**Hibernate Short Note:-**

## ****Draw Backs of JDBC:****

* In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.
* Hear as a programmer we must close the connection, or we may get a chance to get our of connections message…!
* Actually if we didn’t close the connection in the finally block, then jdbc doesn’t responsible to close that connection.
* In JDBC we need to write Sql` commands in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious.
* JDBC used to generate database related error codes if an exception will occurs, but java programmers are unknown about this error codes right.
* In the Enterprise applications, the data flow with in an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text.  Because JDBC doesn’t transfer objects directly.

In order to overcome above problems, Hibernate came into picture..!

## What is Hibernate?

[](http://www.java4s.com/hibernate/why-and-what-is-hibernate-hibernate-introduction/attachment/gavin-king-picture/)

* Hibernate is the ORM tool given to transfer the data between a java (object) application and a database (Relational) in the form of the objects.
* Hibernate is the open source, light weight tool given by **Gavin King**, actually JBoss server is also created by this person only.
* Hibernate is a non-invasive framework,  means it won’t forces the programmers to extend/implement any class/interface, and
* Hibernate we have all POJO classes so its light weight.
* Hibernate can runs with in or without server, i mean it will suitable for all types of java applications (stand alone or desktop or any servlets bla bla.)
* Hibernate is purely for persistence (to store/retrieve data from Database).

**Mapping and Configuration are very familiar keywords we used to hear in the hibernate, every hibernate program must need these 2 xml files.**

## Mapping File:

* Mapping file is the heart of hibernate application.
* Every ORM tool needs this mapping; mapping is the mechanism of placing an object properties into columns of a table.
* Mapping file we can defined there are 2 ways
* in the form of an XML or
* in the form of the annotations.
* The mapping file contains mapping from a pojo class name to a table name and pojo class variable names to table column names.
* While writing an hibernate application, we can construct one or more mapping files, mean a hibernate application can contain any number of mapping files.

Generally an object contains 3 properties like

* Identity (Object Name)
* State (Object values)
* Behavior (Object Methods)

But while storing an object into the database, we need to store only the values(State) right ? but how to avoid identity, behavior.. its not possible. In order to inform what value of an object has to be stored in what column of the table, will be taking care by the mapping,  actually mapping can be done using 2 ways,

* XML
* Annotations.

Actually annotations are introduced into java from JDK 1.5.

## ****Syntax Of Mapping xml:****

|  |  |
| --- | --- |
|  | <hibernate-mapping>  <classname="POJO class name"table="table name in database">  <id name="variable name"column="column name in database"type="java/hibernate type"/>  <propertyname="variable1 name"column="column name in database"type="java/hibernate type"/>  <propertyname="variable2 name"column="column name in database"type="java/hibernate type"/>  </class>  </hibernate-mapping> |

## Configuration:

Configuration is the file loaded into an hibernate application when working with hibernate,

this configuration file contains 3 types of information..

Connection Properties

Hibernate Properties

Mapping file name(s)

We must create one configuration file for each database

we are going to use, suppose if we want to connect with 2 databases, like Oracle, MySql, then we must create 2 configuration files.

No. of databases we are using = that many number of configuration files

We can write this configuration in 2 ways…

* xml
* By writing Properties file.  We don’t have annotations hear, actually in hibernate 1, 2.x we defined this configuration file by writing .properties file, but from 3.x xml came into picture.

so, finally

Mapping –> xml, annotations  
Configuration –> xml, .properties (old style)

## Syntax Of Configuration xml:

|  |  |
| --- | --- |
|  | <hibernate-configuration>  <session-factory>  <!-- Related to the connection START -->  <property name="connection.driver\_class">Driver Class Name </property>  <property name="connection.url">URL </property>  <propertyname="connection.user">user </property>  <propertyname="connection.password">password</property>  <!-- Related to the connection END -->  <!-- Related to hibernate properties START -->  <property name="show\_sql">true/false</property>  <property name="dialet">Database dialet class</property>  <propertyname="hbm2ddl.auto">create/update or what ever</property>  <!-- Related to hibernate properties END-->  <!-- Related to mapping START-->  <mappingresource="hbm file 1 name .xml"/ >  <mappingresource="hbm file 2 name .xml"/ >  <!-- Related to the mapping END -->   </session-factory>  </hibernate-configuration> |

But XML files are always recommended to work.

# Main Advantage And Disadvantages Of Hibernates

Let us see what the advantages and disadvantages are of hibernate framework

## Advantages of hibernates:

* Hibernate supports Inheritance, Associations, Collections
* In hibernate if we save the derived class object,  then its base class object will also be stored into the database, it means hibernate supporting inheritance
* Hibernate supports relationships like One-To-Many,One-To-One, Many-To-Many-to-Many, Many-To-One
* This will also supports collections like List,Set,Map (Only new collections)
* In jdbc all exceptions are **checked exceptions**, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws.  Actually in hibernate we have the translator which converts checked to Un-checked ;)
* Hibernate has capability to generate primary keys automatically while we are storing the records into database
* Hibernate has its own query language, i.e hibernate query language which is database independent
* So if we change the database, then also our application will works as HQL is database independent
* HQL contains **database independent** commands
* While we are inserting any record, if we don’t have any particular table in the database, JDBC will rises an error like “View not exist”, and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us ;)
* Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
* Hibernate supports annotations, apart from XML
* Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.
* Getting pagination in hibernate is quite simple.

## Disadvantages of hibernates:

* I don’t think there are disadvantages in hibernate
* You know some thing.., Its saying hibernate is little slower than pure JDBC, actually the reason being hibernate used to generate many SQL statements in run time, but i guess this is not the disadvantage :-)
* Hibernate don’t have any controller. So it depending on some other framework technology.
* But there is one major disadvantage, which was boilerplate code issue, actually we need to write same code in several files in the same application, but spring eliminated this

# Simple Hibernate Application Requirements

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Any hibernate application, for example consider even first hello world program must always contains 4 files totally.

* POJO class
* Mapping XML
* Configuration XML
* One java file to write our logic

Actually these are the minimum requirement to run any hibernate application, and in fact we may require any number of POJO classes and any number of mapping xml files (**Number of POJO classes = that many number of mapping xmls**), and only one configuration xml and finally one java file to write our logic.

## POJO Class:

* POJO is a simple java file, no need to extend any class or implement any interface.
* This POJO class contain private properties variables, and for each property a setter and a getter

**Example:**

|  |  |
| --- | --- |
|  | Public class  Java4s  {  Private intstNo;  Private String stName;  Private String stAddress;        Public void setStno(int stNo)       {       this.stNo=stNo;       }       Public int getStNo()       {       Return stNo;       }        Public void setStName(int stName)       {       this.stName=stName;       }       Public int getStName()       {       Return stName;       }        Public void setStAddress(String stAddress)       {       this.stAddress=stAddress;       }       Public int getStAddress()       {       Return stAddress;       }   } |

## ****Mapping xml For POJO:****

Hear is the mapping file related to above pojo class, if you have any doubts on the syntax of the Mapping xml file, you can check our **previous session**

|  |  |
| --- | --- |
|  | <hibernate-mapping>     <class name="Java4s"table="STable">           <id name="stNo" column="SNo">                <generator class="assigned"/>           </id>           <property name="stName" column="SName"/>           <property name="stAddress "/></class>  </hibernate-mapping> |

Yes., see in this above mapping xml, for stAddress property i have not written any column name i just been specified  **<property name=”stAddress “/>,**this means in the database the column name for stAddress property will also be stAddress, in these cases we can ignore the column attribute to write, and i will explain about this <generator /> element later.

## Configuration XML

|  |  |
| --- | --- |
|  | <?xmlversion='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“>    <!-- Related to the connection START -->  <propertyname="connection.driver\_class">oracle.jdbc.driver.OracleDriver  ­</property>  <propertyname="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>  <propertyname="connection.user">user</property>  <propertyname="connection.password">password</property>  <!-- Related to the connection END -->    <!-- Related to hibernate properties START -->  <propertyname="show\_sql">true</property>  <propertyname="dialet">org.hibernate.dialect.OracleDialect</property>  <propertyname="hbm2ddl.auto">update</property>  <!-- Related to hibernate properties END -->    <!-- Related to mapping START -->  <mapping resource="Our mapping xml file name"/>  <!-- Related to the mapping END -->    </session-factory>  </hibernate-configuration> |

Usually configuration file name will be hibernate.cfg.xml

# Where To Download, How To Install Hibernate

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see what are the jar files we need to download to work with hibernate framework, and how to install.

Working with the framework software is nothing but, adding the .jar(s) files provided by that framework to our java application.  Each framework software is not an installable software, it means we do not contain any setup.exe  :-)

When we download any framework software, we will get a ‘zip‘ file and we need to unzip it, to get the jar files required, actually all framework softwares will follow same common principles like…

* Framework software will be in the form of a set of jar files, where one jar file acts as main**(**We can call this file as core**)** and remaining will  acts as dependent jar files.
* Each Framework software contain at least one configuration xml file, but multiple configuration files also allowed.
* In this case, in order to setup the Hibernate framework environment into a java application, the configuration file is the first one to be loaded into a java application, will see about this in later sessions.

# Steps To Use Hibernate In Any Java Application

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Hello mates, this is the exact flow of any hibernate application. so you must put little more concentration while you are reading this post, to understand better.

Whether the java application will run in the server or without server, and the application may be desktop or stand alone, swing, awt, servlet…what ever, but the steps are common to all.

In order to work with hibernate we don’t required any server as mandatory but we need hibernate software (.jar(s) files).

## Follow The Steps:

**1.** Import the hibernate API, they are many more, but these 2 are more than enough…

import org.hibernate.\*;  
import org.hibernate.cfg.\*;

**2.** Among Configuration, Mapping xml files, first we need to load configuration xml, because once we load the configuration file, automatically mapping file will be loaded as we registered this mapping xml in the configuration file.

So to load configuration xml, we need to create object of **Configuration** class, which is given in **org.hibernate.cfg.\***;  and we need to call **configuration()** method in that class, by passing xml configuration file name as parameter.

Eg:

Configuration **cf** = new Configuration ();  
cf.configure(“hibernate.cfg.xml”);

Hear our configuration file name is your choice, but by default am have been given hibernate.cfg.xml,  so once this configuration file is loaded in our java app, then we can say that hibernate environment is started in our program.

So once we write the line\_ **cf.configure(“hibernate.cfg.xml”),**configuration object**cf**will reads this xml file**hibernate.cfg.xml,**actually internally cf will uses DOM parsers to read the file.

Finally…

* cf will reads data from hibernate.cfg.xml
* Stores the data in different variables
* And finally all these variables are grouped and create one high level hibernate object we can call as SessionFactory object.
* So Configuration class only can create this SessionFactory object

Like SessionFactory **sf** =  cf.buildSessionFactory();

Actually SessionFactory is an interface not a class, and

SessionFactoryImpl is the implimented class for SessionFactory, so we are internally creating object of SessionFactoryImpl class and storing in the interface reference, so this SessionFactory object **sf**contains all the data regarding the configuation file so we can call**sf**as heavy weight object.

3. Creating an object of session,

* Session is an interface and SessionImpl is implemented class, both are given in org.hibernate.\*;
* When ever session is opened then internally a database connection will be opened, in order to get a session or open a session we need to call openSession() method in SessionFactory, it means SessionFactory produces sessions.

Session **session** = **sf**.openSession();

**sf** = SessfionFactory object

4. Create a logical transaction

While working with insert, update, delete, operations from an hibernate application onto the database then hibernate needs a logical Transaction, if we are selecting an object from the database then we do not require any logical transaction in hibernate.  In order to begin a logical transaction in hibernate then we need to call a method beginTransaction() given by Session Interface.

Transaction tx = **sessio**n.beginTransaction();

**session** is the object of Session Interface

5. Use the methods given by Session Interface,  to move the objects from application to database and  from database to application

|  |  |  |
| --- | --- | --- |
| session .save(s) | - | Inserting object ‘s’ into database |
| session.update(s) | - | Updating object ‘s’ in the database |
| session.load(s) | - | Selecting objcet ‘s’ object |
| session.delete(s) | - | Deletig object ‘s’ from database |

* So finally we need to call **commit()**in Transaction, like **tx.commit()**;
* As i told earlier,  when we open session a connection to the database will be created right, so we must close that connection as session. close().
* And finally close the SessionFactory as **sf.close()**
* That’s it.., we are done.

Final flow will be\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Configuration

SessionFactory

Session

Transaction

Close Statements

**what is Session factory in Hibernate?**

Here we can explain what is SessionFactory..

SessionFactory is an interface, which is available in “org.hibernate” package.

* Session factory is long live multithreaded object.
* Usually one session factory should be created for one database.
* When you have multiple databases in your application you should create multiple SessionFactory object.
* Assume the scenario that you are using one database called mysql in your application then following is the way to create the SessionFactory object.  
  Configuration cfg=new Configuration();  // Empty object will be created.  
  cfg=cfg.configure();  
    
  Here when you called  configure()  method  It looks for hibernate cfg.xml and  for Hibernate mapping file.  
  Filled with all the properties defined in the configuration documents and mapping documents  
     SessionFactory sc=cfg.buildSessionFactory();
* SessionFactory object will be created once and will be used by multiple users for long time.
* Session Factory object is the factory for session objects.

If you are using  two databases called mysql and oracle in your hibernate application then you need to build 2 SessionFactory object

Configuration cfg=**new** Configuration();

Configuration cfg1=cfg.configure(“mysql.cfg.xml”);

SessionFactory sf1=cfg1.builed SessionFactory();

 Configuration cfg2=cfg.configure(“oracle.cfg.xml”);

SessionFactory sf2=cfg2.builed SessionFactory();

When we are using more than one database in our application than we use the HibernateUtil class which is implemented based on singleton design pattern which insure that one and only one session Factory object will be created for entire application

# Hibernate Versioning Example, Hibernate Versioning Of Objects

Once an object is saved in a database, we can modify that object any number of times right, If we want to know how many no of times that an object is modified then we need to apply this versioning concept.  
When ever we use versioning then hibernate inserts version number as **zero**, when ever object is saved for the first time in the database.  Later hibernate increments that version no by one automatically whenever a modification is done on that particular object.  
In order to use this versioning concept, we need the following two changes in our application

* Add one property of type int in our pojo class
* In hibernate mapping file, add an element called version soon after id element

Files required to execute this program..

* Product.java (My POJO class)
* Product.hbm.xml  (Xml mapping file )
* hibernate.cfg.xml  (Xml configuration file)
* ClientForSave\_1.java (java file to write our hibernate logic)
* ClientForUpdate\_2.java

## Product.java

|  |  |
| --- | --- |
|  |  |
|  | <?xmlversion="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">   <hibernate-mapping>  <classname="str.Product"table="products">   <id name="productId"column="pid"  />  <version name="v"column="ver"/>  <propertyname="proName"column="pname"length="10"/>  <propertyname="price"/>   </class>  </hibernate-mapping> |

**Note**:

* Remember friends, first we must run the logic to **save** the object then hibernate will inset 0 (Zero) by default in the version column of the database, its very important point in the interview point of view also
* First save logic to let the hibernate to insert zero in the version column, then any number ofupdate logic’s (programs) we run, hibernate will increments +1 to the previous value
* But if we run the update logic for the first time, hibernate will not insert zero..! it will try toincrement the previous value which is NULL in the database so we will get the exception.

# Hibernate Lifecycle Of pojo Class Objects

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Actually our POJO class object having 3 states like…

* Transient state
* Persistent state
* Detached state

## Transient & Persistent states:

* When ever an object of a pojo class is created then it will be in the Transient state
* When the object is in a Transient state it doesn’t represent any row of the database, i mean not associated with any Session object, if we speak more we can say no relation with the database its just an normal object
* If we modify the data of a pojo class object, when it is in transient state then it doesn’t effect on the database table
* When the object is in persistent state, then it represent one row of the database, if the object is in persistent state then it is associated with the unique Session
* if we want to move an object from persistent to detached state, we need to do either closing that session or need to clear the cache of the session
* if we want to move an object from persistent state into transient state then we need to delete that object permanently from the database

## Example\_\_\_\_ ClientProgram.java

|  |  |
| --- | --- |
|  | importorg.hibernate.\*;  importorg.hibernate.cfg.\*;   public class ClientProgram {       public static void main(String[] args)      {        Configuration cfg = new Configuration();          cfg.configure("hibernate.cfg.xml");           SessionFactory factory = cfg.buildSessionFactory();          Session session = factory.openSession();            // Transient state\_\_\_\_\_start          Product p=newProduct();          p.setProductId(101);          p.setProName("iPhone");          p.setPrice(25000);           // Transient state\_\_\_\_\_end            // Persistent state\_\_\_\_\_start          Transaction tx = session.beginTransaction();          session.save(p);          System.out.println("Object saved successfully.....!!");          tx.commit();           // Persistent state\_\_\_\_\_end           session.close();          factory.close();      }  } |

Note:

see the above client program, line numbers **16** to **19** we just loaded the object and called the corresponding setter methods, its not related to the database row

* if you see, line number **24** we called save method in the Session Interface, means the object is now having the relation with the database
* if we want to convert the object from Transient state to Persistentstate we can do in 2 ways
* By saving that object like above
* By loading object from database
* If we do any modifications all the changes will first applied to the object in session cache only (Let\_\_ we do the modifications 5 times, then 5 times we need to save the changes into the database right, which means number of round trips from our application to database will be increased, Actually if we load an object from the database, first it will saves in the cache-memory so if we do any number of changes all will be effected at cache level only and finally we can call save or update method so with the single call of save or update method the data will be saved into the database.
* If we want to save an object into database then we need to call any one of the following 3 methods
* save()
* persist()
* saveOrUpdate()

i will explain about persist, saveOrUpdate methods later….

**Here is the difference between save and persist method:**

1. First difference between save and persist is there return type.
2. The return type of persist method is void
3. while return type of save method is Serializable object.
4. But both of them also INSERT records into database
5. Another difference between persist and save is that both methods make a transient object to persistent state. However, persist() method doesn’t guarantee that the identifier value will be assigned to the persistent state immediately, the assignment might happen at flush time.
6. Third difference between save and persist method in Hibernate is behavior on outside of transaction boundaries. persist() method will not execute an insert query if it is called outside of transaction boundaries. Because save() method returns an identifier so that an insert query is executed immediately to get the identifier, no matter if it are inside or outside of a transaction.
7. Fourth difference between save and persist method in Hibernate: persist method is called outside of transaction boundaries, it is useful in long-running conversations with an extended Session context. On the other hand save method is not good in a long-running conversation with an extended Session context.

If we want to load an object from database, then we need to call either load() or get() methods

## Transient:

One newly created object, without having any relation with the database, means never persistent, not associated with any Session object

## Persistent:

Having the relation with the database, associated with a unique Session object

## Detached:

previously having relation with the database [persistent ], now not associated with any Session

see the next sessions for the better understanding of the life cycle states of pojo class object(s) the hibernate

# Inheritance Mapping In Hibernate – Introduction

Compared to JDBC we have one main advantage in hibernate, which is hibernate inheritance.  Suppose if we have base and derived classes, now if we save derived(sub) class object, base class object will also be stored into the **database.**  
But the thing is we must specify in what table we need to save which object data ( i will explain about this point later, just remember as of now).

**Hibernate supports 3 types of Inheritance Mappings**:

* Table per class hierarchy
* Table per sub-class hierarchy
* Table per concrete class hierarchy

**Note**: We can also called this Hibernate Inheritance Mapping as Hibernate Hierarchy

Will see these 3 Inheritance Mappings in depth\_\_\_, **friends ensure** you are clear about all previous concepts so far we covered, if not so you may not understand further, please refer once if you have any doubts.

# Hibernate Inheritance: Table Per Class Hierarchy

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Hear is the explanation and one example on hibernate table per class hierarchy, consider we have base class named Payment and 2 derived classes like CreditCard, Cheque

If we save the **derived class** object like CreditCard or Cheque then automatically Payment class object will also be saved into the database, and in the database all the data will be stored into a **single table** only, which is base class table for sure.

But hear we must use one extra discriminator column in the database,  just to identify which derived class object we have been saved in the table along with the base class object,  if we are not using this column hibernate will throws the exception, see this example so that you will get one idea on this concept.

**Mapping Files:-**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | <?xmlversion="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">   <hibernate-mapping>   <classname="str.Payment" table="PAYMENT">  <idname="paymentId"column="pid"/>  **<discriminator column="dcolumn" type="string" length="5"/>**  <property name="amount" column="amt"/>    <subclass name="str.CreditCard"discriminator-value="CC">  <propertyname="CreditCardType"column="cctype"length="10"/>  </subclass>   <subclassname="str.Cheque"discriminator-value="cq">  <propertyname="ChequeType"column="cqtype"length="10"/>  </subclass>   </class>  </hibernate-mapping> Hibernate Inheritance: Table Per subClass Hierarchy  |  |  | | --- | --- | | [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |   This is also just like previous example, but some changes are there, in table per class hierarchy all the data was saved in a single table but hear,  x number of classes = x number of tables in the database  If we save the CreditCard class object, then first hibernate will saves the data related to super class object into the super class related table in the database and then CreditCard object data in CreditCard related table in the database, so first base class data will be saved  Required files\_   * Payment.java (Base class) * CreditCard.java (Derived class) * Cheque.java (Derived class) * ClientForSave.java (for our logic) * Payment.hbm.xml * hibernate.cfg.xml   All are same but mapping file is different than previous example.. Payment.hbm.xml:  |  |  | | --- | --- | |  | <?xmlversion="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">    <hibernate-mapping>    <classname="str.Payment"table="PAYMENT">    <idname="paymentId"column="pid"/>  <propertyname="amount"column="amt"/>    <joined-subclassname="str.CreditCard"table="CreditCard">  <keycolumn="dummy1"/>  <propertyname="CreditCardType"column="cctype"length="10"/>  </joined-subclass>    <joined-subclassname="str.Cheque"table="Cheque">  <keycolumn="dummy2"/>  <propertyname="ChequeType"column="cqtype"length="10"/>  </joined-subclass>    </class>  </hibernate-mapping> | |

# Hibernate Inheritance: Table Per Concrete Class Hierarchy

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Something like previous example but the changes are at mapping file only, and one more thing is..

x number of derived classes = x number of tables in the database

* Once we save the derived class object, then derived class data and base class data will be saved in the derived class related table in the database
* for this type we need the tables for derived classes, but not for the base class
* in the mapping file we need to use one new element <union-subclass — >under <class —>

Required files\_

* Payment.java (Base class)
* CreditCard.java (Derived class)
* Cheque.java (Derived class)
* ClientForSave.java (for our logic)
* Payment.hbm.xml
* hibernate.cfg.xml

 ­Payment.hbm.xml:

|  |  |
| --- | --- |
|  | <?xmlversion="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">   <hibernate-mapping>   <classname="str.Payment"table="PAYMENT">   <idname="paymentId"column="pid"/>  <propertyname="amount"column="amt"/>    <union-subclassname="str.CreditCard">  <propertyname="CreditCardType"column="cctype"length="10"/>  </union-subclass>   <union-subclassname="str.Cheque">  <propertyname="ChequeType"column="cqtype"length="10"/>  </union-subclass>  </class>  </hibernate-mapping> |

|  |
| --- |
|  |

# Example On Composite Primary Keys In Hibernate

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Composite primary keys means having more than one primary key, let us see few points on this concept

* If the table has a**primary**key then in the hibernate mapping file we need to configure that column by using <id /> element right..!
* Even though the database table doesn’t have any primary key, we must configure one column as id (one primary key is must)
* If the database table has more than one column as primary key then we call it as composite primary key, so if the table has multiple primary key columns , in order to configure these primary key columns in the hibernate mapping file we need to use one new element called **<composite-id** …..> **</composite-id>**

## Product.hbm.xml

|  |  |
| --- | --- |
|  | <?xmlversion="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">    <hibernate-mapping>  <classname="str.Product"table="products">   <composite-id>  <key-property name="productId"column="pid"  />  <key-propertyname="proName"column="pname"length="10"/>  </composite-id>   <propertyname="price"/>    </class>  </hibernate-mapping> |

**Notes**:

* see Product.java pojo class, in line number 3 i have implemented the **java.io.Serializable**interface,  this is the first time am writing this implementation for the pojo class right…!  we will see the reason why we use this serializable interface later.
* But remember, if we want to use the composite primary keys we must implement our pojo class with **Serializable** interface
* **hibernate.cfg.xml** is normal as previous programs, something like hello world program
* come to **Product.hbm.xml**, see line number **9**-**12,**this time we are using one new element<**composite-id**>
* Actually if we have a single primary key, we need to use <id> element, but this time we have multiple primary keys, so we need to use this new element <composite-id>
* Actually we will see the exact concept of this composite primary keys in the next example (loading an object with composite key)

# Generators <generator> In Hibernate

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

<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

## List of generators

The following are the list of main generators we are using in the hibernate framework

* assigned
* increment
* sequence
* identify
* hilo
* native
* foregin
* uuid.hex
* uuid.string

In the above generators list, the first **7** are used for int,long,short types of primary keys, and last **2** are used when the primary key column type is String type (varchar2)

* This generator supports in all the databases
* 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

|  |  |
| --- | --- |
| 1  2  3 | <idname="prodId"column="pid">  <generator/>  </id> |

## Increment

* This generator supports in all the databases, database independent
* This generator is used for generating the id value for the new record by using the formula

Max 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

## sequence

* Not has the support with**MySql**
* This generator class is database dependent it means, we cannot use this generator class for all the database, we should know whether the database supports sequence or not before we are working with it
* 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

|  |  |
| --- | --- |
| 1  2  3  4  5 | <id name="productId"column="pid">  <generator>  <paramname="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 than 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

sql> create sequence MySequence incremented by 5;

* first it will starts with 1 by default
* though you send the primary key value., hibernate uses this sequence concept only
* But if we not create any sequence, then first 1 and increments by 1..bla bla. in this case hibernate creating right..? so ensure we have hbm2ddl.auto enabled in the configuration file

## identity

* This is database dependent, actually its not working in oracle
* In this case (identity generator) the id value is generated by the database, but not by the hibernate, but in case of increment hibernate will take over this
* 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

|  |  |
| --- | --- |
| 1  2  3 | <id name="productid"column="pid">  <generator class="......."/>  </id> |

As this is not working in Oracle, if you would like to check this in MySql you must change the configuration file as…….

**class**: com.mysql.jdbc.Driver  
**url**: jdbc:mysql://www.java4s.com:3306/test (test is default database)  
**user**: root (default)  
**pass**: (default)  
**dialet**: org.hibernate.dialet.MySQLDialet

**Note**:

* jar file required (in class path we must set..  
  **mysql-connector-java-3.0.8-stable-bin.jar** (version number may change)
* Actually this jar will never come along with mysql database software,  to get this jar file we need to download the following file, and unzip it.**mysql-connectar-java-3.0.8-stable.zip**

# Part 1 Hibernate Query Language Introduction

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

So far we done the operations on single object (single row), hear we will see modifications, updates onmultiple rows of data (multiple objects) at a time.  In hibernate we can perform the operations on a single row (or) multiple rows at a time, if we do operations on multiple rows at once, then we can call this as bulk operations.

* HQL is the own query language of hibernate and it is used to perform bulk operations on hibernate programs
* An object oriented form of SQL is called HQL
* hear we are going to replace table column names  with POJO class variable names and table names with POJO class names in order to get HQL commands

## Advantages Of HQL:

* HQL is database independent, means if we write any program using HQL commands then our program will be able to execute in all the databases with out doing any further changes to it
* HQL supports object oriented features like ***Inheritance***, ***polymorphism***, ***Associations***(Relation ships)
* HQL is initially given for selecting object from database and in hibernate 3.x we can do **DML**operations ( insert, update…) too

## Different Ways Of Construction HQL Select

* If we want to select a **Complete Object** from the database, we use POJO class reference in place of   **\***  while constructing the query
* In this case (select a complete object from the database) we can directly start our HQL command from,  **from** key word

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // InSQL  sql> select\* fromProduct  Note: Product isthe tablenameright....!!!    // InHQL  hql> selectp fromProduct p       [ or]       fromProduct p  Note: hear p isthe reference...!! |

* If we want to load the **Partial Object** from the database that is only selective properties (selected columns) of an objects then we need to replace column names with POJO class variable names.

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | // InSQL  sql> selectpid,pname fromProduct  Note: pid, pname are the columns Product isthe tablenameright..!    // InHQL  hql> selectp.productid,p.productName fromProduct p       [ or]       fromProduct p ( we should notstart from, fromkeyword hear because  we selecting the columns hope you are getting me )    Note: hear p isthe reference...!!             productid,productName are POJO variables |

**Next Session HQL part 2**

# Part 2 Hibernate Query Language, Executing HQL Commands

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see, how to execute HQL commands..

## Procedure To Execute HQL Command:

* If we want to execute an HQL query on a database, we need to create a query object
* ” Query ” is an interface given in org.hibernate package
* In order to get query object, we need to call createQuery() method in the session Interface
* Query is an interface, QueryImpl is the implemented class
* we need to call list method for executing an HQL command on database, it returns java.util.List
* we need to use java.util.Iterator for iterating the List collection

**Syntax:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Query qry = session.createQuery("--- HQL command ---");  List l = qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {    Object o = it.next();    Product p = (Product)o;    ----- ------- ---------  } |

**Notes:**

* line 1: Getting the Query object
* line 2: Executing the object content (which is HQL command)
* line 3: Iterating, then while loop and type cast into our class type that’s it

**Next Session HQL part 3**

|  |
| --- |
|  |

# Part 3 HQL, Different Ways Of Executing HQL Commands

We can execute our HQL command in 3 ways,  like by **selecting total object**, partial object (more than one column), partial object (with single column).  Let us see..

## Product.hbm.xml

|  |  |
| --- | --- |
|  | <?xmlversion="1.0"?>  <!DOCTYPE hibernate-mapping PUBLIC  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"  "<http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd>">  <hibernate-mapping>  <classname="str.Product"table="products">    <idname="productId"column="pid"  />  <propertyname="proName"column="pname"length="10"/>  <propertyname="price"/>    </class>  </hibernate-mapping> |

## ForOurLogic.java

|  |  |
| --- | --- |
|  | packagestr;   publicclassForOurLogic {       publicstaticvoidmain(String[] args)      {            Configuration cfg = newConfiguration();          cfg.configure("hibernate.cfg.xml");            SessionFactory factory = cfg.buildSessionFactory();          Session session = factory.openSession();    /\* Selecting all objects(records) start\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \*/        /\*            Query qry = session.createQuery("from Product p");            List l =qry.list();          System.out.println("Total Number Of Records : "+l.size());          Iterator it = l.iterator();            while(it.hasNext())          {              Object o = (Object)it.next();              Product p = (Product)o;              System.out.println("Product id : "+p.getProductId());              System.out.println("Product Name : "+p.getProName());              System.out.println("Product Price : "+p.getPrice());              System.out.println("----------------------");          }        \*/    /\* Selecting all objects(records) end\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \*/    /\* Selecting partial objects(More than one object) start\_\_\_\_\_\_\_\_\_\_ \*/        /\*    Query qry = session.createQuery("select p.productId,p.proName from Product p");            List l =qry.list();          System.out.println("Total Number Of Records : "+l.size());          Iterator it = l.iterator();            while(it.hasNext())          {              Object o[] = (Object[])it.next();               System.out.println("Product id : "+o[0]+ "Product Name : "+o[1]);               System.out.println("----------------");          }           \*/  /\* Selecting partial objects(More than one object)end\_\_\_\_\_\_\_\_\_\_\_\_\_ \*/    // Selecting single object start\_\_\_\_\_\_\_\_\_\_\_\_\_            Query qry = session.createQuery("select p.productId  from Product p");            List l =qry.list();          System.out.println("Total Number Of Records : "+l.size());          Iterator it = l.iterator();            while(it.hasNext())          {              Integer i = (Integer)it.next();              System.out.println("Product id : "+i.intValue());              System.out.println("---------------------------");            }    // selecting single object end\_\_\_\_\_\_\_\_\_\_\_\_            session.close();          factory.close();      }    } |

# Criteria Query, Hibernate Criteria Query Introduction

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Unlike HQL, Criteria is only for selecting the data from the database, that to we can select complete objects only not partial objects, in fact by combining criteria and projections concept we can select partial objects too we will see this angle later,  ;) but for now see how we are using criteria for selecting complete objects form the database. We cant perform non-select operations using this criteria.  Criteria is suitable for executing dynamic queries too, let us see how to use this criteria queries in the hibernate..

**syntax:**

Criteria crit = session.createCriteria(–Our class object–);

**Usage:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | Criteria crit = session.createCriteria(Employee.class);  // let Product is our pojo class  List l = crit.list()  // need to call list() to execute criteria  Iterator it = l.iterator();  while(it.hasNext())  {  Object o = it.next();  Product p = (Product)o;  ------ ----- -----  } |

## Adding Conditions To Criteria

* If we want to put conditions to load data from database, using criteria then we need to create one Criterion Interface object and we need to add this object to Criteria Class object

crit.add(**Criterion Interface Object**)  
crit = criteria class object

* Criterion is an interface given in “org.hibernate.criterion” package
* In order to get Criterion object, we need to use Restrictions class
* Restrictions is the factory for producing Criterion objects, but friends there is no explicit relation between Criterion interface and Restrictions class, it means Restrictions class is not implemented from Criterion Interface
* In Restrictions class, we have all static methods and each method of this class returns Criterion object
* Restrictions class is also given in “org.hibernate.criterion” package

**Usage:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | Criteria crit = session.createCriteria(Products.class);  Criterion c1=Restrictions.gt("price", newInteger(12000));  //price is our pojo class variable  crit.add(c1); // adding criterion object to criteria class object  List l = crit.list(); // executing criteria query |

Note: See line number 2, am calling gt(-,-) method of Restrictions class, (means greater than), in our above example am fetching the data by comparing price greater than (>) 12000

* If we want to put more conditions on the data (multiple conditions) then we can use **and** method ,**or** method give by the Restrictions class

**Usage:**

|  |  |
| --- | --- |
| 1  2  3  4 | crit.add(Restrictions.and(Restrictions.like("proName","%R%"),  Restrictions.eq("price",newInteger(12000))));  List l=crit.list();  Iterator it = l.iterator(); |

Like this we can add any number of conditions…

**Let us see an example program on HQL criteria  in the next session…**

## ForOurLogic.java

|  |  |
| --- | --- |
|  | packagestr;    importjava.util.Iterator;  importjava.util.List;    importorg.hibernate.Criteria;  importorg.hibernate.Session;  importorg.hibernate.SessionFactory;  importorg.hibernate.cfg.Configuration;  importorg.hibernate.criterion.Criterion;  importorg.hibernate.criterion.Restrictions;    publicclassForOurLogic {        @SuppressWarnings("unchecked")      publicstaticvoidmain(String[] args)      {            Configuration cfg = newConfiguration();          cfg.configure("hibernate.cfg.xml");            SessionFactory factory = cfg.buildSessionFactory();          Session session = factory.openSession();            Criteria crit = session.createCriteria(Product.class);          Criterion cn = Restrictions.gt("price",newDouble(17000));          crit.add(cn);          List l=crit.list();          System.out.println("List total size...\_"+l.size());          Iterator it=l.iterator();            while(it.hasNext())          {              Product p=(Product)it.next();              System.out.println(p.getProductId());              System.out.println(p.getProName());              System.out.println(p.getPrice());              System.out.println("-----------------");          }            session.close();          factory.close();      }    } |

**Note:**

* In line number 25, created object of Criteria
* In line number 26,  created Criterion interface object by using Restrictions class
* In line number 27, added criterion interface object to criteria object
* In line number 28,  executed criteria query by calling list() method in criteria
* See line number 34, in this case we must typecast into our POJO class type only

Actually the internal concept is,  once we called the list() method in criteria (line number 28) all objects (records) will come and stores in list object, from there we used to take iterate then typecast into our POJO class type bla bla.

# Hibernate Criteria, Adding Conditions To Criteria

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

If we want to add some sorting order for the objects, before the objects are going to store in list object then we need to add an Order class object to the Criteria class object by calling addOrder() method..,

* Order is a class given in “org.hibernate.Criterion” package
* In Order class, we have 2 static methods, asc()[ascending order] and dsc()[descending order] for getting an objects in required order
* Internal concept is, hibernate will select the records (rows) from PRODUCT table and stores them into a ResultSet and then converts each row data of resultset into a POJO class object basing on our field type, then all these objects into a list according to the order you have given

**Hibernate Projections Introduction**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

So far in criteria, we are able to load complete object right….! let us see how to load the partial objects while working with criteria.  The projections concept is introduced in hibernate 3.0 and mainly we can do the following 2 operations using the projection

* We can load partial object from the database
* We can find the Result of Aggregate functions

*Projection* is an Interface given in “org.hibernate.criterion” package, *Projections* is an class given in same package,  actually Projection is an interface, and Projections is an class and is a factory for producing projection objects.

In Projections class, we have all static methods and each method of this class returns Projection interface object.

If we want to add a Projection object to Criteria then we need to call a method **setProjection()**

**Remember**, while adding projection object to criteria, it is possible to add one object at a time.  It means if we add 2nd projection object then this 2nd one will overrides the first one (first one wont be work), so at a time we can only one projection object to criteria object.

Using criteria, if we want to load partial object from the database, then we need to create a projection object for property that is to be loaded from the database

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | Criteria crit = session.createCriteria(Products.class);  crit.setProjection(Projections.proparty("proName"));  List l=crit.list();  Iterator it=l.iterator();  while(it.hasNext())  {  String s = (String)it.next();  // ---- print -----  } |

If we add multiple projections to criteria then the last projection added will be considered to execute see…

**Example:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Criteria crit = session.createCriteria(Products.class);    Projection p1 = Projection.property("proName");  Projection p2 = Projection.property("price");    crit.setProjection(p1):  crit.setProjection(p2):  List l=crit.list();  -------------- ----  ------- - ---  ---- |

Hear collections list l, is going to contain the price in the form of Double objects, but product names are over ridden,  second projection over rides the first one, i mean p2 only will works p1 will not works,actually there is a way to add multiple projections to criteria to select more than one column from the database i will tell you in the next example :-)

# Example On Hibernate Criteria With Multiple Projections

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

If we want to load partial object, with multiple columns using criteria then we need to create theProjectionList with the multiple properties and then we need to add that ProjectionList to the criteria

files required…

|  |
| --- |
|  |

**Difference between HQL and Criteria Query in Hibernate**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see the main differences between HQL and Criteria Query

* HQL is to perform both select and non-select operations on the data,  but Criteria is only for selecting the data, we cannot perform non-select operations using criteria
* HQL is suitable for executing Static Queries, where as Criteria is suitable for executing Dynamic Queries
* HQL doesn’t support pagination concept, but we can achieve pagination with Criteria
* Criteria used to take more time to execute then HQL
* With Criteria we are safe with SQL Injection because of its dynamic query generation but in HQL as your queries are either fixed or parametrized, there is no safe from SQL Injection.

# Hibernate Native SQL Query Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Native SQL is another technique of performing bulk operations on the data using hibernate

* By using Native SQL, we can perform both select, non-select operations on the data
* In face Native SQL means using the direct SQL command specific to the particular (current using) database and executing it with using hibernate

## Advantages and Disadvantages of Native SQL

* We can use the database specific keywords (commands), to get the data from the database
* While migrating a JDBC program into hibernate, the task becomes very simple because JDBC uses direct SQL commands and hibernate also supports the same commands by using this Native SQL
* The main draw back of Native SQL is, some times it makes the hibernate application as database dependent one

If we want to execute Native SQL Queries on the database then, we need to construct an object of SQLQuery, actually this SQLQuery is an interface extended from Query and it is given in ” org.hibernate package ”

In order to get an object of SQLQuery, we need to use a method createSQLQuery() given by session interface.

While executing native sql queries on the database, we use directly tables, column names directly in our command.

**Remember**, while executing Native SQL Queries, even though we are selecting complete objects from teh database we need to type cast into object array only, not into our pojo class type, because we are giving direct table, column names in the Native SQL Querie so it does’nt know our class name

If we execute the command, always first it will put’s data in **ResultSet** and from there List

**Usage:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | SQLQuery qry = session.createSQLQuery("select \* from PRODUCTS");  // Hear PRODUCTS is the table in the database...  List l = qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {  Object row[] = (Object[])it.next();  --- -------  } |

* while selecting data from the table, even though you are selecting the complete object from the table, in while loop still we type cast into object array only right
* See the above code, we typecast into the object[] arrays right..,  in case if we want to type cast into our POJO class (i mean to get POJO class obj), then we need to go with entityQueryconcept
* In order to inform the hibernate that convert each row of ResultSet into an object of the POJO class back, we need to make the query as an entityQuery
* to make the query as an entityQuery, we need to call addEntity() method

**Usage:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | //We are letting hibernate to know our pojo class too  SQLQuery qry = session.createSQLQuery("select \* from PRODUCTs").addEntity(Product.class);  List l = qry.list();  Iterator it = l.iterator();  while(it.hasNext())  {  Product p = (Product)it.next();  --- -------  } |

**Notes:**

* See line number 2, i have been added addEntity(Product.class) at the end, which will let the hibernate to know about our POJO class, so now we can typecast into our POJO class type like what i have done at  line number 7
* And that’s it, this is the total concept on this Native SQL, am not going to give any example on this separately hope you understood the concept

|  |
| --- |
|  |

# Hibernate Named Query Introduction Tutorial

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see few points, before going to see an example on Named Queries in HIbernate..

* 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
* 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()
* If you want to create Named Query then we need to use **query element** in the hibernate mapping file

## Syntax Of hibernate mapping file [For HQL]

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <hibernate-mapping>    <classname="---"table="---">     <idname="---"column="---"  />     <propertyname="---"column="---"length="10"/>     <propertyname="---"column="---"/>     --- --- --- ---  </class>    <queryname="Give Query Name">      <![CDATA[from Product p where p.price = :java4s]]>  </query>    </hibernate-mapping> |

**Notes:**

* See line numbers 10,11,12, this is the new element we have to add to work with Named Queries
* there colon (:) java4s is the label, i will pass the value into that label in the run time.., or let us see the client program logic too

## Example Logic in Application:

|  |  |
| --- | --- |
| 1  2  3 | Query qry = session.getNamedQuery("Name we given in hibernate mapping xml");  qry.setParameter("java4s",newInteger(1022));  List l = qry.list(); |

**Notes**:

* Line number 1, getting the query from hibernate mapping file to our client program
* Line number 2, passing run time value to that query
* Line number 3, calling list() method to execute the query

Up to now this is the case if we use HQL query in hibernate mapping file, let us see the case if we would like to use nativeSQL query

## Syntax Of hibernate mapping file [For Native SQL]

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | <hibernate-mapping>    <classname="---"table="---">     <idname="---"column="---"  />     <propertyname="---"column="---"length="10"/>     <propertyname="---"column="---"/>     --- --- --- ---  </class>    <sql-queryname="Give Query Name">      select \* from PRODUCTS  </sql-query>    </hibernate-mapping> |

**Notes:**

* If we want to give HQL query in hiberante mapping file, we need to use <query/> element, but we have to use <sql-query /> element in case of Native SQL
* See line number 11, its the normal sql command, and PRODUCTS is the table name, not the pojo class name :-)

Done…!!!!!

**Hibernate In Servlet Example, Hibernate In Servlet Tutorial**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

With servlet, if we want to do some operations on the database, then we can also use hibernate ORMrather than JDBC.  We call this as servlet-Hibernate integration.

While integration servelt with hibernate, it is there to follow these setps

* In init() method of servlet, we need to create SessionFactory of hibernate, because SessionFactory is an heavy weight component, its the programmer responsibulity to make it assingleton, for this we need to create the object of SessionFactory in init() method of servelet
* Open a Session in service() method and perform the operations on the database close the Session in service()
* Close the SessionFactory of hibernate in destroy() method
* While integration servlet with hibernate, hibernate mapping files (.hbm.xml) and hibernate configure files(cfg.xml) need to be stored in classes folder only
* In the lib folder, we need to store all the jars related to hibernate and database

Let us see an example on this servelt with  hibernate in the next session

# Example On Hibernate Pagination With Servlet In Eclipse

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see an example on hibernate pagination with servlet..

* when response for request is too large [If we have 1000's of records in the database] then instead of displaying all records at a time on browser  we can display the response page by page manner using pagination mechanism
* In pagination, initially one page response will be displayed and we will get links for getting thenext pages response
* In this servelt & hibernate integration, we are going to display  **4** records or **4** objects of products using hibernate for selecting the data and we will get links to display the records of the next pages

## Regarding Logic  In Order To Get pagination

* The servlet accepts pageIndex parameter and if the parameter is passed then servlet takes the given number as pageIndex, otherwise the servlet will takes the pageIndex as one [ 1  ]
* Servlet uses Criteria API and the pagination methods of Criteria for loading the records (objects) related to that particular page, and servlet display those records on the browser
* In servlet we use Criteria with projection for finding the total number of records available in the table, and we store that number into the variable
* We will find out the number of hyperlinks required by dividing the total number of records with records per page
* we need to use a loop in order to display the links on the browser, while creating each link,  we use the <a href  /> to servlet url pattern [Hiper reference] and by passing that page nomber as value for pageIndex parameter

**Hibernate Relationships In Depth**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Using hibernate, if we want to put relationship between two entities [ objects of two pojo classes ], then in the database tables, there must exist foreign key relationship, we call it as Referential integrity.

The main advantage of putting relation ship between objects is, we can do operation on one object, and the same operation can transfer onto the other object in the database [ remember, object means one row in hibernate terminology ]

While selecting, it is possible to get data from multiple tables at a time if there exits relationship between the tables, nothing but in hibernate relationships between the objects

Using hibernate we can put the following 4 types of relationships

* One-To-One
* Many-To-One
* Many-To-Many
* One-To-Many

Let’s learn all the Hibernate Relations in detail….

* Hibernate One-to-One mapping explanation with complete example
* Hibernate Many- to-One mapping explanation with complete example
* Hibernate Many-to-Many mapping explanation with complete example
* Hibernate One-to-Many mapping explanation with complete example

# Hibernate One to Many Mapping Insert Query Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

**One-to-Many**:  according to database terminology, one row of table related with multiple rows of other table

[**or**]

According to hibernate, one object of one pojo class related to multiple objects of other pojo

I mean, one [parent] to many [Children], example of one-to-many is some thing category books contains different type of books, one vendor contains lot of customers bla bla.

To achieve one-to-many between two pojo classes in the hibernate, then the following two changes are required

* In the parent pojo class, we need to take a collection property, the collection can be eitherSet,List,Map (We will see the example on separate collection later)
* In the mapping file of that parent pojo class, we need to configure the collection

I will take this vendor-customer as an example..

|  |
| --- |
|  |

# Hibernate One to Many Select Query Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see the logic for hibernate one to many mapping select query,

hibernate one to many select means, if you select the parent object then automatically its corresponding child objects will also be selected, see i have given for the logic for selecting single parent object with all its childs & all parent objects with all child objects

But mates, ensure you came through these sessions for better understand

* [**Hibernate One-to-Many Mapping Insert**](http://www.java4s.com/hibernate/hibernate-one-to-many-mapping-insert/)

[**Hibernate Relationships In Depth**](http://www.java4s.com/hibernate/hibernate-relationships/)

# Hibernate Many to One Mapping Insert Query Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see how to achieve hibernate many to one mapping with insert query, just go through few points before we start the example

* In the many to one relationship, the relationship is applied from child object to parent object, but in one to may parent object to child object right..! just remember
* many to one is similar to one to many but with the little changes
* If we want to apply many to one relationship between two pojo class objects then the following changes are required

In the child pojo class, create an additional property of type parent for storing the parent object in child object [ If you are confused just remember you will be able to understand in the example ], in the child pojo class mapping file we need to write <many-to-one name=”"> for parent type property unlike <property name=”">

# Hibernate Many to One Mapping Select Query Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

In many to one relationship, when ever child object is loaded from the database then automatically theparent object will also be loaded from the database. Let us an example on selecting single child object with its parent object.

But remember, in many to one we can see **2** types of object loadings (selecting)

* proxy
* early loading

In many to one relationship, the lazy attribute (in mapping xml) values are either proxy or false.  If we write lazy=”false” then when ever child object is loading then immediately parent object will also be loading from the database.

The default value of lazy is proxy, means hear when ever child object is loaded then parent object is not loaded immediately, but a proxy object will be loaded with it (logical object)

When ever the parent object is accessed then at that moment that parent object will be loaded from the database.

**Note**: In our program am not using lazy attribute in xml files so just go ahead and insert this lazy attribute and test the output on the eclipse console

# Hibernate Many to One Mapping Delete Query Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see the example on hibernate many to one delete query…

* If we delete child, parent will not deleted because, it may have lot of other child objects
* In many to one relationship, when ever a child object is deleted then its parent object is also deleted, provided if  that parent object has no other child objects, means if parent has only one child, in this case if we delete child, parent will also got deleted, but in all other cases it willthrows exception

# Hibernate One To Many Bidirectional Mapping Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see how to achieve, Bidirectional one to many  mapping in hibernate…

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 right…?

**[ and ]**

Similarly in many to one the relation is from child  to parent object, hope you remembered this concept,  if not so just go back and have a look once..

Bidirectional one to many  >>>>  Combination of these above 2

Let us see an example…

**Notes:**

* See line number 59,60 actually we can save any object either parent or child [ as it is Bi directional inverse will take automatically ], but in our application i saved child object.
* In this above logic, even though we are saving a single child object, but in the database all child objects are inserted at the time of executing the code, the reason being… the time of saving c1 object, first its parent object **v**  will be inserted, as the parent object **v** has 3 child objects so hibernate will save all the 3 child objects in the database
* In **Vendor.hbm.xml**, we have included an attribute in the set element called inverse, this attribute informs the hibernate that the relation ship is Bi Directional
* If we write inverse = “false” then hibernate understands that relationship as unidirectional and generates additional update operations on the database, so in order to reduce the internal operations, we need to include inverse=”true“
* **remember**, default value of inverse =”false”
* If we make inverse =”true” the performance will be increased, i guess :-)

|  |
| --- |
|  |

# Hibernate Many to Many Mapping Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see an example on this many to many relationship in hibernate.  Actually hear there is no question of unidirectional, only Bi-Directional.

Applying many to many relationship between two pojo class objects is nothing but applying one to manyrelationship on both sides, which tends to Bi-Directional i mean many to many.

**Example:**

Let us see this, if we apply many to many association between two pojo class objects student andcourse, provided the relationship is one student may joined in multiple courses and one course contains lot of students (joined by multiple students)

**Remember**, when ever we are applying many to many relationship between two pojo class objects, on both sides  we need a collection property [As we are applying one to many from both the sides]

## Note Points:

* While applying many to many relationship between pojo classes,  a mediator table is mandatoryin the database, to store primary key as foreign key both sides, we call this table as Join table
* In many to many relationship join table contain foreign keys only

|  |
| --- |
|  |

# Hibernate Cascade Options – Cascade Attribute In Hibernate

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

## Cascade Attribute 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

If we write cascade = “all” then changes at parent class object will be effected to child class object too,  if we write cascade = “all” then all operations like insert, delete, update at parent object will be effected to child object also

Example: if we apply insert(or update or delete) operation on parent class object, then child class objects will also be stored into the database.

default value of cascade =”none” means no operations will be transfers to the child class

Example: if we apply insert(or update or delete) operation on parent class object, then child class objects will not be effected, if cascade = “none”

Cascade having the values…….

In hibernate relations, if we load one parent object from the database then child objects related to that parent object will be loaded into one collection right (see one-to-many insert example).

Now if we delete one child object from that collection, then the relationship between the parent object and that child object will be removed, but the record (object) in the database will remains at it is, so if we load the same parent object again then this deleted child will not be loaded [ but it will be available on the database ]

so finally what am saying is all-delete-orphan means, breaking relation between objects not deleting the objects from the database, hope you got what am saying ;)

**Note**:   
what is orphan record ..?  
an orphan record means it is a record in child table but it doesn’t have association with its parent in the application.

[ **And** ]

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”

And that’s it about this cascade attribute in hibernate, hope i explained all the values..!!

**Joins In Hibernate**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us see few points regarding this hibernate joins…., like why and where we need to us bla bla

* We use join statements, to select the data from multiple tables of the database, when there existrelationship
* with joins, its possible to select data from multiple tables of the database by construction a singlequery

Hibernate supports 4 types of joins..

* Left Join
* Right  Join
* Full Join
* Inner Join

the **DEFAULT** join in hibernate is Inner join

* Left join means, the objects from both sides of the join are selected and more objects  from leftside are selected, even though no equal objects are there at right side
* Right join means equal objects are selected from database and more objects are from right side of join are selected even though there is no equal objects are exist left side
* Full join means, both equal and un-equal objects from both sides of join are selected
* Inner join means only equal objects are selected  and the remaining are discarded
* At the time of construction the join statements, we need to use the properties created in pojo class to apply relationship between the objects
* To construct a join statement, we use either HQL, or NativeSql

Please find one by one hibernate join in the next sessions…

|  |
| --- |
|  |
|  |

# Hibernate Left Join, Hibernate Left Join Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Left join means, the objects from both sides of the join are selected and more objects  from left side are selected, even though no equal objects are there at right side, no confusion you will be able tounderstand if you go through this example i guess :-)

Let us see an example on hibernate left join, am taking **one-to-many** to explain this concept files required….

|  |
| --- |
|  |

**Hibernate Caching Mechanism, Hibernate Cache**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Every fresh session having its own cache memory, Caching is a mechanism for storing the loaded objects into a cache memory.  The advantage of cache mechanism is, whenever again we want to load the same object from the database then instead of hitting the database once again, it loads from the local cache memory only, so that the no. of round trips between an application and a database server got decreased.  It means caching mechanism increases the performance of the application.

In hibernate we have two levels of caching

* First Level Cache [ or ] Session Cache
* Second Level Cache [ or ] Session Factory Cache [ or  ] JVM Level Cache

Let us see one by one session cache concepts in depth.

# Hibernate First Level Cache Example

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Let us try to understand the first level cache in hibernate,  actually i tried to give almost all the concept about this first level cache hope you will enjoy this :-)

* By default, for each hibernate application, the first level cache is automatically been enabled
* As a programmer, we no need to have any settings to enable the first level cache and also we cannot disable this first level cache
* the first level cache is associated with the session object and scope of the cache is limited to one session only
* When we load an object for the first time from the database then the object will be loaded from thedatabase and the loaded object will be stored in the cache memory maintained by that sessionobject
* If we load the same object once again, with in the same session, then the object will be loaded from the local cache memory not from the database
* If we load the same object by opening other session then again the object will loads from the database and the loaded object will be stored in the cache memory maintained by this new session

## Example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | Session ses1 = factory.openSession();  Object ob1 = ses1.get(Student.class, newInteger(101));    Object ob2 = ses1.get(Student.class, newInteger(101));  Object ob3 = ses1.get(Student.class, newInteger(101));  Object ob4 = ses1.get(Student.class, newInteger(101));    --  session.close();    Session ses2 = factory.openSession();  Object ob5 = ses2.get(Student.class, newInteger(101)); |

**Explanation:**

* In line number1, i have opened one session with object is ses1
* In line number2, loaded one object with id 101, now it will loads the object from the database only as its the first time, and keeps this object in the session cache
* See at line number 4,5,6 i tried to load the same object 3 times, but hear the object will be loaded from the stored cache only not from the database, as we are in the same session
* In line number 9, we close the first session, so the cache memory related this session also will be destroyed
* See line number 11, again i created one new session and loaded the same object with id 101 inline number 12, but this time hibernate will loads the object from the database

Finally what am trying to tell you is…

x . number of sessions = that many number of cache memories

## Important

The loaded objects will be stored in cache memory maintained by a session object and if we want toremove the objects that are stored in the cache memory,  then we need to call either evict() or clear()methods.   Actually evice() is used to remove a particular object from the cache memory and clear() is toremove all objects in the cache memory

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | Session ses = factory.openSession();  Object ob = ses1.get(Student.class, newInteger(101));  Student s = (Student)ob  System.out.println(s.getStudentId());    ses.evict(s);    Object ob1 = ses.get(Student.class, newInteger(101)); |

* Opened session at line number 1
* Loaded the object with id 101, so hibernate will lads this object from the database as this is thefirst time in the session
* At line number 4, i printed my data  bla bla..
* then in line number 6, i removed this object [ with id 101 ] from the cache memory of the session by calling evict() method
* Now in line number 8 again i tried to load the same object,  so as we are in the same session hibernate first will verify whether the object is there in the cache or not, if not loads the object from the database, but we removed the object from the cache with evict() method right, so hibernate will loads from the database
* I mean, first checks at local session then only from the database if its not available in the local cache

And that’s it mates :-) …………!!!!!!!!

# How To Enable Second Level Caching In Hibernate

First level cache will be enabled by default, but for enable second level cache we need to follow somesettings, let us see few points regarding this..

* Second level cache was introduced in hibernate 3.0
* When ever we are loading any object from the database,  then hibernate verify whether that object is available in the local cache memory of that particular session [**means first level cache** ], if not available then hibernate verify whether the object is available in global cache or factory cache [**second level cache** ], if not available then hibernate will hit the database and loads the object from there, and then first stores in the local cache of the session [ first level ] then in the global cache [ second level cache ]
* When another session need to load the same object from the database,  then hibernate copies that object from global cache [ second level cache ] into the local cache of this new session

Second level cache in the hibernate is of  from **4** vendors…

* Easy Hibernate [EHCache] Cache from hibernate framework
* Open Symphony [OS] cache from Open Symphony
* SwarmCache
* TreeCache from JBoss

## How to enable second level cache in hibernate

We need one provider class, hear we are going to see hibernate provider class that is EHCache

## Changes required

To enable second level cache in the hibernate, then the following **3** changes are required

* Add provider class in hibernate configuration file like…

|  |  |
| --- | --- |
| 1  2  3 | <property name="hibernate.cache.provider\_class">  org.hibernate.cache.EhCacheProvider  </property> |

* Configure cache elementfor a class in hibernate mapping file…

|  |  |
| --- | --- |
| 1 | <cacheusage="read-only"/> |

* **Note**: this must write soon after <class>
* create xml file called ehcache.xml and store in at class path location [ no confusions, i mean in the place where you have mapping and configuration XML's ] in web application.

## Important points on this second level cache

Lets take an example, we have 2 pojo classes in our application like Student, Employee.

* If we load student object from the database, then as its the first time hibernate will hits the database and fetch this student object data and stores in the session1 cache memory [ First level cache ], then in the global cache [ second level cache ] provided if we write <cache usage=”read-only” /> in the student mapping file
* I mean hibernate will stores in the local session memory by default, but it only stores in the global cache [ second level cache ] only if we write <cache usage=”read-only” /> in the student mapping file, if not so hibernate wont stores in the global cache
* Now take another session like session 2 for example, if session 2 also load the student object then hibernate will loads from the global cache [ second level cache ] as student object is available at global [Actually when ever we want to load any object hibernate first will checks at local, then global then database right hope you remembered this ], now if session 3 modify that student object then hibernate will thorows an error because we have written <cache usage=”read-only” /> in student mapping file
* We can avoid this by writing <cache usage=”read-write” />
* so remember <cache /> element has that much importance

Mates, still i have not explained about ehcache.xml did you observe :-)   i will explain about this by taking one example on this hibernate second level cache in the next session

## Regarding ehcache.xml

* In ehcache.xml, if eternal=”true” then we should not write timeToIdealSeconds,timeToLiveSeconds,  hibernate will take care about those values
* So if you want to give values manually better eternal=”false” always,  so that we can assignvalues into timeToIdealSeconds, timeToLiveSeconds manually, and play ;)
* timeToIdealSeconds=”seconds” means, if the object in the global chche is ideal, means not using by any other class or object then it will be waited for some time we specified and deleted from the global cache if time is exceeds more than timeToIdealSeconds value
* timeToLiveSeconds=”seconds” means, the other Session or class using this object or not, i mean may be it is using by other sessions or may not, what ever the situation might be, once it competed the time specified timeToLiveSeconds, then it will be removed from the global cache by hibernate
* Actually <defaultCache … /> will reflects to all the pojo classes in our application,  and we can also assign the ehcache values to specified pojo class by <cache name=”– your pojo class name—” …….. />

## Regarding ForOurLogic.java

* From line numbers 16 – 22 we opened session1 and loaded object and closed session1, this time object will be loaded from the database as its the first time
* Then from 27 – 31 we have been waited for 6 seconds,  but in our ehcache.xml we have giventimeToIdleSeconds=”5″ , i mean after 5 seconds object will be removed from the global cache
* And again in ForOurLogic.java line numbers 35 – 41 we opened second session and loaded the object, this time hibernate will loads the object from the database, and closed the session
* Immediately from 43 – 49 we opened session3 and loaded the object, this time hibernate will loads the object form the global session not from the database

## We have existing record in the database

|  |  |
| --- | --- |
| bg00000 | bg00000 |

## Output after run the ForOurLogic.java

|  |  |
| --- | --- |
| bg00000 | bg00000 |

And that’s it mates….!!!!!!!

[download-hibernate-source-code](http://www.java4s.com/wp-content/hibernate-source-code/SecondLevel_Cache_Example.rar)

|  |
| --- |
|  |

# Hibernate Annotations Introduction

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

Gun short point :-  Annotations are replacement for XML

Let us see few points regarding annotations in hibernate

* Annotations are introduced in java along with JDK 1.5, annotations are used to provide META data to the classes, variables, methods of java
* Annotations are given by SUN as replacement to the use of xml files in java
* In hibernate annotations are given to replace hibernate mapping [ xml ] files
* While working with annotations in hibernate, we do not require any mapping files, but hibernate xml configuration file is must
* hibernate borrowed annotations from java persistence API but hibernate it self doesn’t contain its own annotations
* Every annotations is internally an Interface, but the key words starts with @ symbol

## Like…..

|  |  |
| --- | --- |
| 1  2  3  4 | public@interfaceJava4s  {      // code.....  } |

* For working with annotations, our java version must be 1.5 or higher and hibernate version must be hibernate 3.3 or higher, actually up to now we have been used hibernate 2.2 in our previous tutorials, get ready to download the higher version :-)
* At j2se level, sun has provided very limited set of annotations like @Override and @Deprecated …etc…
* Sun has provided the annotations related to j2se level under java.lang.annotations.\* package
* Most of the annotations related to j2ee level are given by sun and their implementations are given by the vendors
* In hibernate, the annotations supported are given by sun under javax.persistence package and hibernate has provided implementations for annotations given in that package

The basic annotations we are using while creating hibernate pojo classes are…

* @Entity
* @Table
* @Id
* @Column

Actually @Entity, @Table are class level annotations, and @Id, @Column are the field level annotations, no worries you will be able to understand while seeing the first example :-)

* in hibernate, if we are annotations in the pojo class then hibernate mapping file is not required, it means annotations are reducing the use of xml files in the hibernate

Though we are using annotations in our pojo class with mapping xml also, then hibernate will give first preference to xml only not for annotations,  actually this concept is same in struts, hibernate, spring too

And that’s it friends,  now you are ready to work with annotations in hibernate ;)

# Jars Required For Hibernate Annotations

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

For working with annotations in hibernate,  ensure our java version must be 1.5 or higher and we must use hibernate version 3.3 or higher, and in fact no need to use all the jar files to work with these annotations.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48 | packagestr;    importjavax.persistence.Column;  importjavax.persistence.Entity;  importjavax.persistence.Id;  importjavax.persistence.Table;    @Entity  @Table(name = "student\_talbe")  publicclassProduct{        @Id      @Column(name="proid")      privateintproductId;        @Column(name="proName", length=10)      privateString proName;        @Column(name="price")      privatedoubleprice;        publicvoidsetProductId(intproductId)      {          this.productId = productId;      }      publicintgetProductId()      {          returnproductId;      }        publicvoidsetProName(String proName)      {          this.proName = proName;      }      publicString getProName()      {          returnproName;      }        publicvoidsetPrice(doubleprice)      {          this.price = price;      }      publicdoublegetPrice()      {          returnprice;      }  } |

## ClientForSave.java

|  |  |
| --- | --- |
|  | packagestr;    importorg.hibernate.Session;  importorg.hibernate.SessionFactory;  importorg.hibernate.Transaction;  importorg.hibernate.cfg.AnnotationConfiguration;    publicclassClientForSave {        publicstaticvoidmain(String[] args)      {            AnnotationConfiguration cfg=newAnnotationConfiguration();          cfg.configure("hibernate.cfg.xml");            SessionFactory factory = cfg.buildSessionFactory();          Session session = factory.openSession();          Product p=newProduct();            p.setProductId(105);          p.setProName("java4s");          p.setPrice(15000);            Transaction tx = session.beginTransaction();          session.save(p);          System.out.println("Object saved successfully using annotations.....!!");          tx.commit();          session.close();          factory.close();      }   } |

## Vendor.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | packagestr;    importjava.util.Set;    importjavax.persistence.CascadeType;  importjavax.persistence.Column;  importjavax.persistence.Entity;  importjavax.persistence.FetchType;  importjavax.persistence.Id;  importjavax.persistence.JoinColumn;  importjavax.persistence.OneToMany;  importjavax.persistence.Table;    @Entity  @Table(name = "Vendor")    publicclassVendor{        @Id      @Column(name = "vid")      privateintvendorId;        @Column(name = "vname", length=10)      privateString vendorName;        publicintgetVendorId() {          returnvendorId;      }        publicvoidsetVendorId(intvendorId) {          this.vendorId = vendorId;      }        publicString getVendorName() {          returnvendorName;      }        publicvoidsetVendorName(String vendorName) {          this.vendorName = vendorName;      }    } |

# Difference Between Merge And Update Methods In Hibernate

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

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

## Let Us Take An Example

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | ------  -----  SessionFactory factory = cfg.buildSessionFactory();  Session session1 = factory.openSession();    Student s1 = null;  Object o = session1.get(Student.class, newInteger(101));  s1 = (Student)o;  session1.close();    s1.setMarks(97);    Session session2 = factory.openSession();  Student s2 = null;  Object o1 = session2.get(Student.class, newInteger(101));  s2 = (Student)o1;  Transaction tx=session2.beginTransaction();    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
* Hear 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 we update and merge methods will come into picture when ever we loaded the same object again and again into the database, like above.

Folks i have been changed the values in the source code  
 [Difference-between-merge-and-update-methods](http://www.java4s.com/wp-content/hibernate-source-code/Merge_Update_Difference.rar)

|  |
| --- |
|  |

**Difference Between Hibernate Save And Persist Methods**

|  |  |
| --- | --- |
| [Hibernate](http://www.java4s.com/tutorials/hibernate/) » | Updated On Apr 07, 2012 |  |

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

* If our generator class is assigned, then there is no difference between save() and persist() methods. Because generator ‘assigned’ means, as  a programmer we need to give the primarykey value to save in the database right [ Hope you know this generators concept ]
* In case of other than assigned generator class, suppose if our generator class name is Increment means hibernate it self will assign the primary key id value into the database right [ other than assigned generator, hibernate only used to take care the primary key id value remember :-) ], so in this case if we call save() or persist() method then it will insert the record into the database normally

But hear thing is,  save() method can return that primary key id value which is generated by hibernate and we can see it by

long s = session.save(k);

In this same case, persist() will never give any value back to the client, hope you are clear.

**15. What’s the difference between session.save() and session.saveOrUpdate() methods in hibernate?**  
Sessionsave() method saves a record only if it’s unique with respect to its primary key and will fail to insert if primary key already exists in the table.  
saveOrUpdate() method inserts a new record if primary key is unique and will update an existing record if primary key exists in the table already.