping>
```
AccidentalInsurancePlan.hbm.xml
```
 <hibernate-mapping package="com.tpcc.entities">
        <union-subclass name="AccidentalInsurancePlan"
table="ACCIDENTAL_INSURANCE_PLAN" extends="InsurancePlan">
        <property name="coveragePeriod" column="COVERAGE_PERIOD" />
        <property name="partialDisability"
column="PARTIAL_DISABILITY_COVERAGE" />
        <property name="deathCoverage" column="DEATH_COVERAGE" />
        </union-subclass>
</hibernate-mapping>
```
once we have mapping files for the Entity classes next we need to write the hibernate.cfg.xml as follows..

hibernate.cfg.xml
```
<hibernate-configuration>
   <session-factory>
       <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
```

```
<mapping resource="com/tpcc/entities/InsurancePlan.hbm.xml" />
<mapping resource="com/tpcc/entities/LifeInsurancePlan.hbm.xml" />
<mapping resource="com/tpcc/entities/AccidentalInsurancePlan.hbm.xml" />
        </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

Now we can write Test class to check wether it is working or not?

```
 public class TPCCTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;

                InsurancePlan ip = null;
                LifeInsurancePlan lip = null;
                AccidentalInsurancePlan aip = null;

                try {
                sessionFactory = HibernateUtil.getSessionFactory();
                session = sessionFactory.getCurrentSession();
                transaction = session.beginTransaction();
/*ip = new InsurancePlan();
                        ip.setPlanName("Jeevan Arogya");
                        ip.setDescription("Arogyam plan");
                        ip.setPlanProvider("icici lamboard");
            session.save(ip);

    lip = new LifeInsurancePlan();
                        lip.setPlanName("Jeevan Abhaya");
                        lip.setDescription("Life coverage policy");
                        lip.setPlanProvider("lic");
                        lip.setEligibleAge(60);
                        lip.setPlanPeriod(360);
                        lip.setPreMedicalCheckup(true);
            session.save(lip);

    aip = new AccidentalInsurancePlan();
                        aip.setPlanName("Jeevan Anand");
                        aip.setDescription("Risk coverage policy");
                        aip.setPlanProvider("Bajaj Allianz");
                        aip.setCoveragePeriod(365);
                        aip.setPartialDisability(false);
                        aip.setDeathCoverage(true);
            session.save(aip);*/

    /*lip = (LifeInsurancePlan) session.get(LifeInsurancePlan.class, 2);
                System.out.println(lip);*/

    ip = (InsurancePlan) session.get(InsurancePlan.class, 3);
            System.out.println(ip);
                        flag = true;
      } finally {
                        if (transaction != null) {

                                if (flag) {
                                transaction.commit();
                        } else {
                                transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test☹(permutations and combinations)

07-08-2016&08-08-2016

Working with TablePerConcreteClass (**Annotation Mapping**)

InsurancePlan.java

```
@Entity
@Table(name = "INSURANCE_PLAN")
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
public class InsurancePlan {
        @Id
        @GeneratedValue(strategy = GenerationType.TABLE)
        protected int uPlanNo;
        @Column(name = "U_PLAN_NM")
        protected String planName;
        @Column(name = "DESCR")
        protected String description;
        @Column(name = "PLAN_PROVIDER")
        protected String planProvider;
//setters and getters
//@hashCode()
//@equals()
}
```

LifeInsurancePlan.java

```
@Entity
@Table(name = "LIFE_INSURANCE_PLAN")
public class LifeInsurancePlan extends InsurancePlan {
        @Column(name = "MAX_ELIGIBLE_AGE")
        protected int eligibleAge;
        @Column(name = "PLAN_PERIOD")
        protected int planPeriod;
        @Column(name = "PRE_MEDICAL_CHECKUP_REQ")
        protected boolean preMedicalCheckup;
//setters and getters
//@toString()
}
```

AccidentalInsurancePlan.java

```
@Entity
@Table(name = "ACCIDENTAL_INSURANCE_PLAN")
public class AccidentalInsurancePlan extends InsurancePlan {
@Column(name = "COVERAGE_PERIOD")

         protected int coveragePeriod;
        @Column(name = "PARTIAL_DISABILITY_COVERAGE")
        protected boolean partialDisability;
        @Column(name = "DEATH_COVERAGE")
        protected boolean deathCoverage;
//setterss and getters
//@toString()
}
```

hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
<property name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</property>
                <property name="connection.username">root</property>
                <property name="connection.password">root</property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
                <property name="show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <property
name="hibernate.current_session_context_class">thread</property>
<mapping class="com.tpcc.entities.InsurancePlan"/>
        <mapping class="com.tpcc.entities.LifeInsurancePlan"/>
        <mapping class="com.tpcc.entities.AccidentalInsurancePlan"/>
        </session-factory>
</hibernate-configuration>
```

Note:Remaining Util and Test classes are same as XML mapping only.
Test-(permutation and combinations)

Implicit Polymorphism mapping model

Implicit Polymorphism mapping is specific to Hibernate,It does support polymorphism and is not supported to work with JPA..

When it comes to Implicit Polymorphism mapping,Here we dont need to write any special things rather if we configure the classes as normal classes then Implicit Polymorphism mapping derives the inheritance relationship between those classes implicitly and persist the data.

Working Example on ImplicitPolymorphism:

```
ImplicitPolymorphism
|-src
    |-com.ip.entities
        |-cd.java
        |-cd.hbm.xml
        |-AudioCd.java
        |-AudioCd.hbm.xml
        |-VideoCd.java
        |-videoCd.hbm.xml
     |-com.ip.util
        |-HibernateUtil.java
     |-com.ip.test
        |-IPTest.java
|-hibernate.cfg.xml
```

Cd.java

```
public class Cd {
        private int id;
        private String title;
        private float price;
//setters and getters
//@toString()
}
```

Cd.hbm.xml

```
<hibernate-mapping package="com.ip.entities">
        <class name="Cd" table="CD">
                <id name="id" column="CD_ID">
                        <generator class="increment"/>
                </id>
        <property name="title" column="TITLE" not-null="true"/>
        <property name="price" column="PRICE" not-null="true" />
        </class></hibernate-mapping>
```

AudioCd.java

```
public class AudioCd extends Cd {
        private String artist;
        private int tracks;
//setters and getters
//@toString()
}
```

AudioCd.hbm.xml

```
<hibernate-mapping package="com.ip.entities">
        <class name="AudioCd" table="AUDIO_CD">
                <id name="id" column="CD_ID">
                        <generator class="increment"/>
                </id>
                <property name="title" column="TITLE" not-null="true"/>
                <property name="price" column="PRICE" not-null="true" />
                <property name="artist" column="ARTIST" not-null="true"/>
                <property name="tracks" column="TRACKS" not-null="true" />
        </class>
</hibernate-mapping>
```

VideoCd.java

```
public class VideoCd extends Cd {
        private String director;
        private long duration;
//setters and getters
//@toString()
}
```

VideoCd.hbm.xml

```
<hibernate-mapping package="com.ip.entities">
        <class name="VideoCd" table="VIDEO_CD">
                <id name="id" column="CD_ID">
                        <generator class="increment"/>
                </id>
        <property name="title" column="TITLE" not-null="true"/>
        <property name="price" column="PRICE" not-null="true" />
        <property name="director" column="DIRECTOR" not-null="true"/>
        <property name="duration" column="DURATION" not-null="true" />
</class>
</hibernate-mapping>
```

Hibernate.cfg.xml

```
<hibernate-configuration>
   <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/tpcc/entities/InsurancePlan.hbm.xml" />
<mapping resource="com/tpcc/entities/LifeInsurancePlan.hbm.xml" />
<mapping resource="com/tpcc/entities/AccidentalInsurancePlan.hbm.xml" />
          </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);

        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

Now we can write Test class to check wether it is working or not?

```
public class **IPTest** {
      public static void main(String[] args) {
              SessionFactory sessionFactory = null;
              Transaction transaction = null;
              Session session = null;
              boolean flag = false;
              try {
                      sessionFactory = HibernateUtil.getSessionFactory();
                      session = sessionFactory.getCurrentSession();
                      transaction = session.beginTransaction();
              /*Cd cd = new Cd();
                      cd.setTitle("Avatar");
                      cd.setPrice(23);
              session.save(cd);

              AudioCd acd = new AudioCd();
                      acd.setTitle("Dangerous");
                      acd.setPrice(334);
                      acd.setArtist("Micheal");
                      acd.setTracks(10);
              session.save(acd);

              VideoCd vcd = new VideoCd();
                      vcd.setTitle("IRobot");
                      vcd.setPrice(383);
                      vcd.setDirector("James");
                      vcd.setDuration(243);
                      session.save(vcd);*/

              Cd cd = (Cd) session.get(Cd.class, 2);
              System.out.println(cd);
               flag = true;
      } finally {
                      if (transaction != null) {

                              if (flag) {
                              transaction.commit();
                      } else {
                        transaction.rollback();
                    }
                }
                    HibernateUtil.closeSessionFactory();
                }
        }
}
```
Test☹(permutations and combinations)

Pros and cons of Inheritance Mapping Models:

- **Table Per Class**

  Advantages:

  1.As it representing all the classes hierarchy as one single Table,So its **easy** to perform “**storing** and r**etrieval**” of data.

  2.Any **changes** in the **attributes** of these classes only **affects** one **Single Table**.

  3.As there are **no joins** in querying/persisting the data,we can consider it as **Fastest** mapping model.

  4.It supports Polymorphic access.

  Disadvantages:

  1.It stores the data in Denormalized format,which is highly discouraged .

  2.Most of the columns will be inserted with **nulls**,which is not efficient.

- **Table Per SubClass**

  when to use:

  most of the time we will go for Table Per SubClass(If sub classes are less in number)

  Advantages:

  1.It stores the data in Normalized format.

  2 It supports Polymorphism accessing.

  Disadvantages:

  1.To perform persistency we have to perform multiple insert operations on super class table and sub class table.

  2.while querying the data using super class reference type,it will join all the sub classes in the hierarchy which will degrades the performance.

- **Table Per ConcreteClass**

  when to use:

  If the number of sub classes are more then we can go for Table Per ConcreteClass.

Advantages:
1.As Every concrete class has table,it is fast in perssisting the data for any classes in hierarchy.
2. It supports Polymorphism accessing.
Disadvantages:
1.we should ensure the ID is unique in the hierarchy To determine the sub classes.
2.A change in the super class will not only impact super class represented table rather all the tables representing the hierarchy will gets complicated.

- **ImplicitPolymorphism**

Advantages:
1.No expertise is required to work with ImplicitPolymorphism.
Disadvantages:
1.It doesnt support polymorphic access.

----------------Inheritance mapping models completed---------------------

09-08-2016

Association Mapping Models

Associations talks about dependancy betwen two classes.If a class tries to talk to the another class then we can said those those classes are associated with each other.Association can be in the following four forms and they are...
✔Association✔composition✔aggregation✔dependancy

1. **Association**
When it comes to Association we will not have Owner and ownee relationships. The life time of these two objects are independant on each other,and here we dont have any directional relationship.

2.**Aggregation**
When it comes to Association we will have Owner and ownee relationships. The life time of these two objects are not dependant on each other,here we have directional relationship.If the Owner dies then the Ownee will not die.

3.**composition**
It is called as highest degree of association, When it comes to composition we will have Owner and ownee relationships. The life time of these two objects are dependant on each other, If the owner dies then the ownee wil also dies.

4.dependancy
It is similar to Association only,Dependancy talks about who is dependant on whoom?.

We can broadly classify the Association relationships into two groups
1)Association Mapping
2)Composition Mapping(Component Mapping)
Lets try to work on the Association Mapping in the Unidirectional.

Association Mapping

The mapping of associations between entity classes and the relationships between tables is the soul of ORM. Following are the four ways in which the **cardinality** of the relationship between the objects can be expressed. An association mapping can be **unidirectional** as well as **bidirectional**.

1.One-to-One
2.One-to-Many
3.Many-to-One
4.Many-to-Many

10-08-2016

1.ManyToOne

Lets take an use case ot understand the concept of expressing OneToOne as ManyToMany relationship.
UseCase☹(CreditCard)
Lets Say we have an Account and for that Account we can have multiple CardHolders.For Every CardHolder there will be One CreditCard.
The CardHolder Entity Class will be as follows..

```
public class CardHolder {
                protected int cardHolderId;
                protected String firstName;
                protected String lastName;
                 protected String gender;
                 protected int age;
//CardHolder relationship with the AccountHolder(spouse,son and etc..)
                 protected String relationShip;
                 protected String mobile;
                 protected String email;
//Setters and Getters
}
```

Along with this we need to have Card as Entity class and it will be as follows..

```
public class Card {
                protected int cardId;
                protected String cardNumber;
                protected String type;
                protected int expiryMonth;
                protected int expiryYear;
                protected int cvv;
                protected String nameOnCard;
                protected Date issuedDate;
protected CardHolder holder;
//setters and getters
}
```

We can define the Association Between these two classes is as follows..
1.CardHolder-Owner,Card-Ownee
2.If the CardHolder dies,still the Card will exists.(unless we close explicitly)
3.As we are working with the unidirectional,Card will have CardHolder because One card belongs to One CardHolder.so we need to take the PK of the CARD_HOLDER as FK of the CARD Table.so we can declare CardHolder as an attribute in the Card class.
The Tables will looks as below..

```
 CARD_HOLDER
  |-CARD_HOLDER_ID(PK)
  |-FIRST_NAME
  |-LAST_NAME
  |-GENDER
  |-AGE
  |-RELATIONSHIP
  |-MOBILE
  |-EMAIL
CARD
  |-CARD_ID
  |-CARD_NUMBER
  |-CARD_TYPE
  |-EXPIRY_MONTH
  |-EXPIRY_YEAR
  |-CVV
  |-NAME_ON_CARD
  |-ISSUED_DT
  |-CARD_HOLDER_ID(FK,Unique,Not-null)
```

Inorder to persist the Entity classes we need to provide the mapping details to the Hibernate as follows..

Card.hbm.xml

```
<class name="Card" table="CARD">
                <id name="cardId" column="CARD_ID">
                        <generator class="increment" />
                </id>
                <property name="cardNumber" column="CARD_NUMBER" />
                <property name="type" column="CARD_TYPE" />
                <property name="expiryMonth" column="EXPIRY_MONTH" />
                <property name="expiryYear" column="EXPIRY_YEAR" />
                <property name="cvv" column="CVV" />
                <property name="nameOnCard" column="NAME_ON_CARD" />
                <property name="issuedDate" column="ISSUED_DT" />
```

//As it is PK column in CARD_HOLDER Table,and we wanted to store the data of CardHolder Entity object within CARD table.So we need to join with one FK which is in CARD Table.After joining(PK+FK) we will get **one** record of data,so the **cardinality** is **One**.so we need to provide the **PK Key Column** and **Class name**.

```
            <many-to-one name="holder" column="CARD_HOLDER_ID"
              class="CardHolder" unique="true" not-null="true" />
        </class>
</hibernate-mapping>
```

Working Example:

```
OneToOneAsManyToOne
|-src
   |-com.oto.entities
       |-Card.java
       |-Card.hbm.xml
       |-CardHolder.java
       |-CardHolder.hbm.xml
   |-com.oto.util
       |-HibernateUtil.java
   |-com.oto.test
       |-OTOTest.java
|-hibernate.cfg.xml
```

CardHolder.java

```
public class CardHolder {
                protected int cardHolderId;
                protected String firstName;
                protected String lastName;
```

```
                    protected String gender;
                    protected int age;
//CardHolder relationship with the AccountHolder(spouse,son and etc..)
                    protected String relationShip;
                    protected String mobile;
                    protected String email;
//Setters and Getters
//@toString();
}
```

Card.java

```
public class Card {
                   protected int cardId;
                   protected String cardNumber;
                   protected String type;
                   protected int expiryMonth;
                   protected int expiryYear;
                   protected int cvv;
                   protected String nameOnCard;
                   protected Date issuedDate;
protected CardHolder holder;
//setters and getters
//@tostring();
}
```

CardHolder.hbm.xml

```
<hibernate-mapping package="com.oto.entities">
         <class name="CardHolder" table="CARD_HOLDER">
                  <id name="cardHolderId" column="CARD_HOLDER_ID">
         <generator class="increment" />
                  </id>
                  <property name="firstName" column="FIRST_NM" />
                  <property name="lastName" column="LAST_NM" />
                  <property name="gender" column="GENDER" />
                  <property name="age" column="AGE" />
                  <property name="relationShip" column="RELATION_SHIP" />
                  <property name="mobile" column="MOBILE" />
                  <property name="email" column="EMAIL" />
         </class>
</hibernate-mapping>
```

Card.hbm.xml

```
<class name="Card" table="CARD">
                <id name="cardId" column="CARD_ID">
                        <generator class="increment" />
                </id>
                <property name="cardNumber" column="CARD_NUMBER" />
                <property name="type" column="CARD_TYPE" />
                <property name="expiryMonth" column="EXPIRY_MONTH" />
                <property name="expiryYear" column="EXPIRY_YEAR" />
                <property name="cvv" column="CVV" />
                <property name="nameOnCard" column="NAME_ON_CARD" />
                <property name="issuedDate" column="ISSUED_DT" />
```

//As it is PK column in CARD_HOLDER Table,and we wanted to store the data of CardHolder Entity object within CARD table.So we need to join with one FK which is in CARD Table.After joining(PK+FK) we will get **one** record of data,so the **cardinality** is **One**.so we need to provide the **PK Key Column** and **Class name**.

```
            <many-to-one name="holder" column="CARD_HOLDER_ID"
              class="CardHolder" unique="true" not-null="true" />
        </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
       <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/oto/entities/Card.hbm.xml" />
<mapping resource="com/tpcc/entities/CardHolder.hbm.xml" />
```

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

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);

        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

Now we can write Test class to check wether it is working or not?

```
public class OTOTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                CardHolder holder = null;
                Card card = null;
        try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();

                /*holder = new CardHolder();
                        holder.setFirstName("penchala");
                        holder.setLastName("Prasad");
                        holder.setAge(24);
```

```
                    holder.setGender("Male");
                    holder.setRelationShip("Father");
                    holder.setMobile("3938293733");
                    holder.setEmail("john@gmail.com");
        session.save(holder);
//when we have CardHolder then only we can create Card
        card = new Card();
                    card.setCardNumber("4321454534211231");
                    card.setType("Credit Card");
                    card.setExpiryMonth(9);
                    card.setExpiryYear(2017);
                    card.setCvv(453);
                    card.setIssuedDate(new Date());
                    card.setNameOnCard("penchala prasad");
                    card.setHolder(holder);
        session.save(card);
                    */
//Accessing the data.
                card = (Card) session.get(Card.class, 1);
                System.out.println(card);
                flag = true;
} finally {
                    if (transaction != null) {

                        if (flag) {
                        transaction.commit();
                    } else {
                        transaction.rollback();
                    }
                }
                    HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test:

case#1:session.save(card) then both Card_Id&CardHolder_Id will be persisted.

case#1-when we call card = (Card) session.get(Card.class, 1);
then both the details of CardHolder and Card will come.

11-08-2016:

OneToOneAsManyToOne(WorkingWith Annotation)

CardHolder.java

```
@Entity
@Table(name = "CARD_HOLDER")
public class CardHolder {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "CARD_HOLDER_ID")
        protected int cardHolderId;
        @Column(name = "FIRST_NM")
        protected String firstName;
        @Column(name = "LAST_NM")
        protected String lastName;
        protected String gender;
        protected int age;
        @Column(name = "RELATION_SHIP")
        protected String relationShip;
        protected String mobile;
        protected String email;
```

Card.java

```
@Entity
@Table(name = "CARD")
public class Card {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "CARD_ID")
protected int cardId;
        @Column(name = "CARD_NBR")
protected String cardNumber;
        @Column(name = "CARD_TYPE")
protected String type;
        @Column(name = "EXPIRY_MONTH")
protected int expiryMonth;
        @Column(name = "EXPIRY_YEAR")
protected int expiryYear;
protected int cvv;
       @Column(name = "NAME_ON_CARD")
protected String nameOnCard;
```

```
        @Column(name = "ISSUED_DT")
protected Date issuedDate;
@ManyToOne
@JoinColumn(name = "CARD_HOLDER_ID", unique = true, nullable = false)
protected CardHolder holder;
//Setters and Getters
}
```

Hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
                <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</property>
                <property name="connection.username">root</property>
                <property name="connection.password">root</property>
                <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
               <property name="show_sql">true</property>
                <property name="hbm2ddl.auto">update</property>
                <property
name="hibernate.current_session_context_class">thread</property>
                <mapping class="com.oto.entities.CardHolder"/>
                <mapping class="com.oto.entities.Card"/>
        </session-factory>
</hibernate-configuration>
```

Note:Remaining HibernateUtil,Test classes are same as in Xml mapping.

2.OneToOne

Lets say we have a Mobile class as follows..

```
public class Mobile {
                protected int mobileId;
                 protected String modelNo; (PK and FK)
                 protected String modelName;
        protected String description;
        protected String type;
        protected String manufacturer;
        protected float amount;
//setters and getters
}
```

And we have anothe class as Specification which looks as below..

```
public class Specification {
        protected int mobileId;
        protected String networkType;
        protected boolean dualSim;
        protected String mobileData;
        protected String processor;
        protected String storage;
        protected String ram;
//Here One Specification belongs to One Mobile only so within the
Specfication we need to declare the Mobile as an attribute
        protected Mobile mobile;
//setters and getters
}
```

The cardinality of the above Relation is “One”.

Now The Tables will looks as Below..

MOBILE
|-MOBILE_ID
|-MODEL_NO**(PK,FK)**
|-MODEL_NM
|-DESCR
|-MOBILE_TYPE
|-MANUFACTURER
|-AMOUNT

SPECIFICATIONS
|-MOBILE_ID
|-NETWORK_TYPE
|-DUAL_SIM_SUPPORT
|-MOBILE_DATA
|-PROCESSOR_TYPE
|-STORAGE_CAPACITY
|-RAM

(DQL)How to get/query the data per Associated Entity object Mobile:

→Inorder to query the data we need to identiy the FK,Here the PK itself is the FK,It usually happens when both the tables are in one-to-one relation.

“OneToOne Relationship means Our Table column PK itself is the FK of other Table.”

→So to the Hibernate we need to indicate the same by expressing that Relationship as one-to-one.

→By joining these two tables we need to fetch the data for mobile(which) associated entity object for that one we should specify the other side of the relationship as class=Mobile. (class is optioanl)

DML Operations:

→we dont need to map any association mapping while persisting the specification because the ID attribute itself will carry the data representing the primaryKey.

→It is recommended to use foreign generator to perssit id from relational associated entity objects.

Now lest write the mapping file as follows..

mobile.hbm.xml

```
<hibernate-mapping package="com.oto.entities">
        <class name="Mobile" table="MOBILE">
                <id name="mobileId" column="MOBILE_ID">
                        <generator class="increment" />
                </id>
<property name="modelNo" column="MODEL_NO" not-null="true" />
<property name="modelName" column="MODEL_NM" not-null="true" />
<property name="description" column="DESCR" />
<property name="type" column="MOBILE_TYPE" not-null="true" />
<property name="manufacturer" column="MANUFACTURER" not-null="true"
/>
<property name="amount" column="AMOUNT" not-null="true" />
      </class>
</hibernate-mapping>
```

specifications.hbm.xml

```
<hibernate-mapping package="com.oto.entities">
        <class name="Specification" table="SPECIFICATION">
                <id name="mobileId" column="MOBILE_ID">
                        <generator class="foreign">
//we need to provide the attribute name in specification class.
                                <param name="property">mobile</param>
                        </generator>
                </id>
     <property name="networkType" column="NETWORK_TYPE" not-
null="true" />
<property name="dualSim" column="DUAL_SIM_SUPPORT" not-null="true" />
<property name="mobileData" column="MOBILE_DATA" not-null="true" />
```

```
<property name="processor" column="PROCESSOR_TYPE" not-null="true" />
<property name="storage" column="STORAGE_CAPACITY" not-null="true" />
<property name="ram" column="RAM" not-null="true" />
        <one-to-one name="mobile" />
       </class>
</hibernate-mapping>
```

WorkingExample:

```
OneToOne
|-src
   |-com.oto.entities
      |-Mobile.java
      |-Mobile.hbm.xml
      |-Specifications.java
      |-Specifications.hbm.xml
   |-com.oto.util
      |-HibernateUtil.java
   |-com.oto.test
      |-OTOTest.java
|-hibernate.cfg.xml
```

Mobile.java

```
public class Mobile {
                protected int mobileId;
                 protected String modelNo; (PK and FK)
                 protected String modelName;
        protected String description;
        protected String type;
        protected String manufacturer;
        protected float amount;
//setters and getters
//@toString()
}
```

Specificatons.java

```
public class Specification {
        protected int mobileId;
        protected String networkType;
        protected boolean dualSim;
        protected String mobileData;
        protected String processor;
         protected String storage;
        protected String ram;
        protected Mobile mobile;
//setters and getters
//@tostring()
}
```

mobile.hbm.xml

```
<hibernate-mapping package="com.oto.entities">
        <class name="Mobile" table="MOBILE">
                <id name="mobileId" column="MOBILE_ID">
                        <generator class="increment" />
                </id>
<property name="modelNo" column="MODEL_NO" not-null="true" />
<property name="modelName" column="MODEL_NM" not-null="true" />
<property name="description" column="DESCR" />
<property name="type" column="MOBILE_TYPE" not-null="true" />
<property name="manufacturer" column="MANUFACTURER" not-null="true"
/>
<property name="amount" column="AMOUNT" not-null="true" />
      </class>
</hibernate-mapping>
```

specifications.hbm.xml

```
<hibernate-mapping package="com.oto.entities">
        <class name="Specification" table="SPECIFICATION">
                <id name="mobileId" column="MOBILE_ID">
                        <generator class="foreign">
//we need to provide the attribute name in specification class.
                                <param name="property">mobile</param>
                        </generator>
                </id>
     <property name="networkType" column="NETWORK_TYPE" not-
null="true" />
<property name="dualSim" column="DUAL_SIM_SUPPORT" not-null="true" />
<property name="mobileData" column="MOBILE_DATA" not-null="true" />
<property name="processor" column="PROCESSOR_TYPE" not-null="true" />
<property name="storage" column="STORAGE_CAPACITY" not-null="true" />
<property name="ram" column="RAM" not-null="true" />
       <one-to-one name="mobile" />
      </class>
</hibernate-mapping>
```

Hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/oto/entities/Mobile.hbm.xml" />
<mapping resource="com/oto/entities/Specification.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);

        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

OTOTest.java

```
public class OTOTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                Mobile mobile = null;
                Specification specification = null;
          try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
        /*mobile = new Mobile();
                        mobile.setModelNo("m-9372");
                        mobile.setModelName("Samsung Galaxy S7");
                        mobile.setDescription("Galaxy series");
                        mobile.setManufacturer("Samsung");
                        mobile.setType("smart");
                        mobile.setAmount(55448.0f);
                session.save(mobile);
        specification = new Specification();
                        specification.setDualSim(false);
                        specification.setMobileData("LTE");
                        specification.setNetworkType("GSM");
                        specification.setProcessor("Snapdragon");
                        specification.setStorage("32 gb");
                        specification.setRam("4 gb");
                        specification.setMobile(mobile);
              session.save(specification);*/
//Accessing the data
specification = (Specification) session.get(Specification.class, 1);
                System.out.println(specification);
        flag = true;
} finally {
                        if (transaction != null) {

                                if (flag) {
                                transaction.commit();
                        } else {
                                transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test☹(permutations &combinations)
case#1-storing the data.
case#2-accessing the data.

12-08-2016

OneToOne Annotations:

Mobile.java

```
@Entity
@Table(name = "MOBILE")
public class Mobile {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "MOBILE_ID")
    protected int mobileId;
    @Column(name = "MODEL_NO")
    protected String modelNo;
    @Column(name = "MODEL_NM")
    protected String modelName;
    protected String description;
    protected String type;
    protected String manufacturer;
     protected float amount;
//setters and getters
//@toString
}
```

Specifications.java

```
@Entity
@Table(name = "SPECIFICATION")
public class Specification {
        @Id
        @GenericGenerator(name = "foreign_generator", strategy = "foreign",
parameters = { @Parameter(name = "property", value = "mobile") })
        @GeneratedValue(generator = "foreign_generator")
        protected int mobileId;
        protected String networkType;
        protected boolean dualSim;
        protected String mobileData;
        protected String processor;
        protected String storage;
        protected String ram;
        @OneToOne
        @PrimaryKeyJoinColumn
        protected Mobile mobile;
//setters and getters
//@toString
}
```

Hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
                <property
name="connection.driver_class">com.mysql.jdbc.Driver</property>
                <property
name="connection.url">jdbc:mysql://localhost:3306/hib_tools</property>
                <property name="connection.username">root</property>
                <property name="connection.password">root</property>
                <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
               <property name="show_sql">true</property>
                <property name="hbm2ddl.auto">update</property>
                <property
name="hibernate.current_session_context_class">thread</property>
                <mapping class="com.oto.entities.Mobile"/>
                <mapping class="com.oto.entities.Specifications"/>
        </session-factory>
```

</hibernate-configuration>
Note:Remaining HibernateUtil,Test classes are same as in Xml mapping.

13-08-2016

One-to-ManySet

UseCase:

✓Lets say we have the Entity classes **Property** and **Approval**.The **Relationship** between these two classes is **Aggregation**(because approval will exists irrespective of the property).
✓The **cardinality** of this relationship is **One-to-Many**(because a property may have multiple approvals)
✓To represent **One-to-Many** association **cardinality** between the Entity classes we should not declare one Approval rather we need collection of Approvals.
✓In Java we have Three types of Collections as List,Set,Map.
✓when we dont want our child to be duplicate and if we wanted to appear it once only then we should use Set as collection type.
✓Lets try to declare the approvals of a property class using **set** collection because **Approvals** should **not** gets **duplicated** for a single property.
✓To distinguish between the unique Approvals and duplicated Approvals we need to overrride equals() and hashCode() in the Entity class.
Now The **Property** class will looks like as below...

```
public class Property {
                protected int propertyId;
                protected String propertyName;
                protected String description;
                protected String type;
                protected int area;
                protected String location;
                protected float amount;
               protected Set<Approval> approvals;
//setters and getters
//@toString
}
```

and The **Approval** class will looks as follows.....

```
public class Approval {
          protected int approvalId;
          protected String approvalName;
          protected String description;
          protected Date issuedDate;
          protected String issuedAuthority;
```

```
        protected Property property;
//setters and getters
//@hashCode,@equals
//@toString
}
```
The Tables will looks as below for the Above two classes.

PROPERTY
|-PROPERTY_ID(**pk**)
|-PROPERTY_NM
|-DESCR
|-PROPERTY_TYPE
|-AREA
|-LOCATION
|-AMOUNT

APPROVAL
|-APPROVAL_ID
|-APPROVAL_NM
|-DESCR
|-ISSUED_DT
|-ISSUED_AUTHORITY
|-PROPERTY_ID(**fk**)

If we wanted to store the Approval,As the approval has to have the property_id so within the Approval class we will represent Property as an attribute.

DQL(For property with associated approvals)

✔To the Hibernate we need to tell to Join PK of Our Table with FK of another Table,which returns many child records which we need to store in the Set.

✔To the Hibernate we need to tell that we have an one attribute which is associated object,The type of the object is set.For this set<approvals> we need to join the Key of our table with the FK of other table.So that we will get many approvals.

DML

✔As This association is unidirectional so the Child Table FK column should be nullable.

✔First persist the child records with FK as null,then create parent,associate the child objects to the parent Entity collection.

✔while persisting the parent Hibernate will pick the associated child and updates the relationaship column of the child with the key of the parent.

Here we need to write the association mapping for property.

Lets write the mapping files to meet the above requirements..

property.hbm.xml

```
<hibernate-mapping package="com.otm.entities">
        <class name="Property" table="PROPERTY">
                <id name="propertyId" column="PROPERTY_ID">
                        <generator class="increment" />
                </id>
                <property name="propertyName" column="PROPERTY_NM" />
                <property name="description" column="DESCR" />
                <property name="type" column="PROPERTY_TYPE" />
                <property name="area" column="AREA" />
               <property name="location" column="LOCATION" />
<property name="amount" column="AMOUNT" />
              <set name="approvals" inverse="true">
                <key column="PROPERTY_ID" not-null="true" />
                <one-to-many class="Approval" />
                </set>
        </class>
</hibernate-mapping>
```

Approval.hbm.xml

```
<hibernate-mapping package="com.otm.entities">
        <class name="Approval" table="APPROVAL">
                <id name="approvalId" column="APPROVAL_ID">
                        <generator class="increment" />
                </id>
                <property name="approvalName" column="APPROVAL_NM" />
                <property name="description" column="DESCR" />
                <property name="issuedDate" column="ISSUED_DT" />
        <property name="issuedAuthority" column="ISSUED_AUTHORITY" />
                <many-to-one column="PROPERTY_ID" name="property"/>
        </class>
</hibernate-mapping>
```

This is how we will perform one-to-many mapping Association.

14-08-2016& 15-08-2016

Working Example:

```
OneToManySet
|-src
      |-com.otm.entities
          |-Property.java
          |-Approval.java
          |-Property.hbm.xml
          |-Approval.hbm.xml
     |-com.otm.util
          |-HibernateUtil.java
    |-com.otm.test
         |-OTMSTest.java
|-hibernate.cfg.xml
```

Property.java

```
public class Property {
                    protected int propertyId;
                    protected String propertyName;
                    protected String description;
                    protected String type;
                    protected int area;
                    protected String location;
                    protected float amount;
                   protected Set<Approval> approvals;
//setters and getters
//@toString
}
```

Approval.java

```
public class Approval {
            protected int approvalId;
            protected String approvalName;
            protected String description;
            protected Date issuedDate;
            protected String issuedAuthority;
            protected Property property;
//setters and getters
//@hashCode,@equals
//@toString
}
```

property.hbm.xml

```
<hibernate-mapping package="com.otm.entities">
        <class name="Property" table="PROPERTY">
                <id name="propertyId" column="PROPERTY_ID">
                        <generator class="increment" />
                </id>
                <property name="propertyName" column="PROPERTY_NM" />
                <property name="description" column="DESCR" />
                <property name="type" column="PROPERTY_TYPE" />
                <property name="area" column="AREA" />
               <property name="location" column="LOCATION" />
<property name="amount" column="AMOUNT" />
              <set name="approvals" inverse="true">
                <key column="PROPERTY_ID" not-null="true" />
                <one-to-many class="Approval" />
                </set>
        </class>
</hibernate-mapping>
```

Approval.hbm.xml

```
<hibernate-mapping package="com.otm.entities">
        <class name="Approval" table="APPROVAL">
                <id name="approvalId" column="APPROVAL_ID">
                        <generator class="increment" />
                </id>
                <property name="approvalName" column="APPROVAL_NM" />
                <property name="description" column="DESCR" />
                <property name="issuedDate" column="ISSUED_DT" />
        <property name="issuedAuthority" column="ISSUED_AUTHORITY" />
                <many-to-one column="PROPERTY_ID" name="property"/>
        </class>
</hibernate-mapping>
```

hibernate.cfg.xml

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

```
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/oto/entities/Property.hbm.xml" />
<mapping resource="com/oto/entities/Approval.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
	private static SessionFactory sessionFactory;
static {
	Configuration configuration = new Configuration().configure();
	StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
		builder.applySettings(configuration.getProperties());
		StandardServiceRegistry registry = builder.build();
		sessionFactory =
configuration.buildSessionFactory(registry);

	}
public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}
public static void closeSessionFactory() {
	if (sessionFactory != null) {
		sessionFactory.close();
		  }
```

OTMSTest.java

```
public class OTMSetTest {
	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Transaction transaction = null;
		Session session = null;
		boolean flag = false;
		Approval approval = null;
		Property property = null;
		Set<Approval> approvals = null;
```

```
        try {

                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
        approvals = new HashSet<Approval>();
                        property = new Property();
                        property.setPropertyName("Cyber Zone");
                        property.setDescription("Gated Villas");
                        property.setType("Gated");
                        property.setArea(3552);
                        property.setLocation("Madhapur");
                        property.setAmount(3837232.34f);
                        property.setApprovals(approvals);
        session.save(property);
       approval = new Approval();
                        approval.setApprovalName("Waterboard approval");
                        approval.setDescription("GHMC Approval");
                        approval.setIssuedAuthority("GHMC");
                        approval.setIssuedDate(new Date());
                        approval.setProperty(property);
                        approvals.add(approval);
        session.save(approval);
        approval = new Approval();
                        approval.setApprovalName("Electricity Connection");
                        approval.setDescription("Society Electricity Connection");
                        approval.setIssuedAuthority("TSDPCL");
                        approval.setIssuedDate(new Date());
                        approval.setProperty(property);
        approvals.add(approval);
        session.save(approval);
//accessing the data
                /*property = (Property) session.get(Property.class, 1);
        System.out.println(property);*/
                /*approval = (Approval) session.get(Approval.class, 1);
        System.out.println(approval.getProperty().getDescription());*/
                flag = true;
} finally {
                        if (transaction != null) {
                                if (flag) {
                                transaction.commit();
                        } else {
                                transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test☹(permutations &combinations)
case#1-storing the data.
case#2-accessing the data.
OneToManySet **Annotation** Example:
property.java

```
@Entity
@Table(name = "PROPERTY")
public class Property {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "PROPERTY_ID")
        protected int propertyId;
        @Column(name = "PROPERTY_NM")
        protected String propertyName;
        @Column(name = "DESCR")
        protected String description;
        protected String type;
        protected int area;
        protected String location;
        protected float amount;
        @OneToMany(mappedBy = "property")
        protected Set<Approval> approvals;
//setters and getters
//@toString()
}
```

Approval.java

```
@Entity
@Table(name = "APPROVAL")
public class Approval {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	@Column(name = "APPROVAL_ID")
	protected int approvalId;
	@Column(name = "APPROVAL_NM")
	protected String approvalName;
	@Column(name = "DESCR")
	protected String description;
	@Column(name = "ISSUED_DT")
	protected Date issuedDate;
	@Column(name = "ISSUED_AUTHORITY")
	protected String issuedAuthority;   @ManyToOne
	@JoinColumn(name = "PROPERTY_ID", nullable = false)
	protected Property property;
//setters and getters
//@toString()
}
```

Note:Remaining are same.

OneToMany List

UseCase:

✔Lets say we have **Doctor** and **Appointmen**t Entity classes.The cardinality between these two clases is one-To-many.

✔Here we wanted to map the association from Doctor to the Patient ,because one doctor will have List of appointments.(appointmenst should be in order and can be duplicated).

✔The Entity classes will looks like as below..........

```
public class Doctor {
	protected int doctorId;
	protected String name;
	protected String specialization;
	protected String qualification;
	protected int experience;
	protected String mobile;
	protected String email;
	protected List<Appointment> appointments;
//setters and getters
//@toString
}
public class Appointment {
        protected int appointmentId;
        protected Date appointmentDate;
        protected String patientName;
        protected String contactNo;
        protected String comments;
//setters and getters
//@toString
}
```

Tables for these Two classes will looks as below...

```
DOCTOR
|-DOCTOR_ID(PK)
|-NAME
|-SPECIALIZATION
|-QUALIFICATION
|-EXPERIENCE
|-MOBILE
|-EMAIL
|- APPOINTMENT_SEQ_NO
APPOINTMENT
|-APPOINTMENT_ID
|-APPOINTMENT_DT
|-PATIENT_NM
|-CONTACT_NO
|-COMMENTS
|- DOCTOR_ID(FK)
```

→If we tried creating the Doctor then the concerned details should be stored DOCTOR table,If we tried creating the Three Appointments for the doctor then those three should be stored in APPOINTMENT Table.

→If we wanted to access the Doctor,then these Appointment should come in an order in which the Appointments got inserted.

→So here we need an additional column APPOINTMENT_SEQ_NO in the DOCTOR Table to keep track the insertion order of Appointments.

16-08-2016&17-08-2016

WorkingExample:

```
OneToManyList
|-src
   |-com.otm.entities
         |-Doctor.java
         |-Doctor.hbm.xml
         |-Appointment.java
         |-Appointment.hbm.xml
   |-com.otm.util
         |-HibernateUtil.java
   |-com.otm.test
         |-OTMLTest.java
|-hibernate.cfg.xml
```

Doctor.java

```
public class Doctor {
        protected int doctorId;
        protected String name;
        protected String specialization;
        protected String qualification;
        protected int experience;
        protected String mobile;
        protected String email;
        protected List<Appointment> appointments;
//setters and getters
//@toString
}
```

Appointment.java

```
public class Appointment {
        protected int appointmentId;
        protected Date appointmentDate;
        protected String patientName;
        protected String contactNo;
        protected String comments;
//setters and getters
//@toString
}
```

Mapping files:

Appointment.hbm.xml

```
<hibernate-mapping package="com.otml.entities">
	<class name="Appointment" table="APPOINTMENT">
		<id name="appointmentId" column="APPOINTMENT_ID">
			<generator class="increment" />
		</id>
		<property name="appointmentDate" 
column="APPOINTMENT_DT" />
		<property name="patientName" column="PATIENT_NM" />
		<property name="contactNo" column="CONTACT_NO" />
		<property name="comments" column="COMMENTS" />
	</class>
</hibernate-mapping>
```

Doctor.hbm.xml

```
<hibernate-mapping package="com.otml.entities">
	<class name="Doctor" table="DOCTOR">
		<id name="doctorId" column="DOCTOR_ID">
			<generator class="increment" />
		</id>
		<property name="name" column="NAME" />
		<property name="specialization" column="SPECIALIZATION" />
		<property name="qualification" column="QUALIFICATION" />
		<property name="experience" column="EXPERIENCE" />
		<property name="mobile" column="MOBILE" />
		<property name="email" column="EMAIL" />
		<list name="appointments">
			<key column="DOCTOR_ID" />
			<list-index column="APPOINTMENT_SEQ_NO" base="1"/>
			<one-to-many class="Appointment" />
		</list>
	</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/oto/entities/Doctor.hbm.xml" />
<mapping resource="com/oto/entities/Appointment.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);

        }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

OTMLTest.java

```
public class OTMLTest {
	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Transaction transaction = null;
		Session session = null;
		boolean flag = false;
		Doctor doctor = null;
		Appointment appointment = null;
		List<Appointment> appointments = null;
	try {
			sessionFactory = HibernateUtil.getSessionFactory();
			session = sessionFactory.getCurrentSession();
			transaction = session.beginTransaction();
	appointment = new Appointment();
			appointment.setAppointmentDate(new Date());
			appointment.setPatientName("Sam");
			appointment.setContactNo("92337234");
			appointment.setComments("Emergency");
	session.save(appointment);
		appointments = new ArrayList<Appointment>();
		appointments.add(appointment);
		appointment = new Appointment();
			appointment.setAppointmentDate(new Date());
			appointment.setPatientName("Rod");
			appointment.setContactNo("92337353234");
			appointment.setComments("Regular Checkup");
	session.save(appointment);
		appointments.add(appointment);
		doctor = new Doctor();
			doctor.setName("Raghupati");
			doctor.setSpecialization("Cardiac");
			doctor.setQualification("MS");
			doctor.setExperience(35);
			doctor.setEmail("raghupati@gmail.com");
			doctor.setMobile("3938394");
			doctor.setAppointments(appointments);
	session.save(doctor);
```

```
            appointment = new Appointment();
                  appointment.setAppointmentDate(new Date());
                   appointment.setPatientName("Mathew");
                   appointment.setContactNo("92234");
                   appointment.setComments("Regular Checkup");
         session.save(appointment);
            appointments = new ArrayList<Appointment>();
         appointments.add(appointment);
            doctor = new Doctor();
                   doctor.setName("K Y reddy");
                   doctor.setSpecialization("Orthopetic");
                   doctor.setQualification("DH");
                   doctor.setExperience(35);
                   doctor.setEmail("kyreddy@gmail.com");
                   doctor.setMobile("3938394");
                   doctor.setAppointments(appointments);
         session.save(doctor);
//Accessing the data
                   /*doctor = (Doctor) session.get(Doctor.class, 1);
         System.out.println(doctor);*/
              flag = true;
} finally {
                   if (transaction != null) {
                        if (flag) {
                        transaction.commit();
                   } else {
                        transaction.rollback();
                   }
              }
                   HibernateUtil.closeSessionFactory();
              }
         }
}
```

Test☹(permutations &combinations)
case#1-storing the data.
case#2-accessing the data.

OneToManyList **Annotation** Example:

Appointment.java

```
@Entity
@Table(name = "APPOINTMENT")
public class Appointment {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "APPOINTMENT_ID")
        protected int appointmentId;
        @Column(name = "APPOINTMENT_DT")
        protected Date appointmentDate;
        @Column(name = "PATIENT_NM")
        protected String patientName;
        @Column(name = "CONTACT_NO")
        protected String contactNo;
        protected String comments;
//setters and getters
//@toString() }
```

Doctor.java

```
@Entity
@Table(name = "DOCTOR")
public class Doctor {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "DOCTOR_ID")
        protected int doctorId;
        protected String name;
        protected String specialization;
        protected String qualification;
        protected int experience;
        protected String mobile;
        protected String email;
        @OneToMany
        @JoinColumn(name = "DOCTOR_ID")
        @OrderColumn(name = "APP_SEQ_NO")
        protected List<Appointment> appointments;
//setters and getters,//@toString
}
```

Note:Remaining hibernate.cfg.xml and util class, and Test classes are same.

18-08-2016

OneToManyMap

UseCase

🚶 Lets say we have Two Entity classes **Incident** and **SupportEngineer**.

🚶The SupportEngineer Entity class will be as follows.......

```
public class SupportEngineer {
        protected int supportEngineerId;
        protected String name;
        protected String designation;
        protected int level;
        protected String contactNo;
        protected String email;
        protected Map<String, Incident> assignedIncidents;
//setters and getters
}
```

🚶The Incident Entity class will be as follows.......

```
public class Incident {
                protected int incidentId;
                protected String description;
                protected Date reportedDate;
                protected int priority;
                protected String status;
//setters and getters
}
```

🚶The **Association** between these two classes is **one-to-many**,and we wanted to map SupportEngineer to Incident,so we declared Map collection in the SupportEngineer Entity class.

🚶The Tables will looks like as below....

SUPPORT_ENGINEER

```
|- SUPPORT_ENGINEER_ID(Primary key column)
|- NAME
|- DESIGNATION
|- ENGINEER_LEVEL
|- CONTACT_NO
|- EMAIL
```

INCIDENT
|- INCIDENT_ID
|- DESCR
|- REPORTED_DATE
|- PRIORITY
|- STATUS
|- CASE_LOG_ID(Map key column)
|- SUPPORT_ENGINEER_ID(Foreign key column)

🏃 when we assign the Incident to the SupportEngineer then only he will populate the CASE_LOG_ID.
🏃 As it is unidirectional so the case_log_id is nullable only.
🏃 The mapping files for the above entity classes as follows..

DQL

🏃To the Hibernate we need to tell that"Join the PK of SupportEngineer table with the FK of Incident Table"
🏃 Once Hibernate fetch the child records,to the Hibernate we need to tell that " in the Incident(child) record CASE_LOG_ID column acts as key column for the Map,and the value for the Map is represented by Incident(child) Entity object.
🏃 The Mapping file for SupportEngineer will be as written below...

```
<hibernate-mapping package="com.otmm.entities">
        <class name="SupportEngineer" table="SUPPORT_ENGINEER">
                <id name="supportEngineerId"
column="SUPPORT_ENGINEER_ID">
                        <generator class="increment" />
                </id>
                <property name="name" column="NAME" />
                <property name="designation" column="DESIGNATION" />
                <property name="level" column="ENGINEER_LEVEL" />
                <property name="contactNo" column="CONTACT_NO" />
                <property name="email" column="EMAIL" />
<map name="assignedIncidents" cascade="all">
                <key column="SUPPORT_ENGINEER_ID" />
                <map-key column="CASE_LOG_ID" type="string" length="50" />
                    <one-to-many class="Incident" />
                </map>
        </class>
</hibernate-mapping>
```

Working Example☹(XML)

```
OneToManyMap
|-src
   |-com.otm.entities
           |-Incident.java
           |-Incident.hbm.xml
           |-SupportEngineer.java
           |-SupportEngineer.hbm.xml
   |-com.otm.util
           |-HibernateUtil.java
   |-com.otm.test
           |-OTMMTest.java
|-hibernate.cfg.xml
```

Incident.java

```
public class Incident {
                protected int incidentId;
                protected String description;
                protected Date reportedDate;
                protected int priority;
                protected String status;
//setters and getters
}
```

SupportEngineer.java

```
public class SupportEngineer {
        protected int supportEngineerId;
        protected String name;
        protected String designation;
        protected int level;
        protected String contactNo;
        protected String email;
        protected Map<String, Incident> assignedIncidents;
//setters and getters
}
```

Incident.java

```
<hibernate-mapping package="com.otmm.entities">
        <class name="Incident" table="INCIDENT">
                <id name="incidentId" column="INCIDENT_ID">
                        <generator class="increment" />
                </id>
                <property name="description" column="DESCR" />
                <property name="reportedDate" column="REPORTED_DATE" />
                <property name="priority" column="PRIORITY" />
                <property name="status" column="STATUS" />
        </class>
</hibernate-mapping>
```

SupportEngineer.hbm.xml

```
<hibernate-mapping package="com.otmm.entities">
        <class name="SupportEngineer" table="SUPPORT_ENGINEER">
                <id name="supportEngineerId"
column="SUPPORT_ENGINEER_ID">
                        <generator class="increment" />
                </id>
                <property name="name" column="NAME" />
                <property name="designation" column="DESIGNATION" />
                <property name="level" column="ENGINEER_LEVEL" />
                <property name="contactNo" column="CONTACT_NO" />
                <property name="email" column="EMAIL" />
<map name="assignedIncidents" cascade="all">
                <key column="SUPPORT_ENGINEER_ID" />
                <map-key column="CASE_LOG_ID" type="string" length="50" />
                    <one-to-many class="Incident" />
                </map>
        </class>
</hibernate-mapping>
```

Hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/oto/entities/Incident.hbm.xml" />
<mapping resource="com/oto/entities/SupportEngineer.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
            }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
                  }
```

OTMMTest.java

```
public class OTMMTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                Incident incident = null;
                SupportEngineer supportEngineer = null;
                Map<String, Incident> assignedIncidents = null;
        try {
                sessionFactory = HibernateUtil.getSessionFactory();
                session = sessionFactory.getCurrentSession();
                transaction = session.beginTransaction();
assignedIncidents = new HashMap<String, Incident>();
                        incident = new Incident();
                        incident.setDescription("Lost/Stolen Card");
                        incident.setPriority(1);
                        incident.setReportedDate(new Date());
                        incident.setStatus("new");
assignedIncidents.put("CASE_LOG_1", incident);
       incident = new Incident();
                        incident.setDescription("Report mis-use of card");
                        incident.setPriority(1);
                        incident.setReportedDate(new Date());
                        incident.setStatus("new");
assignedIncidents.put("CASE_LOG_2", incident);
        supportEngineer = new SupportEngineer();
                 supportEngineer.setName("Rakesh");
                 supportEngineer.setDesignation("Engineer");
                 supportEngineer.setLevel(1);
                 supportEngineer.setContactNo("3393844");
                  supportEngineer.setEmail("rakesh@gmail.com");
                 supportEngineer.setAssignedIncidents(assignedIncidents);
        session.save(supportEngineer);
flag = true;
} finally {
                        if (transaction != null) {
```

```
                                        if (flag) {
                                        transaction.commit();
                              } else {
                                        transaction.rollback();
                              }
                    }
                              HibernateUtil.closeSessionFactory();
                    }
          }
}
```

Test☹(permutations &combinations)
case#1-storing the data.
case#2-accessing the data.

<u>Working with One-To-ManyMap(Annotations)</u>
Incident.java

```
@Entity
@Table(name = "INCIDENT")
public class Incident {
          @Id
          @GeneratedValue(strategy = GenerationType.IDENTITY)
          @Column(name = "INCIDENT_ID")
          protected int incidentId;
          @Column(name = "DESCR")
          protected String description;
          @Column(name = "REPORTED_DT")
          protected Date reportedDate;
          protected int priority;
          protected String status;
//setters and getters
//@toString()
}
```

SupportEnginner.java

```
@Entity
@Table(name = "SUPPORT_ENGINEER")
public class SupportEngineer {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "SUPPORT_ENGINEER_ID")
        protected int supportEngineerId;
        protected String name;
        protected String designation;
        @Column(name = "SUPPORT_ENGINEER_LEVEL")
        protected int level;
        @Column(name = "CONTACT_NO")
        protected String contactNo;
        protected String email;
 @OneToMany(cascade = CascadeType.ALL)
        @JoinColumn(name = "SUPPORT_ENGINEER_ID", nullable = true)
        @MapKeyColumn(name = "CASE_LOG_ID", nullable = true)
        protected Map<String, Incident> assignedIncidents;
//setteers and getters
//@toString
}
```

Note: Remaianing classes are same as XML Mapping.

Cascade Attributes in Hibernate

✓Main concept of hibernate relations is to getting the relation between parent and child class objects.

✓Cascade attribute is mandatory, when ever we apply relationship between objects, cascade attribute transfers operations done on one object onto its related child objects.

✓The "Cascade" keyword is often appear on the collection mapping to manage the state of the collection automatically.

✓Hibernate Cascade Options**:**There are mainly 6 types of cascade options

1.all
2.none
3.save-update
4.delete
5.delete-orphan
6.all-delete-orphan

Let Assume: Stock (as Parent table) and StockDailyRecord (as Child table)

cascade=”all”

If we write cascade = “all” then changes at parent class object will be effected to child class object too,all operations like insert, delete, update at parent object will be effected to child object also

cascade=”none”(Default Value)

cascade =”none” means no operations will be transfers to the child class.

if we apply insert,update,delete operation on parent class object, then child class objects will not be effected, if cascade = “none”

cascade=”save-update”

when you save the ‘Stock’, it will “cascade” the save operation to it’s referenced ‘stockDailyRecords’ and save both into database automatically.

cascade=”delete”

When you delete the ‘Stock’, all its reference ‘stockDailyRecords’ will be deleted automatically.

cascade=”delete-orphan”

In short, delete-orphan allow parent table to delete few records (delete orphan) in its child table.

```
StockDailyRecord sdr1 =
(StockDailyRecord)session.get(StockDailyRecord.class,new Integer(56));
StockDailyRecord sdr2 =
(StockDailyRecord)session.get(StockDailyRecord.class,new Integer(57));
Stock stock = (Stock)session.get(Stock.class, new Integer(2));
stock.getStockDailyRecords().remove(sdr1);
stock.getStockDailyRecords().remove(sdr2);
session.saveOrUpdate(stock);
```

cascade="all-delete-orphan"

In an application, if a child record is removed from the collection and if we want to remove that child record immediately from the database, then we need to set the cascade ="all-delete-orphan"

Note: orphan record means it is a record in child table but it doesn't have association with its parent in the application.

19-08-2016

Many-to-Many

✓In this Relationship One parent can contain multiple childs,similarly one child can contain multiple parents.

✓Lets say we have two Entity classes **Journey** and **Passenger**.

```
public class Journey {
        protected int journeyId;
        protected String source;
        protected String destination;
        protected Date journeyDate;
        protected float fare;
        protected int distance;
        protected Set<Passenger> passengers;
//setters and getters
}
public class Passenger {
        protected int passengerId;
        protected String name;
        protected int age;
        protected String gender;
        protected String mobile;
        protected String emergencyContactNo;
        protected Set<Journey> journeys;
//setters and getters
}
```

✓The Association between these two classes is **aggregation**,because passenger can still exists in the system even he hasn't travelled.

✓The **cardinality** between these two classes is **many-to-Many**,because one journey can have multiple passangers and one passanger can travel in multiple journeys.

✓we will represent this relationship in java using **collections** on **both sides**.

✓The corresponding Tables will looks like as below…
JOURNEY
|-JOURNEY_ID(PK)
|-SOURCE
|-DEST
|-JOURNEY_DT
|-FARE
|-DISTANCE
PASSENGER
|-PASSENGER_ID(PK)
|-NAME
|-AGE
|-GENDER
|-MOBILE
|-EMERGENCY_CONTACT_NO
✓To represent the many-to-Many relationship always we need to have the Third Table which contains the Two Tables PK's as Key Columns.
JOURNEY_PASSENGER
|-JOURNEY_ID(Fk)
|-PASSENGER_ID(fk)
<u>DQL-How to get set of passengers:</u>
1.Find the cardinality,if it is many-to-many both the tables PK's will become FK in third Table.
2.To get the set of passengers first we need to join the pK of Journey with the corresponding Fk in third Table.So that we get many-column of the Relationship.
3.In turn this many-column should be joined with PK of other table to get related records of passengers.
<u>DML-How to store Journey</u>
1.When we are trying to persist the associated object,two insert will happen to represent the relationship.
2.First it perssit teh journey into journey table and goes to the associated child objects and makes an insert operation into the Third Table by storing both the Entity PK's as FK's to represent relationship.
✓Lest see how the mapping files will looks like…
```
<hibernate-mapping package="com.mtms.entities">
        <class name="Journey" table="JOURNEY">
                <id name="journeyId" column="JOURNEY_ID">
```

```
                    <generator class="increment" />
            </id>
            <property name="source" column="SOURCE" />
            <property name="destination" column="DEST" />
            <property name="journeyDate" column="JOURNEY_DT" />
            <property name="fare" column="FARE" />
            <property name="distance" column="DISTANCE" />
         <set name="passengers" table="JOURNEY_PASSENGER"
cascade="save-update">
                    <key column="JOURNEY_ID" not-null="true" />
        <many-to-many column="PASSENGER_ID" class="Passenger" />
            </set>
        </class>
</hibernate-mapping>
```

<u>WorkingExample:</u>

```
ManyToManySet
|-src
   |-com.mtm.entites
         |-Journey.java
         |-Journey.hbm.xml
         |-Passenger.java
         |-Passenger.hbm.xml
   |-com.mtm.util
        |-HibernateUtil.java
   |-com.mtm.test
        |-MTMSTest.java
|-hibernate.cfg.xml
```

Journey.java

```
public class Journey {
        protected int journeyId;
        protected String source;
        protected String destination;
        protected Date journeyDate;
        protected float fare;
        protected int distance;
        protected Set<Passenger> passengers;
//setters and getters
//@toString
}
```

Passenger.java

```
public class Passenger {
        protected int passengerId;
        protected String name;
        protected int age;
        protected String gender;
        protected String mobile;
        protected String emergencyContactNo;
        protected Set<Journey> journeys;
//setters and getters
//@toString
}
```

Journey.hbm.xml

```
<hibernate-mapping package="com.mtms.entities">
        <class name="Journey" table="JOURNEY">
                <id name="journeyId" column="JOURNEY_ID">
        <generator class="increment" />
                </id>
                <property name="source" column="SOURCE" />
                <property name="destination" column="DEST" />
                <property name="journeyDate" column="JOURNEY_DT" />
                <property name="fare" column="FARE" />
                <property name="distance" column="DISTANCE" />
          <set name="passengers" table="JOURNEY_PASSENGER"
cascade="save-update">
                        <key column="JOURNEY_ID" not-null="true" />
        <many-to-many column="PASSENGER_ID" class="Passenger" />
                </set>
        </class>
</hibernate-mapping>
```

Passenger.hbm.xml

```
<hibernate-mapping package="com.mtms.entities">
        <class name="Passenger" table="PASSENGER">
                <id name="passengerId" column="PASSENGER_ID">
                        <generator class="increment" />
                </id>
```

```
            <property name="name" column="NAME" />
            <property name="age" column="AGE" />
            <property name="gender" column="GENDER" />
            <property name="mobile" column="MOBILE" />
        <property name="emergencyContactNo"
column="EMERGENCY_CONTACT_NO" />
            <set name="journeys" table="JOURNEY_PASSENGER">
        <key column="PASSENGER_ID" />
                <many-to-many column="JOURNEY_ID" class="Journey" />
            </set>
        </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
       <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/mtm/entities/Journey.hbm.xml" />
<mapping resource="com/mtm/entities/Passenger.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
            }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
           }
}
```

MTMSTest.java

```
public class MTMSTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                Journey journey = null;
                Passenger passenger = null;
                Set<Passenger> passengers = null;
         try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
        passengers = new HashSet<Passenger>();
                        passenger = new Passenger();
                        passenger.setName("Subbarao");
                        passenger.setGender("Male");
                        passenger.setAge(24);
                        passenger.setMobile("3938273");
                        passenger.setEmergencyContactNo("000000000");
        passengers.add(passenger);
        passenger = new Passenger();
                        passenger.setName("Aishwaraya");
                        passenger.setGender("Female");
                        passenger.setAge(2);
                        passenger.setMobile("39383273");
                        passenger.setEmergencyContactNo("108");
        passengers.add(passenger);
        journey = new Journey();
                        journey.setSource("Hyderabad");
                        journey.setDestination("Banglore");
                        journey.setJourneyDate(new Date());
                        journey.setFare(750.3f);
                        journey.setDistance(567);
                        journey.setPassengers(passengers);
        session.save(journey);
//Accessing the data
                        //journey = (Journey) session.get(Journey.class, 1);
        //System.out.println(journey.getPassengers().size());
flag = true;
} finally {
                        if (transaction != null) {
                            if (flag) {
                            transaction.commit();
                        } else {
                            transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test☹(permutations &combinations)

case#1-storing the data.

case#2-accessing the data.

<u>Working With ManyToManySet using Annotations</u>

Journey.java

```
@Entity
@Table(name = "JOURNEY")
public class Journey {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "JOURNEY_ID")
        protected int journeyId;
        protected String source;
        protected String destination;
        @Column(name = "JOURNEY_DT")
        protected Date journeyDate;
        protected float fare;
        protected int distance;
@ManyToMany(cascade = CascadeType.ALL)
@JoinTable(name = "JOURNEY_PASSENGERS", joinColumns = {
@JoinColumn(name = "JOURNEY_ID") }, inverseJoinColumns = {
@JoinColumn(name = "PASSENGER_ID") })
        protected Set<Passenger> passengers;
//setters and getters
//@toString
}
```

Passenger.java

```
@Entity
@Table(name = "PASSENGER")
public class Passenger {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "PASSENGER_ID")
        protected int passengerId;
        protected String name;
        protected int age;
        protected String gender;
        protected String mobile;
        @Column(name = "EMERGENCY_CONTACT_NO")
        protected String emergencyContactNo;
        protected Set<Journey> journeys;
//setters and getters
//@toString
}
```

Note:Remaining Util,Test classes are same as in XML mapping only.

ManyToManyList

UseCase

✔Lets say we have two Entity classes **Invoice** and **Product** As follows.

```
public class Product {
        protected int productId;
        protected String name;
        protected String description;
        protected String manufacturer;
        protected float price;
//setters and getters
//@toString
}
public class Invoice {
        protected int invoiceId;
        protected Date invoiceDate;
        protected String customerName;
        protected String mobile;
        protected String email;
        protected double amount;
        protected List<Product> products;
//setters and getters
//@toString
}
```

✔The **Association** between these two classes is **many-to-many**,because A product can belongs to multiple invoices and An invoice can have multiple products.

✔The Tables will looks like as below..

```
INVOICE
|-INVOICE_ID(pk)
|-INVOICE_DT
|-CUSTOMER_NM
|-MOBILE
|-EMAIL
|-AMOUNT
PRODUCT
|-PRODUCT_ID(Pk)
|-NAME
|-DESCR
|-MANUFACTURER
```

|-PRICE
INVOICE_PRODUCTS
|-INVOICE_ID(cpk,Fk)
|-PRODUCT_ID(cpk,fk)
|-SERIAL_NO(index key column)
✓As it is a list,we need to maintain the products index order in third table.So the index column should be there in the third table as <list-index column="SERIAL_NO" base="1" />,and we will map index column at one side only.
Working Example:
```
ManyToManyList
|-src
  |-com.mtm.entites
        |-Product.java
        |-Product.hbm.xml
        |-Invoice.java
        |-Invoice.hbm.xml
  |-com.mtm.util
       |-HibernateUtil.java
  |-com.mtm.test
       |-MTMLTest.java
|-hibernate.cfg.xml
```
product.java
```
public class Product {
        protected int productId;
        protected String name;
        protected String description;
        protected String manufacturer;
        protected float price;
//setters and getters
//@toString
}
```
Invoice.java
```
public class Invoice {
        protected int invoiceId;
        protected Date invoiceDate;
        protected String customerName;
        protected String mobile;
        protected String email;
        protected double amount;
        protected List<Product> products;
//setters and getters
//@toString
}
```

Product.hbm.xml

```
<hibernate-mapping package="com.mtml.entities">
        <class name="Product" table="PRODUCT">
                <id name="productId" column="PRODUCT_ID">
                        <generator class="increment" />
                </id>
                <property name="name" column="NAME" />
                <property name="description" column="DESCR" />
                <property name="manufacturer" column="MANUFACTURER" />
                <property name="price" column="price" />
        </class>
</hibernate-mapping>
```

Invoice.hbm.xml

```
<hibernate-mapping package="com.mtml.entities">
        <class name="Invoice" table="INVOICE">
                <id name="invoiceId" column="INVOICE_ID">
                        <generator class="increment" />
                </id>
                <property name="invoiceDate" column="INVOICE_DT" />
                <property name="customerName" column="CUSTOMER_NM" />
                <property name="mobile" column="MOBILE" />
                <property name="email" column="EMAIL" />
                <property name="amount" column="AMOUNT" />
                <list name="products" table="INVOICE_PRODUCTS"
cascade="save-update">
                        <key column="INVOICE_ID" not-null="true" />
                        <list-index column="SERIAL_NO" base="1" />
                        <many-to-many column="PRODUCT_ID" class="Product" />
                </list>
        </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/mtm/entities/Product.hbm.xml" />
<mapping resource="com/mtm/entities/Invoice.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
            }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
           }
}
```

MTMLTest.java

```
public class MTMSTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                Invoice invoice = null;
                Product product = null;
                List<Product> products = null;

        try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
                /*products = new ArrayList<Product>();
        product = new Product();
                        product.setName("IPhone 6s");
                        product.setDescription("Apple smart phone");
                        product.setManufacturer("Apple");
                        product.setPrice(65000);
                products.add(product);
        product = new Product();
                        product.setName("Samsung Galaxy S7");
                        product.setDescription("Samsung smart phone");
                        product.setManufacturer("Samsung");
                        product.setPrice(50000);
               products.add(product);
        invoice = new Invoice();
                        invoice.setInvoiceDate(new Date());
                        invoice.setCustomerName("John");
                        invoice.setMobile("3378495");
                        invoice.setEmail("john@gmail.com");
                        invoice.setAmount(109304);
                        invoice.setProducts(products);
        session.save(invoice);*/
//accessing the data
        invoice = (Invoice) session.get(Invoice.class, 1);
                System.out.println(invoice.getProducts().size());
            flag = true;
} finally {
                if (transaction != null) {
                    if (flag) {
                    transaction.commit();
                } else {
                    transaction.rollback();
                }
            }
                HibernateUtil.closeSessionFactory();
            }
        }
}
```

Test☹(permutations &combinations)
case#1-storing the data.
case#2-accessing the data.

21-08-2016 & 22-08-2016

Many-to-Many Map

UseCase

✔Lets say we have two entity classes **Project** and **Resource** as follows..

```
public class Resource {
        protected int resourceId;
        protected String resourceName;
        protected String role;
//setters and getters
//@tostring
}
public class Project {
        protected int projectId;
        protected String title;
        protected String description;
        protected Date startDate;
        protected Date endDate;
        protected String client;
        protected String keyTechnologies;
        protected Map<String, Resource> assignedResources;
//setters and getters
//@tostring
}
```

✔The Association between these two classes is many-to-many,because One project can contain many resources,a resource can be there in multiple projects.

✔To represent the many-to-many relationship between entity classes we need to take the collection type on both sides.

✔ So lets consider map here as Map<String, Resource>

✔The Tables will looks like as below..

```
PROJECT
|-PROJECT_ID(pk)
|-TITLE
|-DESCR
|-START_DT
|-END_DT
|-CLIENT_NM
|-KEY_TECHNOLOGIES
RESOURCES
|-RESOURCE_ID(pk)
|-RESOURCE_NM
|-ROLE
PROJECT_RESOURCES
|-PROJECT_ID(key-column)
|-RESOURCE_ID(many-to-many key column)
|-CONTRACT_ID(map-key-column)
```

Note:CONTRACT_ID will be linked to the PROJECT_ID.

✔The Mapping file will looks as below..

```
<hibernate-mapping package="com.mtmm.entities">
        <class name="Project" table="PROJECT">
                <id name="projectId" column="PROJECT_ID">
                        <generator class="increment" />
                </id>
                <property name="title" column="TITLE" />
                <property name="description" column="DESCR" />
                <property name="startDate" column="START_DT" />
```

```
              <property name="endDate" column="END_DT" />
              <property name="client" column="CLIENT_NM" />
         <property name="keyTechnologies" column="KEY_TECHNOLOGIES" />
         <map name="assignedResources" table="PROJECT_RESOURCES"
cascade="save-update">
              <key column="PROJECT_ID" />
                   <map-key column="CONTRACT_ID" type="string"
length="50" />
              <many-to-many column="RESOURCE_ID" class="Resource" />
         </map>
      </class>
</hibernate-mapping>
```

<u>WorkingExample:</u>

```
ManyToManyMap
|-src
  |-com.mtm.entites
        |-Project.java
        |-Project.hbm.xml
        |-Resource.java
        |-Resource.hbm.xml
  |-com.mtm.util
       |-HibernateUtil.java
  |-com.mtm.test
       |-MTMMTest.java
|-hibernate.cfg.xml
```

Resource.java

```
public class Resource {
         protected int resourceId;
         protected String resourceName;
         protected String role;
//setters and getters
//@tostring
}
```

Project.java

```
public class Project {
         protected int projectId;
         protected String title;
         protected String description;
         protected Date startDate;
```

```
        protected Date endDate;
        protected String client;
        protected String keyTechnologies;
        protected Map<String, Resource> assignedResources;
//setters and getters
//@tostring
}
```

Resource.hbm.xml

```
<hibernate-mapping package="com.mtmm.entities">
        <class name="Resource" table="RESOURCES">
                <id name="resourceId" column="RESOURCE_ID">
        <generator class="increment" />
                </id>
                <property name="resourceName" column="RESOURCE_NM" />
                <property name="role" column="ROLE" />
        </class>
</hibernate-mapping>
```

Project.hbm.xml

```
<hibernate-mapping package="com.mtmm.entities">
        <class name="Project" table="PROJECT">
                <id name="projectId" column="PROJECT_ID">
                        <generator class="increment" />
                </id>
                <property name="title" column="TITLE" />
                <property name="description" column="DESCR" />
                <property name="startDate" column="START_DT" />

                <property name="endDate" column="END_DT" />
                <property name="client" column="CLIENT_NM" />
        <property name="keyTechnologies" column="KEY_TECHNOLOGIES" />
        <map name="assignedResources" table="PROJECT_RESOURCES"
cascade="save-update">
                <key column="PROJECT_ID" />
                        <map-key column="CONTRACT_ID" type="string"
length="50" />
                <many-to-many column="RESOURCE_ID" class="Resource" />
        </map>
      </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
        <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/mtm/entities/Project.hbm.xml" />
<mapping resource="com/mtm/entities/Resource.hbm.xml"/>
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
            }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
           }
}
```

MTMMTest.java

```
public class MTmSTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                Project project = null;
                Resource resource = null;
                Map<String, Resource> assignedResources = null;

         try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
/*assignedResources = new HashMap<String, Resource>();
                        resource = new Resource();
                        resource.setResourceName("Raja K");
                        resource.setRole("Lead");
                        assignedResources.put("C-1938", resource);
        resource = new Resource();
                        resource.setResourceName("Sumith");
                        resource.setRole("Developer");
                        assignedResources.put("C-1998", resource);
        project = new Project();|
                        project.setTitle("BMS System");
                        project.setDescription("Banking Management System");
                        project.setStartDate(new Date());
                        project.setKeyTechnologies("Java");
                        project.setClient("BOA");
                        project.setAssignedResources(assignedResources);
        session.save(project);*/
//Accessing the data
                        project = (Project) session.get(Project.class, 1);
                        System.out.println(project);
             flag = true;
        } finally {
                        if (transaction != null) {
                                if (flag) {
```

```
                        transaction.commit();
                    } else {
                        transaction.rollback();
                    }
                }
                    HibernateUtil.closeSessionFactory();
                }
        }
}
```
Test☹(permutations &combinations)
case#1-storing the data.
case#2-accessing the data.

Working with bag collection Type:

✔Bag is a special collection provided by Hibernate from 3.0 onwards.
✔The cases where we dont have the specific requirements of Set,List,Map we can go for Bag(not unique,not order)
✔When we are using bag,the reference should be list only.

Configuring bag collection type:
In general if will use set,then it will be as follows...
```
        <set name="approvals" inverse="true">
                <key column="PROPERTY_ID" not-null="true" />
                <one-to-many class="Approval" />
        </set>
```
Now its just enough to replace the existing collection with just bag only.
```
        <!-- <bag name="approvals">
                <key column="PROPERTY_ID" />
                <one-to-many class="Approval" />
        </bag> -->
```
23-08-2016

Bidirectional association mapping

→It is an association mapping in which both sides represents the single FK relationship in the database.

Exampe:
1)Actually in normal one to many, the relation is from parent to child i mean if we do the operations on parent object will be automatically reflected at child objects too.
2)Similarly in many to one the relation is from child to parent object. So here the Bidirectional one to many is the Combination of the above two sides representtion.

inverse="true"/ "false"

(1) In Hibernate, only the "relationship owner" should maintain the relationship.
(2) The "inverse" keyword is used to decide which side is the relationship owner to manage the relationship (insert or update of the foreign key column).
(3) In short, inverse="true" means this is the relationship owner, and inverse="false" (default) means it's not the relationship owner.
(4). This is for bi-directional associations. In the database world, there is no such thing as a bi-directional. You can have a FK field in one table referring to another table. So if you have a bi-directional assocation in Java like

```
Order {
List<OrderItems> items
}
OrderItem {
Order order
}
```

Hibernate won't know that your mapping both sides represents the single FK relationship in the database. So by putting in **inverse=true**, it tells Hibernate that both mappings map to the **same single FK relationship**. Otherwise, Hibernate will try to update both sides seperately, and not in any order. So it might try to insert a child before the parent is inserted, and the database will throw an error.
(5) The **mappedBy** attribute of **@OneToMany** annotation behaves the same as inverse = true in the xml file.
(6).It helps to avoid many unnecessary update statements as follows..
Lets say we have two tables
→ PROPERTY – (Parent)
→ APPROVAL– (Child, foreign-key –PROPERTY_ID)
Relationship between PROPERTY to APPROVAL is the one to many.
Relationship between APPROVAL to PROPERTY is the many to one.
→Relationship of PROPERTY to APPROVAL Not Equals(!=) Relationship of APPROVAL to PROPERTY
→Foreign-key PROPERTY_ID maintain relationship between PROPERTY to APPROVAL
→Same Foreign-key PROPERTY_ID maintain relationship between APPROVAL to PROPERTY.

Relationship of PROPERTY to APPROVAL:
Here Relationship owner is PROPERTY, this relationshipOwner will take care of parent and child relationship with updation of foreign-key, because foreign-key maintain relationship both of them.
Relationship APPROVAL to PROPERTY
Here Relationship owner is APPROVAL, this relationshipowner will take care of parent and child relationship with updation of foreign-key, because foreign-key maintain relationship both of them.
→In Bi-direction means we have 2 uni-directions like
1) PROPERTY to APPROVAL
2) APPROVAL to PROPERTY
→In this scenario foreign-key will by updated by twice, which is not necessary because one uni-direction is enough maintain relationship between two tables.
→To avoid the foreign-key updation twice, hibernate introduces "inverse=true", In the Propeety.hbm.xml as follows..

```
 <set name="approvals" inverse="true">
                <key column="PROPERTY_ID" not-null="true" />
<one-to-many class="Approval" />
</set>
```

it will maintain the relationship means nothing like uni-directional, so that we are avoided updation foreign-key twice.

24-08-2016

Component Mapping

✔In component mapping, we will map the **dependent object** as a component.
✔An component is an object that is stored as an **value** rather than entity reference.
✔This is mainly used if the dependent object doen't have primary key.
✔ It is used in case of composition (**HAS-A** relation), that is why it is termed as component. Let's see the class that have HAS-A relationship.

```
Public Class Address{
private String addressLine1;
--------------------------------
//setters and getters
}
public class Employee{
private int id;
private String name;
private Address address;
//setters and getters}
```

Here address is a dependent object. Hibernate framework provides the facility to map the dependent object as a component.

Working Example☹Component Mapping with XML

```
ComponentMapping
|-src
    |-com.cm.entities
      |-MobilePlan.java
      |-TarrifDetails.java
      |-MobilePlan.hbm.xml
    |-com.cm.util
      |-HibernateUtil.java
    |-com.cm.test
      |-CMTest.java
|-hibernate.cfg.xml
```

MobilePlan.java

```
public class MobilePlan {
                  private int planId;
                  private String planName;
                  private String description;
                  private int billingDayOfMonth;
                private float rentalAmount;
                private TarrifDetails tarrifDetails;
//setters and getters
}
```

TarrifDetails.java

```
public class TarrifDetails {
                private float localCallCost;
                private float stdCallCost;
                private float isdCallCost;
               private float localMessageCost;
//setters and getters
}
```

The Association between these two classes is composition(HAS-A),so we need to use component mapping.

MobilePlan.hbm.xml

```
<hibernate-mapping package="com.cm.entities">
        <class name="MobilePlan" table="MOBILE_PLAN">
                <id name="planId" column="PLAN_ID">
```

```
                    <generator class="increment" />
            </id>
            <property name="planName" column="PLAN_NM" />
            <property name="description" column="DESCR" />
            <property name="billingDayOfMonth"
column="BILLING_DAY_OF_MONTH" />
            <property name="rentalAmount" column="RENTAL_AMOUNT" />
            <component name="tarrifDetails" class="TarrifDetails">
                    <property name="localCallCost"
column="LOCAL_CALL_COST" />
                    <property name="stdCallCost" column="STD_CALL_COST" />
            <property name="isdCallCost" column="ISD_CALL_COST" />
            <property name="localMessageCost"
column="LOCAL_MESSAGE_COST" />
            </component>
        </class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<hibernate-configuration>
  <session-factory>
       <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
 <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="connection.username">hib_tools</property>
<property name="connection.password">welcome1</property>
<property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
 <property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<property
name="hibernate.current_session_context_class">thread</property>
<mapping resource="com/cm/entities/MobilePlan.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
            }
public static SessionFactory getSessionFactory() {
                return sessionFactory;
        }
public static void closeSessionFactory() {
        if (sessionFactory != null) {
                sessionFactory.close();
           }
}
```

CMTest.java

```
public class CMTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                MobilePlan mobilePlan = null;
                TarrifDetails tarrifDetails = null;
        try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
        mobilePlan = new MobilePlan();
                        mobilePlan.setPlanName("Vodafone 199");
                        mobilePlan.setDescription("Post paid 199 monthly plan");
                        mobilePlan.setBillingDayOfMonth(10);
                        mobilePlan.setRentalAmount(199);
        tarrifDetails = new TarrifDetails();
                        tarrifDetails.setLocalCallCost(0.50f);
                        tarrifDetails.setStdCallCost(1);
                        tarrifDetails.setIsdCallCost(3.50f);
                        tarrifDetails.setLocalMessageCost(0.25f);
        mobilePlan.setTarrifDetails(tarrifDetails);
        session.save(mobilePlan);
                mobilePlan = (MobilePlan) session.get(MobilePlan.class, 1);
        System.out.println(mobilePlan.getTarrifDetails().getLocalCallCost());
        flag = true;
} finally {
                        if (transaction != null) {
                                if (flag) {
                                transaction.commit();
                        } else {
                                transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Test☹(permutations &combinations)

case#1-inserting the data{session.save(mobileplan)}

Data will be inserted for both the MobilePlan and TarrifDetails.

case#2-accessing the data.

<u>ComponentMapping with Annotations</u>:

TarrifDetails.java

```
//Its not just Entity class rather it is embedadable in another class so we will
annotate wthis class with @Embeddable Annotation.
@Embeddable
public class TarrifDetails {
        @Column(name = "LOCAL_CALL_COST")
        private float localCallCost;
        @Column(name = "STD_CALL_COST")
        private float stdCallCost;
        @Column(name = "ISD_CALL_COST")
        private float isdCallCost;
        @Column(name = "LOCAL_MESSAGE_COST")
        private float localMessageCost;
//setters and getters
}
```

MobilePlan.java

```
@Entity

@Table(name = "MOBILE_PLAN")

public class MobilePlan {

	@Id

	@GeneratedValue(strategy = GenerationType.IDENTITY)

	@Column(name = "PLAN_ID")

	private int planId;

	@Column(name = "PLAN_NM")

	private String planName;

	@Column(name = "DESCR")

	private String description;

	@Column(name = "BILLING_DAY_OF_MONTH")

	private int billingDayOfMonth;

	@Column(name = "RENTAL_AMOUNT")

	private float rentalAmount;

	@Embedded

	private TarrifDetails tarrifDetails;

//setters and getters
}
```

Note:Remaining hibernate.cfg.xml,HibenrateUtil,Test classes are same as XML Mapping only.

Part-C

3.Hibernate Query Language(HQL)

Introduction:

✓Most of the times As per our business requirements,we wanted to access collection of records (or) multiple records from the database.

✓In such cases instead of writing SQL queries we can go for the Hibernate provided Queries.

So Hibernate created a new language named Hibernate Query Language (HQL), the syntax is quite similar to database SQL language.

✓The main difference between is **HQL uses class name instead of table name, and property names instead of column name**.

Advantages of HQL over SQL:

1. HQL is database agnostic.That is, it works with any database that Hibernate works with.
2. The syntax is quite similar to database SQL language only.
3. A change in the underlying Table will not affects HQL Queries because these are refers to objects not the Tables.
4. HQL Queries are portable across the databases.
5. HQL is extremely simple to learn and use, and the code is always self-explanatory.

Working with HQL Queries::

→If we wanted to execute an HQL Query we need Session.
→Once we have Session then we need query(org.hibernate.Query
→OneToManySet(Tables)
    |-HQLPATest.java(reference)

1-How to getAllProperties

```
Query getAllPropertiesQuery =session.createQuery("from Property");
List<Property> allProperties = getAllPropertiesQuery.list();
for (Property property : allProperties){
System.out.println("Property : " + property.getPropertyName() + " approval : "
+ property.getApprovals());
 }
```

Now this will get all the Properties along with the corresponding approvals.

2-Where clause Get Properties based on type

```
Query getPropertiesByTypeQuery = session .createQuery("from Property p
where p.type = ?");
getPropertiesByTypeQuery.setString(0, "Independent");
        List<Property> propertiesByType =getPropertiesByTypeQuery.list();
```

```
printProperties(propertiesByType);
```

This is the way we will write where condition and how to execute a query by substituing substitutional parameters.

3-Inner Join Get Properties who has approvals

```
Query getPropertiesHasApprovalsQuery = session .createQuery("Select distinct p from Property p inner join p.approvals");
            List<Property> propertiesWithApprovals = getPropertiesHasApprovalsQuery.list();
printProperties(propertiesWithApprovals);
```

This query will get all the properties who has approvals.

4-Where clause on Join tables Get Properties which are of type 'Gated' and has approval of GHMC

```
Query getPropertiesBasedOnTypeAndApprovalQuery =session.createQuery( "Select p from Property p inner join p.approvals a where p.type = ? and a.issuedAuthority = ?" );
getPropertiesBasedOnTypeAndApprovalQuery.setString(0, "Gated");
getPropertiesBasedOnTypeAndApprovalQuery.setString(1, "GHMC");
getPropertiesBasedOnTypeAndApprovalQuery.list();
 printProperties(properties);
```

This query will gets only the properties which are of of Gated and approved from GHMC.

5-Left outer join

```
Query getPropertyAndApproval = session.createQuery("Select p.propertyName, a.approvalName from Property p left outer join p.approvals a where p.amount > ?");
getPropertyAndApproval.setFloat(0, 2000000f);
          List<Object[]> rows = getPropertyAndApproval.list();
                for (Object[] row : rows) {
  System.out.println("PName : " + row[0] + " - approval : "+ row[1]);
}
```

This query will returns all the property with approvals whose amount is greater than 2000000 rupees.

26-08-2016 & 27-08-2016

ManyToManySet(Tables)
|-HQLJPTest.java(reference)

6-SubQuery

```
Query query = session .createQuery("Select p from Passenger p where p.age =
(Select max(age) from Passenger)" );
            List<Passenger> passengers = query.list();
System.out.println(passengers.get(0).getName());
```

This query returns the maximam age holded passenger.

7-Aggregate-Query

```
Query query = session.createQuery("Select count(p) from Passenger p inner
join p.journeys j where j.source = ? and j.destination = ?" );
            query.setString(0, "Hyderabad");
            query.setString(1, "Banglore");
 List<Long> passengers = query.list();
           Long count = passengers.get(0);
 System.out.println(count);
```

This query will returns The number of passengers who ever travelling from Hyderabad to Banglore.

8- size operator

```
Query query = session.createQuery( "select p from Passenger p where
p.journeys.size > 1");
 List<Passenger> rows = query.list(); for(Passenger row : rows) {
 System.out.println(row);
}
```

This query returns list of passengers who have travelled more than one journey.

9-journeyId-passengers

```
Query query = session .createQuery("Select j.journeyId, j.passengers.size from
Journey j" );
List<Object[]> rows = query.list();
for (Object[] row : rows)
{
System.out.println("Journey : " + row[0] + " - noOfPassengers : " + row[1]);
}
```

This query will returns number of passengers travelling for a specific journey id.

10-group by

```
Query query = session.createQuery("Select j.source, count(p.passengerId) from
Journey j inner join j.passengers p group by (j.source)");
        List<Object[]> rows = query.list();
             for(Object[] row : rows) {
            System.out.println("Source : "+ row[0] + " - c : "+ row[1]);
}
```

This query will returns For every Source how many number of passengers have been travelled within any number of trips.

11-list

```
Query query = session.createQuery("Select list(j.source, j.destination) from
Journey j");
    List<List<String>> rows = query.list();
              for(Object[] row : rows) {
 System.out.println(row[0] + "-"+ row[1] );
}
```

This query will returns source and destination of a journey.

12-Custom Object

After querying the data ,If we wanted to transform the data into another object rather than into Entity object.

Lets create one Class as TripDetailsDto as follows..

```
public class TripDetailsDto {
         private String source;
         private String destination;
         private Date tripDate;
         private long noOfPassengers;
//constructor
//setters and getters
}
//writing Query
Query query = session.createQuery("select new
com.mtm.dto.TripDetailsDto(j.source, j.destination, j.journeyDate,
j.passengers.size) from Journey j");
                              List<TripDetailsDto> trips = query.list();
                              for(TripDetailsDto trip : trips) {
         System.out.println(trip);
}
```

13-minimam two journeys travelled

```
Query query = session .createQuery( "Select p from Passenger p where 2 <=
(Select count(jp.journeyId) from Passenger p1 inner join p1.journeys jp where
p1.passengerId = p.passengerId)" );
 List<Passenger> passengers = query.list();
 for(Passenger p :passengers) {
       System.out.println(p);
 }
```

14-fetch join

```
Query query = session.createQuery( "Select p from Passenger p inner join fetch
p.journeys");
  List<Passenger> passengers = query.list(); for(Passenger p : passengers) {
System.out.println(p);
                System.out.println(p.getJourneys());
 }
```

This query returns the journey details for Every passenger.

TablePerConcreteClass(Tables)
 |-TPSCHQLTest.java()

15-polymorphic queries

```
Query query = session.createQuery("Select i from InsurancePlan i where
i.uPlanNo = ?");
                        query.setInteger(0, 23);
                        List<InsurancePlan> plans = query.list();
                        System.out.println(plans.get(0).getPlanName());
```

This query should suport us to query data from any Table and it should give appropriate class object.

Criteria API:

Introduction:

In General There are three way to pulling data from the database in the Hibernate.

1.Using session methods(get() and load() methods) -limited control to accessing data

2.Using HQL - Slightly more control using where clause and other clauses but there are some problems here... is more complicated to maintain in case of bigger queries with lots of clauses.

3.Using Criteria API

WHAT IS CRITERIA API?

✓The API (Application Programming Interface) of Hibernate Criteria provides an elegant way of building dynamic query on the persistence database.

✓The hibernate criteria API is very Simplified API for fetching data from Criterion objects. The criteria API is an alternative of HQL (Hibernate Query Language) queries. It is more powerful and flexible for writing tricky criteria functions and dynamic queries.

✓ Criterions means rules that are the placed to query the data from the Entity.

✓The Criteria interface provides many methods to specify criteria. The object of Criteria can be obtained by calling the **createCriteria()** method of Session interface.

Criteria criteria = session.createCriteria(Student.class)

Advantages☺

1.The Criteria API supports all the comparision operators such as =, <, >, >=,=<, etc in the supported Expression class eq(), lt(), gt(), le() , ge() respectively.

2.Criteria provide some functions to make pagination extremely easy

3.The HCQL provides methods to add criteria, so it is **easy** for the java programmer to add criteria. The java programmer is able to add many criteria on a query.

Disadvantages:

The Criteria API do bring some disadvantages.

1. Performance issue

You have no way to control the SQL query generated by Hibernate, if the generated query is slow, you are very hard to tune the query, and your database administrator may not like it.

2. Maintainece issue

All the SQL queries are scattered through the Java code, when a query went wrong, you may spend time to find the problem query in your application. On the others hand, named queries stored in the Hibernate mapping files are much more easier to maintain.

Working with Criteria API

OneToManySet(Tables)

|-CriteriaAPITest.java(reference)

1-Get All properties

Criteria getAllPropertiesCriteria = session .createCriteria(Property.class);
List<Property> allProperties = getAllPropertiesCriteria.list();
printProperties(allProperties);

2-Where clause Get Properties based on type

Criteria getPropertiesByType =
session.createCriteria(Property.class).add(Restrictions.like("type", "Gated"));
List<Property> allProperties = getPropertiesByType.list();
printProperties(allProperties);

3-Inner Join Get Properties who has approvals

Criteria getPropertiesHasApprovals = session.createCriteria(
Property.class).createCriteria("approvals");
List<Property> allProperties = getPropertiesHasApprovals.list();
printProperties(allProperties);

4-Where clause on Join tables Get Properties which has approval of GHMC

Criteria getPropertiesByApprovalType = session .createCriteria(Property.class)
.createAlias("approvals", "a") .add(Restrictions.like("a.issuedAuthority",
"GHMC"));
List<Property> allProperties = getPropertiesByApprovalType.list();
printProperties(allProperties);

5-Instead of Restrictionsn class using Property

org.hibernate.criterion.Property amountProperty
=org.hibernate.criterion.Property .forName("amount");
CriteriagetPropertyByAmount = session.createCriteria(Property.class).add(
amountProperty.between(2000000f, 35000000f));
List<Property> properties =getPropertyByAmount.list();
printProperties(properties);

→The Criteria API provides the **org.hibernate.criterion.Projections** class which can be used to get average, maximum or minimum of the property values. The Projections class is similar to the Restrictions class in that it provides several static factory methods for obtaining **Projection** instances.

6.Selecting a specific column

Projection

//creating a projection for PropertyName
Projection projection = Projections.property("propertyName");

Criteria getAllPropertyNames =
session.createCriteria(Property.class).setProjection(projection);
 List<String> propertyNames = getAllPropertyNames.list();
 for (String name :propertyNames) {
 System.out.println(name); }

7.Selecting multiple columns

//creating a projection and adding multiple properties.
Projection projection1 =
Projections.projectionList().add(Projections.property("propertyName")).add(Proj
ections.property("amount"));
Criteria getNameAndAmountProperties = session.createCriteria(*
Property.class).setProjection(projection1);
List<Object[]> rows = getNameAndAmountProperties.list(); for (Object[] row :
rows) {
 System.out .println("name : " + row[0] + " - amount : " + row[1]);
}

8.Working with projection for each column

 //creating projections
 Projection groupByType = Projections.groupProperty("type");
 Projection countProperty = Projections.count("propertyName");
 Projection countAndGroupProjection = Projections.projectionList()
.add(countProperty).add(groupByType);
 Criteria getNoOfPropertiesByType = session.createCriteria(
Property.class).setProjection(countAndGroupProjection);

 List<Object[]> rows = getNoOfPropertiesByType.list();
for (Object[] row : rows) {
 System.out .println("name : " + row[0] + " - no : " + row[1]);
 }

28-08-2016 & 29-08-2016

Detached Criteria in Hibernate

Differences between Criteria and Detached Criteria:

✔Using a DetachedCriteria is exactly the same as a Criteria except you can do the initial creation and setup of your query without having access to the session.

✔The detached criteria allows us to create the query without Session. Then you can execute the search in an arbitrary session.

✔ When it comes time to run our query, we must convert it to an executable query with getExecutableCriteria(session).

How to create a Detached Criteria:

```
public static DetachedCriteria buildPropertyByIdCriteria(int propertyId) {
        DetachedCriteria dc = DetachedCriteria.forClass(Property.class);
                dc.add(Restrictions.eq("propertyId", propertyId));
                return dc;
        }
```

How to convert detached query into executbale query:

```
DetachedCriteria dc = buildPropertyByIdCriteria(1);
                Criteria c = dc.getExecutableCriteria(session);
        List<Property> properties = c.list();
                        for(Property p : properties) {
                                System.out.println(p);
                        }
```

Named Queries

✔While executing either HQL, NativeSQL Queries if we want to execute the same queries for multiple times and in more than one client program application then we can use the Named Queries mechanism.

✔Transforming a HQL Query into NativeSQL Query is costly job,to avoid that problem we can make use of Named Query

✔In this Named Queries concept, we use some name for the query configuration, and that name will be used when ever the same query is required to execute.

✔In hibernate mapping file we need to configure a query by putting some name for it and in the client application, we need to use getNamedQuery() given by session interface, for getting the Query reference and we need to execute that query by calling list().

✓Mapping will be loaded at the time creating SessionFactory,so the named queies will be compiled and translated into native SQL Queries.
✓If you want to create Named Query then we need to use query element in the hibernate mapping file.

How to create Named Query in maping file:

```
<query name="journeyById">
        <![CDATA[from Journey j where j.journeyId = ?]]>
</query>
```

How to execute Named SQL Query

```
Query query = session.getNamedQuery("journeyById");
                query.setInteger(0, 1);
                List<Journey> journeys = query.list();
    for(Journey j : journeys) {
                System.out.println(j);
                }
```

Hibernate Named Query using Annotations

✓If you want to use named query in hibernate, you need to have knowledge of @NamedQueries and @NamedQuery annotations.
✓**@NameQueries** annotation is used to define the multiple named queries.
✓**@NameQuery** annotation is used to define the single named query.
Let's see the example of Annotations using the named queries:

```
@NamedQueries(
  {
     @NamedQuery(
           name = " journeyById ",
          query = " from Journey j where j.journeyId = ?"  )
 } )
```

Native SQL Queries

✓Hibernate Query Language (HQL) is used to interact with the database in database independent way. However there might be some scenarios where we would want to use database specific functions or data types etc. To support such scenarios, Hibernate provides the support for native SQL as well.
✓For HQL we use createQuery(SQL) method for HQL where as for native SQL we have to use createSQLQuery(SQL) method of session API to get the SQLQuery instance.

Note: Native SQL uses table names not class names in query.it is not recommended to use native SQL Queries.

How to create Native SQl Query in mapping file:

```
<sql-query name="getJourneyDetails">
        <![CDATA[select * from journey]]>
  </sql-query>
```

How to create Native SQL Query directly:

```
SQLQuery query = session.createSQLQuery("select * from
journey").addEntity(Journey.class);
                        List<Journey> journeys = query.list();
        for(Journey j : journeys) {
                        System.out.println(j);
                        }
```

30-08-2016

Second Level Cache in Hibernate

What is Second Level cache?

Caching is facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate also provide this caching functionality, in two layers.

- Fist level cache: This is enabled by default and works in **session scope**.
- Second level cache: This is apart from first level cache which is available to be used globally in **session factory scope**.
- Query Level Cache☹(It is not recommended interms of perfromance)

second level cache is created in **session factory scope** and is **available** to be **used** in **all sessions** which are created using that particular session factory.
It also means that once session factory **is closed, all cache** associated **with it die** and cache manager also closed down.

Why do we need to use SecondLevelCache?

First Level Cache by default applicable for all the Entity classes Objects Unlike First level cache when it comes Second level cache it is configurable and programmable ,The user has to select the specific set of Entity classes objects which has to be cached,whose data will **not** gets **changed frequently**(Ex:Master Table).

How to enable second Level cache?

By default Second level cahce is not enabled,To make it enable, Hibernate has provided an Integration with several powerful caching API's in the market.It eliminates us from creating cache mecahnisms.The Sun Micro Systems provided caching API is **Jcache**".Similarly lot of third part vendors provided their own imlementation.

Some of he third party vendor implementations are as follows...

- EH Cache
- OS Cache
- Swarm Cache
- JBoss Cache and etc.....

Each implementation provides different cache usage functionality. There are four ways to use second level cache.

1. **read-only:** caching will work for read only operation.
2. **nonstrict-read-write:** caching will work for read and write but one at a time.
3. **read-write:** caching will work for read and write, can be used simultaneously.
4. **transactional:** caching will work for transaction.

Implementation	read-only	nonstrict-read-write	read-write	transactional
EH Cache	Yes	Yes	Yes	No
OS Cache	Yes	Yes	Yes	No
Swarm Cache	Yes	Yes	No	No
JBoss Cache	No	No	No	Yes

Will second Level cache works under Clustered Environment or not?

Yes it will works.

Where we will apply Cache?

1.If the **s** is getting **changed frequently** within an interval of time,but if the **volume** of the **users** who are accesing the data is **high** then it is good to use cache.

2.Data may gets **changed** during certain **interval** of time only and the volume of **users** who are accesing the data could be **moderate to high**,then this scenario it is also qualified for caching.

3.There could be some **data** which **never** gets **modified** but the accessing volume of **users** is **high** then we can apply cache.

31-08-2016

Why SecondLevel Cache called as Distributed Cache?

When another session created from same session factory try to get entity, it is successfully looked up in second level cache and no database call is made.

How to work with EH Cache?

The caching frameworks came up with "regions",which has to be specified by the programmer that describes about where we wanted to apply cache.
To create the region Every Cache vendor provided one configuratio file in which we need to provide the regions along with setting about how to manage the data within the region.
Similarly EH Cache has also provided one default configuration file with the name **ehcache.xml**.
The following xml depicts how a ehcache.xml looks like...

```
<?xml version="1.0" encoding="utf-8"?>
<ehcache>
        <cache name="com.slc.entities.Job" maxElementsInMemory="100"
                eternal="true" />
</ehcache>
```

note:If we specify eternal=true then the data within cache never gets modified.
here cache=region.

WorkingExample:

```
EHCache
|-src
    |-com.ehc.test
            |-EhCacheTest.java
|-ehcache.cfg.xml
|-lib(eh cache jars)
```

ehcache.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ehcache>
        <cache name="com.slc.entities.Job" maxElementsInMemory="100"
                eternal="true" />
</ehcache>
```

EhCacheTest.java

```
public class EHCacheTest {
        public static void main(String[] args) {
                CacheManager cacheManager = new CacheManager();
                Cache myCache = cacheManager.getCache("mycache");
                myCache.put(new Element("key1", "value1"));
               Object value = myCache.get("key1");
        System.out.println(value);
        }
}
```

Usecase

Lets say We Posted a Job in Naukri,It will be accessed by millions of people ove rthe network who can have the access to Naukri.com Then This Job will not gets modified frequently.Once the recruitment is over tehn only it will be deleted from the nakuri.com.

Working Example:

```
SecondLevelCache
|-src
   |-com.slc.entities
       |-Job.java
       |-Job.hbm.xml
  |-com.slc.util
       |-HibernateUtil.java
  |-com.slc.test
       |-SLCTest.java
|-hibernate.cfg.xml
|-ehcache.xml
```

Job.java

```
public class Job {
               protected int jobId;
               protected String jobDescription;
               protected String level;
               protected String technology;
               protected Date postedDate;
               protected String location;
//setters and getters
//@toString
}
```

Job.hbm.xml

```
hibernate-mapping package="com.slc.entities">
        <class name="Job" table="JOBS">
               <cache usage="read-only" />
               <id name="jobId" column="JOB_ID">
```

```
            <generator class="increment" />
            </id>
            <property name="jobDescription" column="JOB_DESCR" />
            <property name="level" column="JOB_LEVEL" />
            <property name="technology" column="TECHNOLOGY" />
            <property name="postedDate" column="JOB_POSTED_DT" />
            <property name="location" column="JOB_LOCATION" />
        </class>
</hibernate-mapping>
```

Hibernate.cfg.xml

```
<hibernate-configuration>
        <session-factory>
            <property
name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
                <property
name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
                <property
name="connection.username">hib_tools</property> <property
name="connection.password">welcome1</property>
                <property
name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<property name="current_session_context_class">thread</property>
<property name="show_sql">true</property>
            <property name="hbm2ddl.auto">update</property>
            <mapping resource="com/slc/entities/Job.hbm.xml" />
        </session-factory>
</hibernate-configuration>
```

HibernateUtil.java

```
public class HibernateUtil {
        private static SessionFactory sessionFactory;
static {
        Configuration configuration = new Configuration().configure();
        StandardServiceRegistryBuilder builder = new
StandardServiceRegistryBuilder();
                builder.applySettings(configuration.getProperties());
                StandardServiceRegistry registry = builder.build();
                sessionFactory =
configuration.buildSessionFactory(registry);
            }
```

```
public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}
public static void closeSessionFactory() {
	if (sessionFactory != null) {
		sessionFactory.close();
	   }
}
```

SLCTest.java

```
public class SLCTest {
	public static void main(String[] args) {
		SessionFactory sessionFactory = null;
		Transaction transaction = null;
		Session session = null;
		boolean flag = false;
		Job job = null;
	try {
			sessionFactory = HibernateUtil.getSessionFactory();
			session = sessionFactory.getCurrentSession();
			transaction = session.beginTransaction();
	/*job = new Job();
			job.setJobDescription("Jee 3 Years Experience Developer");
			job.setLevel("Level-2");
			job.setPostedDate(new Date());
			job.setTechnology("Java");
			job.setLocation("Hyderabad");
	session.save(job);*/
flag = true;
} finally {
			if (transaction != null) {
				if (flag) {
				transaction.commit();
			} else {
				transaction.rollback();
			}
		}
			HibernateUtil.closeSessionFactory();
		}
	}}
```

Test:
Case#1-If we call Session.save(job) then the Data will be inserted into the Database JOBS Table.
Note:SecondLevelCache will not be implemented by the Developers.It will be enabled during the testing of our application.
To enabled Second level cache we need to write one configuration file.
ehcache.xml

```
<?xml version="1.0" encoding="utf-8"?>
<ehcache>
        <cache name="com.slc.entities.Job" maxElementsInMemory="100"
                eternal="true" />
</ehcache>
```

After this we need to add the following properties in the hibernate.cfg.xml

```
<property name="hibernate.cache.use_second_level_cache">true</property>
                <property
name="hibernate.cache.provider_configuration_file_resource_path">/ehcache.x
ml</property>
                <property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhC
acheRegionFactory</property>
//Statistics is used to identify the data coming from which level cache
<property name="hibernate.generate_statistics">true</property>
```

Test:
Lets write all the facts point by point:

1. Whenever hibernate session try to load an entity, the very first place it look for cached copy of entity in first level cache (associated with particular hibernate session).
2. If cached copy of entity is present in first level cache, it is returned as result of load method.
3. If there is no cached entity in first level cache, then second level cache is looked up for cached entity.
4. If second level cache has cached entity, it is returned as result of load method. But, before returning the entity, it is stored in first level cache also so that next invocation to load method for entity will return the entity from first level cache itself, and there will not be need to go to second level cache again.

5. If entity is not found in first level cache and second level cache also, then database query is executed and entity is stored in both cache levels, before returning as response of load() method.
6. Second level cache validate itself for modified entities, if modification has been done through hibernate session APIs.
7. If some user or process make changes directly in database, the there is no way that second level cache update itself until "timeToLiveSeconds" duration has passed for that cache region. In this case, it is good idea to invalidate whole cache and let hibernate build its cache once again. You can use below code snippet to invalidate whole hibernate second level cache.

Case#1

fetching the data per first time(

```
job = (Job) session.get(Job.class, 1);
            System.out.println("First Level Cache Hit : "
      + sessionFactory.getStatistics().getEntityFetchCount());
System.out.println("Second Level Cache Hit : "
+ sessionFactory.getStatistics().getSecondLevelCacheHitCount());
      System.out.println(job);
```

→ 0 0

Explanation: Entity is not present in either 1st or 2nd level cache so, it is fetched from database.

Case#2

Fetching again the same data

```
job = (Job) session.get(Job.class, 1);
            System.out.println("First Level Cache Hit : "
      + sessionFactory.getStatistics().getEntityFetchCount());
System.out.println("Second Level Cache Hit : "
+ sessionFactory.getStatistics().getSecondLevelCacheHitCount());
      System.out.println(job);
```

→1   0

Explanation: Entity is present in first level cache so, it is fetched from there. No need to go to second level cache.

Case#3-(session.evict(job);)

```
job = (Job) session.get(Job.class, 1);
            System.out.println("First Level Cache Hit : "
        + sessionFactory.getStatistics().getEntityFetchCount());
System.out.println("Second Level Cache Hit : "
+ sessionFactory.getStatistics().getSecondLevelCacheHitCount());
        System.out.println(job);
```

→ 1   1

Explanation: First time entity is fetched from database. Which cause it store in 1st and 2nd level cache. Second get call fetched from first level cache. Then we evicted entity from 1st level cache. So third get() call goes to second level cache and getSecondLevelCacheHitCount() returns 1.

Working Example

SecondLevelCache with Annotations:

```
@Entity
@Table(name = "JOBS")
@Cache(region = "job", usage = CacheConcurrencyStrategy.READ_ONLY)
public class Job {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "JOB_ID")
        protected int jobId;
        @Column(name = "JOB_DESCR")
        protected String jobDescription;
        @Column(name = "JOB_LEVEL")
        protected String level;
        protected String technology;
        @Column(name = "JOB_POSTED_DT")
        protected Date postedDate;
        protected String location;
//setters and getters
//@toString
}
```

Complete Test Class:

SLCTest.java

```
public class SLCTest {
        public static void main(String[] args) {
                SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;
                boolean flag = false;
                Job job = null;
        try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
        /*job = new Job();
                        job.setJobDescription("Jee 3 Years Experience Developer");
                        job.setLevel("Level-2");
                        job.setPostedDate(new Date());
                        job.setTechnology("Java");
                        job.setLocation("Hyderabad");
        session.save(job);*/
job = (Job) session.get(Job.class, 1);
                System.out.println("First Level Cache Hit : "
        + sessionFactory.getStatistics().getEntityFetchCount());
System.out.println("Second Level Cache Hit : "
+ sessionFactory.getStatistics().getSecondLevelCacheHitCount());
        System.out.println(job);

job = (Job) session.get(Job.class, 1);
                System.out.println("First Level Cache Hit : "
        + sessionFactory.getStatistics().getEntityFetchCount());
System.out.println("Second Level Cache Hit : "
+ sessionFactory.getStatistics().getSecondLevelCacheHitCount());
        System.out.println(job);

transaction.commit();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();
job = (Job) session.get(Job.class, 1);
                        System.out.println("First Level Cache Hit : "+
sessionFactory.getStatistics().getEntityFetchCount());
        System.out.println("Second Level Cache Hit : "+
sessionFactory.getStatistics() .getSecondLevelCacheHitCount());
flag = true;
} finally {
                        if (transaction != null) {
                                if (flag) {
                                transaction.commit();
                        } else {
```

```
                        transaction.rollback();
                    }
                }
                    HibernateUtil.closeSessionFactory();
                }
        }
}
```

Global Transactions Management in Hibernate

✔In Hibernate, the transaction management is quite standard, just remember any exceptions thrown by Hibernate are **FATAL**, you have to roll back the transaction and close the current session immediately.

✔In hibernate framework, we have **Transaction** interface(JTA) that defines the unit of work.

✔A transaction is associated with Session and instantiated by calling **session.beginTransaction()**.

✔Here's a Hibernate transaction template :

```
SessionFactory sessionFactory = null;
                Transaction transaction = null;
                Session session = null;

try {
                        sessionFactory = HibernateUtil.getSessionFactory();
                        session = sessionFactory.getCurrentSession();
                        transaction = session.beginTransaction();

-----------perssiting the data & fetching the data----------------
} finally {
                        if (transaction != null) {
                                if (flag) {
                                transaction.commit();
                        } else {
                                transaction.rollback();
                        }
                }
                        HibernateUtil.closeSessionFactory();
                }
        }
}
```

Conclusion:

So These are the Several concepts related to Hibernate framework and in order to work with Hibernate Effectively without any problems.