**Hi Friends**,

**Just go through the following small story..** ( taken from **"*You Can Win* "** - by Shiv Khera )

Once upon a time a very strong woodcutter asks for a job in a timber merchant, and he got it. The paid was really good and so were the work conditions. For that reason, the woodcutter was determined to do his best. His boss gave him an axe and showed him the area where he was supposed to work. The first day, the woodcutter brought 18 trees "Congratulations," the boss said. "Go on that way!" Very motivated for the boss' words, the woodcutter try harder the next day, but he only could bring 15 trees. The third day he try even harder, but he only could bring 10 trees. Day after day he was bringing less and less trees. The woodcutter thought that "I must be losing my strength". He went to the boss and apologized, saying that he could not understand what was going on.

The boss asked, "When was the last time you sharpened your axe?"

"Sharpen? I had no time to sharpen my axe. I have been very busy trying to cut trees.

*If we are just busy in applying for jobs & work, when we will sharpen our skills to chase the job selection process?*

*My aim is to provide good and quality content to readers to understand easily. All contents are written and practically tested by me before publishing. If you have any query or questions regarding any article feel free to leave a comment or you can get in touch with me on venus.kumaar@gmail.com.*

*This is just a start, I have achieved till now is just 0.000001n% .*

With Warm Regards

**Venu Kumaar.S**
venus.kumaar@gmail.com

# Hibernate

**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 theform 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.

**Hibernate**

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 wont forces the programmers to extend/implement any class/interface, and in hibernate we have all POJO classes so its light weight.
Hibernate can runs with in or with out 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**

Mapping file is the heart of hibernate application.
•Every ORM tool needs this mapping, mapping is the mechanism of placing an object properties into column's of a table.
•Mapping can be given to an ORM tool either 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 2ways,

•XML

•Annotations.

Actually annotations are introduced into java from JDK 1.5.

**Mapping.xml**

```
<hibernate-mapping>

<class name="POJO class name" table="table name in database">
<id name="variable name" column="column name in database" type="java/hibernate type" />
<property name="variable1 name" column="column name in database" type="java/hibernate type"
/>
<property name="variable2 name" column="column name in database" type="java/hibernate type"
/>
</class>

</hibernate-mapping>
```

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>
<property name="connection.user">user </property>
<property name="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>
<property name="hbm2ddl.auto">create/update or what ever</property>
<!-- Related to hibernate properties END-->

<!-- Related to mapping START-->
<mapping resource="hbm file 1 name .xml" / >
<mapping resource="hbm file 2 name .xml" / >
<!-- Related to the mapping END -->

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

But XML files are always recommended to work.

Advantages & Disadvantages:

**Advantages**

**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
•But there is one major disadvantage, which was boilerplate code issue, actually we need to writesame code in several files in the same application, but spring eliminated this

Steps to use Hibernate:

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 inorg.hibernate.cfg.*; and we need to call configure() 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 isstarted 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

```
likeSessionFactory 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 donot 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 = session.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

**Simple Program:**

an hibernate program..

•Product.java (My POJO class)
•Product.hbm.xml (Xml mapping file )
•hibernate.cfg.xml (Xml configuration file)
•ClientForSave.java (java file to write our hibernate logic)

**Product.java**

```
private int productId;

private String proName;
private double price;

public void setProductId(int productId)
```

```
{
   this.productId = productId;
}
public int getProductId()
{
   return productId;
}

public void setProName(String proName)
{
   this.proName = proName;
}
public String getProName()
{
   return proName;
}

public void setPrice(double price)
{
   this.price = price;
}
public double getPrice()
{
   return price;
}
```

**product.hbm.xml:**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="Product" table="PRODUCTS">

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

<property name="proName" column="pname" />
<property name="price"/>
```

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

In this mapping file, my Product class is linked with PRODUCTS table in the database, and next is the id element, means in the database table what column we need to take as primary key column, that property name we need to give hear, actually i have been given my property name productId which will mapped with pid column in the table.
And proName is mapped with pname column of the PRODUCTS table, see i have not specified anycolumn for the property price, this means that, our property name in the pojo class and the column name in the table both are same.

Remember: the first 3 lines is the DTD for the mapping file, as a programmer no need to remember but we need to be very careful while you are copying this DTD, program may not be executed if you write DTD wrong, actually we have separate DTD's for Mapping xml and Configuration xml files.

**hibernate.cfg.xml:**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>

<!-- Related to the connection START -->
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.user">user</property>
<property name="connection.password">password</property>
<!-- Related to the connection END -->

<!-- Related to hibernate properties START -->
<property name="show_sql">true </property>
<property name="dialet">org.hibernate.dialect.OracleDialect </property>
<property name="hbm2ddl.auto">update </property>
<!-- Related to hibernate properties END -->

<!-- Related to mapping START -->
<mapping resource="product.hbm.xml" />
```

```
<!-- Related to the mapping END -->

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

In this configuration file i have been given my Oracle database connection properties, if you are using MySql then just specify your database related details actually its depends on you.

**ClientForSave.java**

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

public class ClientForSave {

   public static void main(String[] args)
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();
      Product p=new Product();

      p.setProductId(101);
      p.setProName("iPhone");
      p.setPrice(25000);

      Transaction tx = session.beginTransaction();
      session.save(p);
      System.out.println("Object saved successfully.....!!");
      tx.commit();
      session.close();
      factory.close();
   }

}
```

Now compile all .java files and run ClientForSave.java and check the output

Note:

•Make sure all .class, .java, .xml files are exist in the same folder
•Before you compile and run this application, ensure you set the class path for all 12 jars files, this is tedious like what i told you earlier, we can avoid this process from the next example with Eclipse, a real time tool
•except select operation, all other operations must be in the Transaction Scope

**Select Query Example**

**Previous Product.java**

**Product.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Product" table="products">

<id name="productId" column="pid" />
<property name="proName" column="pname" length="10"/>
<property name="price"/>

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

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
```

```
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**ClientProgram.java**

```
package str;


import org.hibernate.*;
import org.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();
      Object o=session.load(Product.class,new Integer(101));
      Product s=(Product)o;
      // For loading Transaction scope is not necessary...
      System.out.println("Loaded object product name is___"+s.getProName());

      System.out.println("Object Loaded successfully.....!!");
      session.close();
      factory.close();
   }

}
```

Note:

•In this program Product.java is just pojo class nothing special
•Mapping and Configuration files are just like previous programs
•But in ClientProgram.java, see in line number 16 load(-,-) method which is in the session, actually we have 2 methods to load the object from the database, they are load and get i will explain when time comes, as of now just remember this point
•Now see line number 19, we are going to print the product name by writing +s.getProName
•Actually once we loaded the object for the database with load or get methods the object data will be loads into the Product.java(POJO) setter methods, so we are printing by using getter methods

**Delete Query Program:**

**Product.java same as above**

**Product.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Product" table="products">

<id name="productId" column="pid" />
<property name="proName" column="pname" length="10"/>
<property name="price"/>

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

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

```
<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**ClientProgram.java**

```
package str;


import org.hibernate.*;
import org.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();

      Object o=session.load(Product.class,new Integer(103));
      Product p=(Product)o;

      Transaction tx = session.beginTransaction();
      session.delete(p);
      System.out.println("Object Deleted successfully.....!!");
      tx.commit();
```

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

}
```

Note:

•To deleting the object( 1 row) form the database we need to call delete method in the session.
•In the hibernate we have only one method to delete an object from the database that is what i have shown you hear..

**Update Example**

**Approach 1**

Load that object from the database, and modify its values, now hibernate automatically modifies the values on to database also, when ever the transaction is committed.

**Approach 2:**

If we want to modify object in the database, then create new object with same id and we must callupdate() given by session interface.

Files required to execute this program..

•Product.java (My POJO class)
•product.hbm.xml (Xml mapping file )
•hibernate.cfg.xml (Xml configuration file)
•ClientProgram.java(java file to write our hibernate logic)

**Related to approach 1:**

**Product.java, Product.hbm.xml same as above**

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**ClientProgram.java**

```
package str;


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

public class ForOurLogic {

   public static void main(String[] args)
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();
      Object o=session.load(Product.class,new Integer(105));
      Product s=(Product)o;

      Transaction tx = session.beginTransaction();

//s.setStno(105);   should not update, because we loaded with that number right..?
      s.setPrice(4000);   // implicitly update method will be called..
```

```
        tx.commit();

        System.out.println("Object Updated successfully.....!!");
        session.close();
        factory.close();
    }

}
```

Notes:

•See line number 20, actually there i tried to update Stno(105), we should not do this, because we have loaded the object from the database with his id number only, see line number 16, if we update hibernate will rises the exception
•See line number 24 once we call the commit(), automatically update method will be called by hibernate.
•When ever an object is loaded from the database then hibernate stores the loaded object incache-memory maintained by session-interface
•Once an object is loaded, if we do any modifications on that object by calling its setter methods, then these modification are stored in the object maintained by cache-memory
•if we modify the loaded object for multiple times then also the modifications will be stored in object maintained by the cache-memory only.
•when ever we issue commit() operation then hibernate verify whether any changes are there between the object stored in the cache and object in the database, if changes exists then hibernate automatically updates the database by generating any update operation.
•What am saying is hibernate automatically maintains synchronization between cache-memoryobject and database table objects (rows)

**Related to approach 2:**

```
package str;

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

public class ForOurLogic {
```

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

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

        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();

        Product p=new Product();
        p.setProductId(104);  // 104 must be in the DB
        p.setProName("Someting");

        Transaction tx = session.beginTransaction();
            session.update(p);
        tx.commit();

        System.out.println("Object Updated successfully.....!!");
        session.close();
        factory.close();
    }

}
```

Notes:

•From line number 17 to 19, we created new object and modified, and in line number 22 we are calling update method explicitly
•Hear we no need to load an object from the database
•we will create a new object, and we will assign same id no's to it and we will call update() explicitly in order to make the changes on the object that is stored in the database
That's it, actually first approach is recommended always..

## 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 when ever 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**

```
package str;

public class Product{

   private int productId;
   private String proName;
   private double price;

      private int v;

   public void setProductId(int productId)
   {
      this.productId = productId;
   }
   public int getProductId()
   {
      return productId;
   }

   public void setProName(String proName)
   {
      this.proName = proName;
   }
   public String getProName()
   {
      return proName;
```

```
    }

    public void setPrice(double price)
    {
       this.price = price;
    }
    public double getPrice()
    {
       return price;
    }

      public void setV(int v)
    {
       this.v = v;
    }
    public int getV()
    {
       return v;
    }
}
```

**Product.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Product" table="products">

<id name="productId" column="pid"  />
<version name="v" column="ver" />
<property name="proName" column="pname" length="10"/>
<property name="price"/>

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

Note:

•In this above mapping file, find the <version> element, there i have given column name as version, actually you can write any name its not predefined.

**Hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**clientForSave_1.java**

```
package str;


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

public class ClientForSave_1 {

   public static void main(String[] args)
   {

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

```
        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();
        Product p=new Product();

        p.setProductId(104);
        p.setProName("AC");
        p.setPrice(14000);

        Transaction tx = session.beginTransaction();
        session.save(p);
        System.out.println("Object saved successfully.....!!");
        tx.commit();
        session.close();
        factory.close();
    }

}
```

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.

**ClientForSave_2.java**

```
package str;


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

public class ClientForUpdate_2 {

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

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

        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();
        Object o=session.load(Product.class,new Integer(104));
        Product s=(Product)o;

        Transaction tx = session.beginTransaction();

         s.setPrice(4000); // implicitly update method will be call

        tx.commit();

        System.out.println("Object Updated successfully.....!!");
        session.close();
        factory.close();
    }

}
```

First run the **ClientForSave_1.java**, then only ClientForUpdate_2.java

**Note**(**Important..!!!**)

**A**ctually we can run any logic (Save or Update) for the first time, but make sure the versioning column is a number (>=0), but save logic has ability to insert zero by default if there is no value, and update logic will directly tries to increments already existing value by 1, it wont insert any value by default if its null, hope you are clear about this point.

Hibernate Lifecycle Of pojo Class Objects

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 inpersistent 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

Ex: ClientProgram.java

```
import org.hibernate.*;
import org.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=new Product();
      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....

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

**Transient:**

One newly created object,with out 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

Hibernate Converting Object From Detached to Persistent state

how to convert the detached state object into Persistent state again…,
As usual hibernate configuration, mapping XML are same and pojo class too, if you want just refer Hello World Program

**ClientLogicProgram.java:**

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

public class ClientLogicProgram {

   public static void main(String... args)
   {

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

      SessionFactory factory = cfg.buildSessionFactory();

      Session session1 = factory.openSession();

       Product p=null;          //Transient state..
       Object o=session1.get(Product.class, new Integer(1001));
       p=(Product)o;          //now p is in Persistent state..

      session1.close();
```

```
        p.setPrice(36000);          // p is in Detached state

        Session session2=factory.openSession();

         Transaction tx=session2.beginTransaction();
           session2.update(p);      // now p reached to Persistent state
         tx.commit();

        session2.close();

        factory.close();
}
}
```

Notes:

•We have opened the session1 at line number 14 and closed at line number 20, see i have been loaded the Product class object by using get(-,-) method
•Remember get() method always returns the super class object (Object)
•so i typecast into my pojo class object type, so now we can use print any value from this object so its in the Persistent state
•see line number 22, am trying to change the Price, but it wont effect the database because its not in the session cache so i need to take one more session to update this value in the database, so for that reason i took one more session from line numbers 24 – 30
•Gun short point is, things related to database must go with in the session only that's it

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).

**For Example:**

```
class Payment
{
  // content will goes hear
}
```

```
class CreditCard extends Payment
{
  // content will goes hear
}
```

See if you save CreditCard class object, then payment class object will also be saved into the database

Hibernate supports 3 types of Inheritance Mappings:

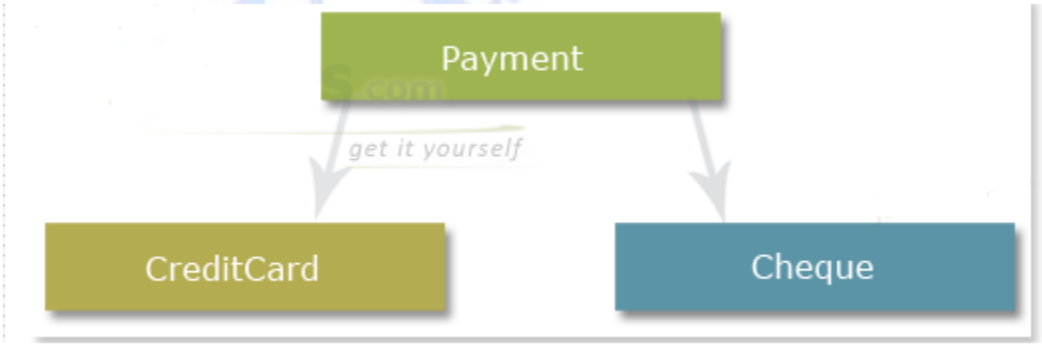
- Table per class hierarchy
- Table per sub-class hierarchy
- Table per concrete class hierarchy

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

**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 tableonly, which is base class table for sure.

But hear we must use one extra discriminator column in the database, just to identify which derivedclass 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.

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.java:

```
package str;


public class Payment{

    private int paymentId;
    private double amount;

    public int getPaymentId() {
        return paymentId;
    }
    public void setPaymentId(int paymentId) {
        this.paymentId = paymentId;
    }
    public double getAmount() {
        return amount;
    }
    public void setAmount(double amount) {
        this.amount = amount;
    }

}
```

**CreditCard.java:**

```
package str;

public class CreditCard extends Payment{
```

```
    private String CreditCardType;

    public String getCreditCardType() {
        return CreditCardType;
    }

    public void setCreditCardType(String creditCardType) {
        CreditCardType = creditCardType;
    }

}
```

**Cheque.java:**

```
package str;


public class Cheque extends Payment{

    private String ChequeType;

    public String getChequeType() {
        return ChequeType;
    }

    public void setChequeType(String chequeType) {
        ChequeType = chequeType;
    }

}
```

**ClientForSave.java**

```
package str;


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

public class ClientForSave {

    public static void main(String[] args)
    {

        Configuration cfg = new Configuration();
```

```
        cfg.configure("hibernate.cfg.xml");

        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();

        CreditCard c=new CreditCard();

        c.setPaymentId(10);
        c.setAmount(2500);
        c.setCreditCardType("Visa");

        Cheque c1=new Cheque();

        c1.setPaymentId(11);
        c1.setAmount(2600);
        c1.setChequeType("ICICI");

        Transaction tx = session.beginTransaction();
        session.save(c);
        session.save(c1);
        System.out.println("Object saved successfully.....!!");
        tx.commit();
        session.close();
        factory.close();
    }

}
```

**Payment.hbm.xml:**

```
<?xml version="1.0"?>

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

<hibernate-mapping>

<class name="str.Payment" table="PAYMENT">
<id name="paymentId" column="pid" />
<discriminator column="dcolumn" type="string" length="5"/>
<property name="amount" column="amt" />

<subclass name="str.CreditCard" discriminator-value="CC">
<property name="CreditCardType" column="cctype" length="10" />
```

```
</subclass>

<subclass name="str.Cheque" discriminator-value="cq">
<property name="ChequeType" column="cqtype" length="10" />
</subclass>

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

**hibernate.cfg.xml:**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Payment.hbm.xml" />

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

Notes:

•Payment.java, CreditCard.java, Cheque.java are just pojo classes nothing to explain, but see in CreditCard.java, Cheque.java i have inherited the Payment.java.

•In this inheritance concept, mapping file is the central part, see in line number 10, we added one new line discriminator, after the id element just to identify which derived class object we have been saved in the table (see the oracle console once)

•every thing has been saved in a single table

## Hibernate Inheritance: Table Per subClass Hierarchy

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.java:**

```
package str;

public class Payment{

   private int paymentId;
   private double amount;

   public int getPaymentId() {
      return paymentId;
   }
   public void setPaymentId(int paymentId) {
      this.paymentId = paymentId;
   }
   public double getAmount() {
      return amount;
```

```
    }
    public void setAmount(double amount) {
        this.amount = amount;
    }

}
```

**CreditCard.java:**

```
package str;


public class CreditCard extends Payment{

    private String CreditCardType;

    public String getCreditCardType() {
        return CreditCardType;
    }

    public void setCreditCardType(String creditCardType) {
        CreditCardType = creditCardType;
    }

}
```

**Cheque.java:**

```
package str;


public class Cheque extends Payment{

    private String ChequeType;

    public String getChequeType() {
        return ChequeType;
    }

    public void setChequeType(String chequeType) {
        ChequeType = chequeType;
    }

}
```

**ClientForSave.java**

```
package str;


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

public class ClientForSave {

   public static void main(String[] args)
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();

      CreditCard c=new CreditCard();

      c.setPaymentId(10);
      c.setAmount(2500);
      c.setCreditCardType("Visa");



      Cheque c1=new Cheque();

      c1.setPaymentId(11);
      c1.setAmount(2600);
      c1.setChequeType("ICICI");



      Transaction tx = session.beginTransaction();
      session.save(c);
      session.save(c1);
      System.out.println("Object saved successfully.....!!");
      tx.commit();
      session.close();
      factory.close();
   }
```

```
}
```

**Payment.hbm.xml:**

```
<?xml version="1.0"?>

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

<hibernate-mapping>

<class name="str.Payment" table="PAYMENT">

<id name="paymentId" column="pid" />
<property name="amount" column="amt" />

<joined-subclass name="str.CreditCard" table="CreditCard">
<key column="dummy1" />
<property name="CreditCardType" column="cctype" length="10" />
</joined-subclass>

<joined-subclass name="str.Cheque" table="Cheque">
<key column="dummy2" />
<property name="ChequeType" column="cqtype" length="10" />
</joined-subclass>

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

**hibernate.cfg.xml:**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>
```

```
<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Payment.hbm.xml" />

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

Notes:

•In the mapping file, <key –> element is because, once we save the derived class object, then hibernate will first save the baseclass object then derived class object right ..!, so at the time of saving the derived class object hibernate will copy the primary key value of the base class into the corresponding derived class, see in the above output 10 copied into dummy1 column of CreditCardtable and 11 copied into Dummy2 column of the cheque table

## Hibernate Inheritance: Table Per Concrete Class Hierarchy

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.java:**

```
package str;

public class Payment{
```

```
    private int paymentId;
    private double amount;

    public int getPaymentId() {
        return paymentId;
    }
    public void setPaymentId(int paymentId) {
        this.paymentId = paymentId;
    }
    public double getAmount() {
        return amount;
    }
    public void setAmount(double amount) {
        this.amount = amount;
    }

}
```

**CreditCard.java:**

```
package str;


public class CreditCard extends Payment{

    private String CreditCardType;

    public String getCreditCardType() {
        return CreditCardType;
    }

    public void setCreditCardType(String creditCardType) {
        CreditCardType = creditCardType;
    }

}
```

**Cheque.java:**

```
package str;

public class Cheque extends Payment{

    private String ChequeType;
```

```
    public String getChequeType() {
        return ChequeType;
    }

    public void setChequeType(String chequeType) {
        ChequeType = chequeType;
    }

}
```

**ClientForSave.java**

```
package str;


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

public class ClientForSave {

    public static void main(String[] args)
    {

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

        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();

        CreditCard c=new CreditCard();

        c.setPaymentId(10);
        c.setAmount(2500);
        c.setCreditCardType("Visa");



        Cheque c1=new Cheque();

        c1.setPaymentId(11);
        c1.setAmount(2600);
        c1.setChequeType("ICICI");
```

```
        Transaction tx = session.beginTransaction();
        session.save(c);
        session.save(c1);
        System.out.println("Object saved successfully.....!!");
        tx.commit();
        session.close();
        factory.close();
    }

}
```

**Payment.hbm.xml:**

```
<?xml version="1.0"?>

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

<hibernate-mapping>

<class name="str.Payment" table="PAYMENT">

<id name="paymentId" column="pid" />
<property name="amount" column="amt" />

<union-subclass name="str.CreditCard">
<property name="CreditCardType" column="cctype" length="10" />
</union-subclass>

<union-subclass name="str.Cheque">
<property name="ChequeType" column="cqtype" length="10" />
</union-subclass>

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

**hibernate.cfg.xml:**

```
<?xml version='1.0' encoding='UTF-8'?>

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

```
<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Payment.hbm.xml" />

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

## Example On Composite Primary Keys In Hibernate

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>

### Example On this__

Files required….

•Product.java (Pojo)
•ForOurLogic.java (for our logic)
•hibernate.cfg.xml
•Product.hbm.xml

### Product.java

```
package str;

public class Product implements java.io.Serializable{
```

```
   private static final long serialVersionUID = 1L;

   private int productId;
   private String proName;
   private double price;

   public void setProductId(int productId)
   {
      this.productId = productId;
   }
   public int getProductId()
   {
      return productId;
   }

   public void setProName(String proName)
   {
      this.proName = proName;
   }
   public String getProName()
   {
      return proName;
   }

   public void setPrice(double price)
   {
      this.price = price;
   }
   public double getPrice()
   {
      return price;
   }
}
```

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

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

```
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**Product.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Product" table="products">

<composite-id>
<key-property name="productId" column="pid"  />
<key-property name="proName" column="pname" length="10" />
</composite-id>

<property name="price"/>

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

**ForOurLogic.java**

```
package str;


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

public class ForOurLogic {

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

```
    {

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

        SessionFactory factory = cfg.buildSessionFactory();
        Session session = factory.openSession();

        Product p=new Product();

        p.setProductId(101);
              p.setProName("iPhone");
              p.setPrice(25000);

           Transaction tx=session.beginTransaction();
             session.save(p);
             System.out.println("Object Loaded successfully.....!!");
           tx.commit();

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

}
```

Notes:

•see Product.java pojo class, in line number 3 i have implemented the java.io.Serializableinterface, 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)

**Ex2:**

Files required….

•Product.java (Pojo)
•ForOurLogic4Load.java (for our logic)
•hibernate.cfg.xml
•Product.hbm.xml

All files are same like previous program, but ForOurLogic4Load.java is the new file for loading an object from the database

```
package str;

public class Product implements java.io.Serializable{

   private static final long serialVersionUID = 1L;

   private int productId;
   private String proName;
   private double price;

   public void setProductId(int productId)
   {
      this.productId = productId;
   }
   public int getProductId()
   {
      return productId;
   }

   public void setProName(String proName)
   {
      this.proName = proName;
   }
   public String getProName()
   {
      return proName;
   }

   public void setPrice(double price)
   {
      this.price = price;
   }
   public double getPrice()
```

```
   {
      return price;
   }
}
```

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**Product.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Product" table="products">

<composite-id>
<key-property name="productId" column="pid"  />
<key-property name="price" />
</composite-id>
```

```
<property name="proName" column="pname" length="10"/>

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

**ForOurLogic4Load.java**

```
package str;


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

public class ForOurLogic4Load {

   public static void main(String[] args)
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();

      Product p=new Product();

      p.setProductId(101);
      p.setPrice(25000);

      Object o=session.get(Product.class, p);
      // hear p must be an serializable object,

      Product p1=(Product)o;

      System.out.println("The price is: "+p1.getProName());

      System.out.println("Object Loaded successfully.....!!");
      session.close();
      factory.close();
   }

}
```

Notes:

•regarding ForOurLogic4Load.java actually for loading an object we have to pass the primary key column value like…

```
1 Object o=session.get(Product.class, new
  Integer(101));
```

•Actually that is the case with single primary key, but see in this example we are using multiple primary keys (2 primary key values), so while loading an object how we will give two values….?

```
1 Object o=session.get(Product.class, new
  Integer(101,25000));
```

• which is absolutely wrong……!!!!!!!!!

•so how we can pass the multiple primary key values for loading that object.. ?

•First we need to set the values just like in line numbers 19,20 , ensure you are setting the values which already have in the database for that particular object.

•Then we need to pass that object as the parameter to load the object.

```
1 Object
  o=session.get(Product.class, p);
```

•What am saying is that the second parameter of the get method is always a Serializable object, so in the Product.java line number 3 i just implemented withjava.io.Serializable

•Remember, if we have single primary key we used to give

```
1 Object o=session.get(Product.class, new
  Integer(101));
```

hear new Integer(101) is the wrapper, all wrappers are Serializable by default.

## Generators <generator> In Hibernate

<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 theresponse

•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 implementedfrom

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

**Example**

```
<generator class="">
<param name=""> value </param>
</generator>
```

**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)

**assigned**

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

```
<id name="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

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

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

If the programmer has not passed any sequence name, then hibernate creates its own sequence with name "Hibernate-Sequence" and gets next value from that sequence, and 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

```
<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

**hilo**

This generator is database independent
•for the first record, the id value will be inserted as 1
•for the second record the id value will be inserted as 32768
•for the next records the id value will be incremented by 32768 and will stores into the database (i mean adds to the previous)
•actually this hibernate stores the count of id values generated in a column of separated table, with name "hibernate_unique_key" by default with the column name "next_hi"
•if we want to modify the table and column names theen wee need to pass 2 parameter's for the hilo

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

<generator>

<param name="table">your table name</param>
<param name="column">your column name </param>

</generator>

</id>
**native**

when we use this generator class, it first checks whether the database supports identify or not, if not checks for sequence and if not, then hilo will be used finally the order will be..

- identify
- sequence
- hilo

For example, if we are connecting with oracle, if we use generator class as native then it is equal to the generator class sequence.

**foreign**

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

## Hibernate In Servlet Example, Hibernate In Servlet Tutorial

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

## Example On Hibernate Pagination With Servlet In Eclipse

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 the next 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 Pagination Example In Eclipse**

Mates, let see one real time example on this hibernate pagination with servlet
files required…
•Pagination.java
•Product.java
•Product.hbm.xml
•hibernate.cfg.xml
•web.xml
•

Note:
•src folder contains all *.java files
•build folder contains all *.class files in side classes folder
•Hibernate related xml's hibernate.cfg.xml, mapping files should be in classes folder only

**product.java**

```
public class Product{
```

```
   private int productId;
   private String proName;
   private double price;

   public void setProductId(int productId)
   {
      this.productId = productId;
   }
   public int getProductId()
   {
      return productId;
   }

   public void setProName(String proName)
   {
      this.proName = proName;
   }
   public String getProName()
   {
      return proName;
   }

   public void setPrice(double price)
   {
      this.price = price;
   }
   public double getPrice()
   {
      return price;
   }
}
```

**Product.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="com.java4s.hservlet.pagination.Product" table="products">

<id name="productId" column="pid"  />
<property name="proName" column="pname" length="10"/>
```

```
<property name="price"/>

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

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**Pagination.java**

```
import java.io.IOException;

import java.io.PrintWriter;
import java.util.Iterator;
import java.util.List;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServlet;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
```

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

public class Pagination extends HttpServlet
{

SessionFactory factory;

//init method started
public void init(ServletConfig config)throws ServletException
{
factory = new Configuration().configure().buildSessionFactory();
System.out.println("Factory has been created....");
}
//init method end

//service method start
public void service(ServletRequest req, ServletResponse res)
throws ServletException,IOException
{
int pageIndex = 0;
int totalNumberOfRecords = 0;
int numberOfRecordsPerPage = 4;

String sPageIndex = req.getParameter("pageIndex");

if(sPageIndex ==null)
{
pageIndex = 1;
}else
{
pageIndex = Integer.parseInt(sPageIndex);
}

Session ses = factory.openSession();
int s = (pageIndex*numberOfRecordsPerPage) -numberOfRecordsPerPage;

Criteria crit = ses.createCriteria(Product.class);
crit.setFirstResult(s);
crit.setMaxResults(numberOfRecordsPerPage);

List l = crit.list();
Iterator it = l.iterator();
```

```
PrintWriter pw = res.getWriter();
pw.println("<table border=1>");
pw.println("<tr>");
pw.println("<th>PID</th><th>PNAME</th><th>PRICE</th>");
pw.println("</tr>");

while(it.hasNext())
{
Product p = (Product)it.next();
pw.println("<tr>");
pw.println("<td>"+p.getProductId()+"</td>");
pw.println("<td>"+p.getProName()+"</td>");
pw.println("<td>"+p.getPrice()+"</td>");
pw.println("</tr>");
}

pw.println("<table>");

Criteria crit1 = ses.createCriteria(Product.class);
crit1.setProjection(Projections.rowCount());

List l1=crit1.list();

// pw.println(l1.size());
//returns 1, as list() is used to execute the query if true will returns 1

Iterator it1 = l1.iterator();

if(it1.hasNext())
{
Object o=it1.next();
totalNumberOfRecords = Integer.parseInt(o.toString());
}

int noOfPages = totalNumberOfRecords/numberOfRecordsPerPage;
if(totalNumberOfRecords > (noOfPages * numberOfRecordsPerPage))
{
noOfPages = noOfPages + 1;
}

for(int i=1;i<=noOfPages;i++)
{
String myurl = "ind?pageIndex="+i;
pw.println("<a href="+myurl+">"+i+"</a>");
```

```
}

ses.close();
pw.close();

}
//service method end

//destroy method start
public void destroy()
{
factory.close();
}
//destroy end
}
```

Note:

•We must create the SessionFactory object in the init() only, as it is the heavy weight one
•and nothing to explain, just read slowly. If you got struck at any point just fire a question in our forum, and see line number 101, ind?pageIndex (ind is my url pattern)

**web.xml**

```
<web-app>

<servlet>
<servlet-name>dummyName</servlet-name>
<servlet-class>com.java4s.hservlet.pagination.Pagination</servlet-class>
</servlet>

<servlet-mapping>
<servlet-name>dummyName</servlet-name>
<url-pattern>/ind</url-pattern>
</servlet-mapping>
</web-app>
```

## Hibernate One to Many Mapping Insert Query Example

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 either Set,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 Insert Query**

files required…

•Vendor.java [pojo class]
•Customer.java [pojo class]
•OurLogic.java
•Customer.hbm.xml
•hibernate.cfg.xml
•Vendor.hbm.xml

**Vendor.java**

```
package str;


import java.util.Set;

public class Vendor {

private int vendorId;
private String vendorName;
private Set children;

public int getVendorId() {
return vendorId;
}
public void setVendorId(int vendorId) {
this.vendorId = vendorId;
}
public String getVendorName() {
return vendorName;
}
public void setVendorName(String vendorName) {
this.vendorName = vendorName;
}
public Set getChildren() {
```

```
return children;
}
public void setChildren(Set children) {
this.children = children;
}

}
```

**Customer.java**

```
package str;

public class Customer {

private int customerId;
private String customerName;
private int forevenId;

public int getCustomerId() {
return customerId;
}
public void setCustomerId(int customerId) {
this.customerId = customerId;
}
public String getCustomerName() {
return customerName;
}
public void setCustomerName(String customerName) {
this.customerName = customerName;
}

public int getForevenId() {
return forevenId;
}
public void setForevenId(int forevenId) {
this.forevenId = forevenId;
}

}
```

**Customer.hbm.xml**

```
<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
```

```
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>
<class name="str.Customer" table="customer">

<id name="customerId" column="custid"  />
<property name="customerName" column="custname" length="10"/>

<property name="forevenId" column="forevenid"  insert="false" />

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

**Vendor.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Vendor" table="vendor">

<id name="vendorId" column="vendid"  />
<property name="vendorName" column="vendname" length="10"/>

<set name="children" cascade="all" >

<key column="forevenid" />
<one-to-many class="str.Customer" />

</set>

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

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
```

```
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Customer.hbm.xml"></mapping>
<mapping resource="Vendor.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**OurLogic.java**

```
package str;


import java.util.HashSet;
import java.util.Set;

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

public class OurLogic {

public static void main(String args[])
{

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

SessionFactory factory = cfg.buildSessionFactory();
Session session = factory.openSession();
//parent object
Vendor v =new Vendor();

v.setVendorId(101);
v.setVendorName("java4s");
```

```
//creating 3 child objects
Customer c1=new Customer();

c1.setCustomerId(504);
c1.setCustomerName("customer4");

Customer c2=new Customer();

c2.setCustomerId(505);
c2.setCustomerName("customer5");

Customer c3=new Customer();

c3.setCustomerId(506);
c3.setCustomerName("customer6");

// adding child objects to set, as we taken 3rd property set in parent
Set s=new HashSet();

s.add(c1);
s.add(c2);
s.add(c3);

v.setChildren(s);

Transaction tx = session.beginTransaction();

session.save(v);

tx.commit();
session.close();
System.out.println("One To Many is Done..!!");
factory.close();

}
}
```

Notes:

- In Vendor.java, we have taken a property of type Set
- In Customer.java, we have taken one property forevenId of type int to insert Vendor id

**Regarding Vendor.hbm.xml**

In this mapping file, we used a collection configuration element (Set), because in pojo class of vendor,

we used the collection type as Set, (we can use Map,List too)

•In order to transfer 0perations on parent object to child object we need to add cascade attribute
•By default, cascade value is none, it means even though relationship is exist, the operations we are doing on parent will not transfer to child, i mean to say hear operations are insert, delete, update
•In our Vendor.hbm.xml, we used cascade ="all" means all operations at parent object will be transfer to child
•while applying relationships, we need to configure the foreign key column name, by using which the relationship is done
•Inthe mapping file, we need to use <key /> eliment to configure foregin key column name, in this example forevenid is foreign key
•<one-to-many> is child class with which relation been done, in our example str.Customer is the class [str is the package]

**Hibernate Many to One Mapping Insert Query Example**

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="">

files required...

•Vendor [parent pojo]
•Customer.java [child pojo]
•OurLogic.java [our logic]
•Vendor.hbm.xml
•Customer.hbm.xml
•hibernate.cfg.xml

**Vendor.java**

```
package str;


public class Vendor {

    private int vendorId;
    private String vendorName;

    public int getVendorId() {
        return vendorId;
    }
    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }
    public String getVendorName() {
        return vendorName;
    }
    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
    }

}
```

**Customer.java**

```
package str;

public class Customer {

    private int customerId;
    private String customerName;
    private Vendor parentObjects;

    public Vendor getParentObjects() {
        return parentObjects;
    }
    public void setParentObjects(Vendor parentObjects) {
        this.parentObjects = parentObjects;
    }
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
```

```
        this.customerId = customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
}
```

**Vendor.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Vendor" table="vendor">

<id name="vendorId" column="vendid"  />
<property name="vendorName" column="vendname" length="10"/>

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

**Customer.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Customer" table="customer">

<id name="customerId" column="custid"  />
<property name="customerName" column="custname" length="10"/>
<many-to-one name="parentObjects" column="Vdummy" class="str.Vendor" cascade="all" />

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

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Customer.hbm.xml"></mapping>
<mapping resource="Vendor.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

```
package str;


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

public class OurLogic {

   public static void main(String args[])
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();

      Vendor v =new Vendor();
```

```
                v.setVendorId(101);
                v.setVendorName("java4s");

        Customer c1=new Customer();

                c1.setCustomerId(504);
                c1.setCustomerName("customer4");
                c1.setParentObjects(v);

        Customer c2=new Customer();

                c2.setCustomerId(505);
                c2.setCustomerName("customer5");
                c2.setParentObjects(v);

        Customer c3=new Customer();

                c3.setCustomerId(506);
                c3.setCustomerName("customer6");
                c3.setParentObjects(v);

          Transaction tx = session.beginTransaction();

                        session.save(c1);
                        session.save(c2);
                        session.save(c3);

          tx.commit();
          session.close();
          System.out.println("One To Many is Done..!!");
          factory.close();

    }
}
```

Notes:

- In line numbers 28,34,40 we are adding parent to child object
- In line number 44,45,46 we are saving all child objects, but you know some thing, according to this code at line number 45, first child object will be saved with the parent, at 45,46 just child (customer object) only will be saved as parent is saved already

## Hibernate One To Many Bidirectional Mapping Example

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…

files required…

- Vendor.java [pojo]
- Customer.java [pojo]
- Vendor.hbm.xml
- Customer.hbm.xml
- hibernate.cfg.xml
- OurLogic.java [Our logic]

**Vendor.java**

```
package str;


import java.util.Set;

public class Vendor {

    private int vendorId;
    private String vendorName;
    private Set children;

    public int getVendorId() {
        return vendorId;
    }
    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }
    public String getVendorName() {
        return vendorName;
    }
    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
```

```
    }
    public Set getChildren() {
        return children;
    }
    public void setChildren(Set children) {
        this.children = children;
    }

}
```

**Customer.java**

```
package str;

public class Customer {

    private int customerId;
    private String customerName;
    private int forevenId;
    private Vendor parentObjets;

    public Vendor getParentObjets() {
        return parentObjets;
    }
    public void setParentObjets(Vendor parentObjets) {
        this.parentObjets = parentObjets;
    }
    public int getCustomerId() {
        return customerId;
    }
    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }
    public String getCustomerName() {
        return customerName;
    }
    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }

    public int getForevenId() {
        return forevenId;
    }
    public void setForevenId(int forevenId) {
        this.forevenId = forevenId;
```

```
    }

}
```

**Vendor.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Vendor" table="vendor">

<id name="vendorId" column="vendid"  />
<property name="vendorName" column="vendname" length="10"/>

<set name="children" cascade="all" inverse="true">

<key column="forevenid" />
<one-to-many class="str.Customer" />

</set>

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

Note:

•Hear we are writing one new attribute inverse=”true” , means we are intimating to hibernate that we are doing Bi Directional operation

•But default value of inverse=”false“

**Customer.hbm.xml**

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Customer" table="customer">

<id name="customerId" column="custid"  />
<property name="customerName" column="custname" length="10"/>
```

```
<property name="forevenId" column="forevenid"  insert="false" />

<many-to-one name="parentObjets" column="PrentsIds" cascade="all"/>

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

**hibernate.cfg.xml**

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Customer.hbm.xml"></mapping>
<mapping resource="Vendor.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**OurLogic.java**

```
package str;


import java.util.HashSet;
import java.util.Set;

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

public class OurLogic {
```

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

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

    SessionFactory factory = cfg.buildSessionFactory();
    Session session = factory.openSession();

    Vendor v =new Vendor();

              v.setVendorId(101);
              v.setVendorName("java4s");

     Customer c1=new Customer();

               c1.setCustomerId(504);
               c1.setCustomerName("customer4");

     Customer c2=new Customer();

               c2.setCustomerId(505);
               c2.setCustomerName("customer5");

     Customer c3=new Customer();

               c3.setCustomerId(506);
               c3.setCustomerName("customer6");

        // one-to-many
        Set s=new HashSet();

                 s.add(c1);
                 s.add(c2);
                 s.add(c3);

               v.setChildren(s);

        // many-to-one

          c1.setParentObjets(v);
          c2.setParentObjets(v);
          c3.setParentObjets(v);
```

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

                                session.save(c1);
                            //session.save(v);

        tx.commit();

        session.close();
        System.out.println("One To Many Bi-Directional is Done..!!");
        factory.close();

    }
}
```

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

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

**Many To Many Relationship Example**

files required…..
•Student.java
•Course.java
•Course.hbm.xml
•Student.hbm.xml
•OurLogic.java
•hibernate.cfg.xml

**Student.java**

```
package str;

import java.util.Set;

public class Student {

   private int studentId;
   private String studentName;
   private int marks;

   private Set courses;

   public int getStudentId() {
      return studentId;
   }

   public void setStudentId(int studentId) {
      this.studentId = studentId;
   }

   public String getStudentName() {
      return studentName;
   }

   public void setStudentName(String studentName) {
      this.studentName = studentName;
   }
```

```
    public int getMarks() {
        return marks;
    }

    public void setMarks(int marks) {
        this.marks = marks;
    }

    public Set getCourses() {
        return courses;
    }

    public void setCourses(Set courses) {
        this.courses = courses;
    }

}
```

**Course.java**

```
package str;

import java.util.Set;

public class Course {

    private int courseId;
    private String courseName;
    private int duration;

    private Set students;

    public int getCourseId() {
        return courseId;
    }

    public void setCourseId(int courseId) {
        this.courseId = courseId;
    }

    public String getCourseName() {
        return courseName;
    }
```

```
    public void setCourseName(String courseName) {
        this.courseName = courseName;
    }

    public int getDuration() {
        return duration;
    }

    public void setDuration(int duration) {
        this.duration = duration;
    }

    public Set getStudents() {
        return students;
    }

    public void setStudents(Set students) {
        this.students = students;
    }

}
```

**Student.hbm.xml**

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

<hibernate-mapping>
<class name="str.Student" table="student">

<id name="studentId" column="studentid" />

<property name="studentName" column="studentname" length="20"/>
<property name="marks" />

<set name="courses" cascade="all" table="students_courses">

<key column="student_id "/>
<many-to-many class="str.Course" column="course_id" />

</set>

</class>
```

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

**Course.hbm.xml**

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

<hibernate-mapping>
<class name="str.Course" table="courses">

<id name="courseId" column="courseid" />

<property name="courseName" column="coursename" length="20"/>
<property name="duration" />

<set name="students" inverse="false" cascade="all" table="students_courses">

<key column="course_id" />
<many-to-many class="str.Student" column="student_id "/>

</set>

</class>

</hibernate-mapping>
```

**OurLogic.java**

```
package str;

import java.util.HashSet;
import java.util.Set;

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

public class OurLogic {

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

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

SessionFactory factory = cfg.buildSessionFactory();
Session session = factory.openSession();

Student s1=new Student();
s1.setStudentId(100);
s1.setStudentName("James");
s1.setMarks(98);

Student s2=new Student();
s2.setStudentId(101);
s2.setStudentName("Lee");
s2.setMarks(99);

Course c1=new Course();
c1.setCourseId(500);
c1.setCourseName("Hibernate");
c1.setDuration(7);

Course c2=new Course();
c2.setCourseId(501);
c2.setCourseName("Java");
c2.setDuration(30);

Set s =new HashSet();
     s.add(c1);
     s.add(c2);

s1.setCourses(s);
s2.setCourses(s);

   Transaction tx = session.beginTransaction();

                session.save(s1);
                session.save(s2);

   tx.commit();

   session.close();
   System.out.println("Many To Many Bi-Directional is Done..!!");
   factory.close();
```

```
   }
}
```

**hibernate.cfg.xml**

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Student.hbm.xml"></mapping>
<mapping resource="Course.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

## Hibernate One to One Mapping Example

Let us see few points regarding this one to one mapping..

•One object is associated with one object only

•In this relationship, one object of the one pojo class contains association with one object of the another pojo class

•To apply one to one relationship between two pojo class objects it is possible by without taking a separate foreign key column in the child table of the database

•To apply one to one relationship, we copy the primary key value of parent object into primary key value of the child object. So that the relationship between two objects is one to one

•If we want to copy parent object primary key value into child object primary key, we need to use aspecial generator class given by hibernate called foreign

•Actually this foreign generator is only used in one to one relationship only

•We are going to apply one to one between student and address pojo classes, hear the relation is one address is assigned for one student only

•In order to get one to one relationship between student and address, we are copying primary key value

of student into primary key value of address
Example…..
files required…
•Student.java
•Address.java
•Address.hbm.xml
•Student.hbm.xml
•OurLogic.java
•hibernate.cfg.xml

**Student.java**

```
package str;

public class Student {

    private int studentId;
    private String studentName;

    public int getStudentId() {
        return studentId;
    }
    public void setStudentId(int studentId) {
        this.studentId = studentId;
    }
    public String getStudentName() {
        return studentName;
    }
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }

}
```

**Address.java**

```
package str;
public class Address {

    private int addressId;
    private String city;
    private String state;
    private Student s;

    public Student getS() {
```

```
        return s;
    }
    public void setS(Student s) {
        this.s = s;
    }
    public int getAddressId() {
        return addressId;
    }
    public void setAddressId(int addressId) {
        this.addressId = addressId;
    }
    public String getCity() {
        return city;
    }
    public void setCity(String city) {
        this.city = city;
    }
    public String getState() {
        return state;
    }
    public void setState(String state) {
        this.state = state;
    }

}
```

**Address.hbm.xml**

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

<hibernate-mapping>
<class name="str.Address" table="address">

<id name="addressId" column="addressid" >
<generator>
<param name="property">s</param>
</generator>
</id>
<property name="city" column="city" length="10"/>
<property name="state" column="state" length="10"/>

<one-to-one name="s" class="str.Student" cascade="all" />
```

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

Student.hbm.xml

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Student" table="student">

<id name="studentId" column="studentid"  />
<property name="studentName" column="studentname" length="10"/>

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

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Student.hbm.xml"></mapping>
<mapping resource="Address.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

OurLogic.java

```
package str;


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

public class OurLogic {

   public static void main(String args[])
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();

      Student s=new Student();
      s.setStudentId(100);
      s.setStudentName("java4s");

      Address ad = new Address();
      ad.setAddressId(509);
      ad.setCity("Carry");
      ad.setState("NC");
      ad.setS(s);

         Transaction tx = session.beginTransaction();

                  session.save(ad);

         tx.commit();

         session.close();
         System.out.println("One to One is Done..!!");
         factory.close();

   }
}
```

Let us see by loading the object form the database whether the one to one is working fine or not…

**OurLogic_loading.java**

```
package str;


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

public class OurLogic_loading {

   public static void main(String args[])
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();

      Object o = session.get(Address.class, new Integer(100));
      Address a = (Address)o;
      System.out.println(a.getCity());

      Student s=a.getS();
      System.out.println(s.getStudentName());

         session.close();
         System.out.println("One to One is Done..!!");
         factory.close();

   }
}
```

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…….

- none (default)
- save
- update
- save-update
- delete
- all
- all-delete-orphan

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

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

Hibernate Left Join, Hibernate Left Join Example

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….
•Vendor.java
•Customer.java
•Vendor.hbm.xml
•Customer.hbm.xml
•hibernate.cfg.xml
•urLogic.java

**Vendor.java**

```
package str;


import java.util.Set;

public class Vendor {

    private int vendorId;
    private String vendorName;
    private Set children;
```

```
    public int getVendorId() {
        return vendorId;
    }
    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }
    public String getVendorName() {
        return vendorName;
    }
    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
    }
    public Set getChildren() {
        return children;
    }
    public void setChildren(Set children) {
        this.children = children;
    }

}
```

Vendor.xml

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Vendor" table="vendor">

<id name="vendorId" column="vendid"  />
<property name="vendorName" column="vendname" length="10"/>

<set name="children" cascade="all" lazy="false">

<key column="forevenid" />
<one-to-many class="str.Customer"/>

</set>

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

Customer.java

```
<?xml version="1.0"?>

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

<hibernate-mapping>
<class name="str.Customer" table="customer">

<id name="customerId" column="custid"  />
<property name="customerName" column="custname" length="10"/>

<property name="forevenId" column="forevenid"  insert="false" />

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

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Customer.hbm.xml"></mapping>
<mapping resource="Vendor.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

urLoic.java

```
package str;


import java.util.Iterator;
import java.util.List;

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

public class urLogic {

   public static void main(String args[])
   {

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

      SessionFactory factory = cfg.buildSessionFactory();
      Session session = factory.openSession();

      Query qry= session.createQuery("select v.vendorName, c.customerName from Vendor v Left Join
v.children c");

      List l = qry.list();
      Iterator it=l.iterator();

      while(it.hasNext())
      {
         Object rows[] = (Object[])it.next();
         System.out.println(rows[0]+ " -- " +rows[1]);
      }

   }
}
```

Explanation:

•Actually in urLogic.java see line number 27, i have written select v.vendorName, c.customerName from Vendor v Left Join v.children c

•See before Left Join key word, i selected from Vendor v, after Left Join keywork v.children c

•Means, actually we know in one to many if we select the data from parent then, we will get the data

automatically from the children table, i mean records mapping with the parent table, hope you remembered

•Let in parent we have 5 records, and in children we have 3 records. And 2 records are having relation with children table records

Hibernate Caching Mechanism, Hibernate Cache

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

Hibernate First Level Cache Example

Let us try to understand the first level cache in hibernate

•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 the database and the loaded object will be stored in the cache memory maintained by that session object

•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**

```
Session ses1 = factory.openSession();
Object ob1 = ses1.get(Student.class, new Integer(101));

Object ob2 = ses1.get(Student.class, new Integer(101));
```

```
Object ob3 = ses1.get(Student.class, new Integer(101));
Object ob4 = ses1.get(Student.class, new Integer(101));

--
session.close();

Session ses2 = factory.openSession();
Object ob5 = ses2.get(Student.class, new Integer(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

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

```
Session ses = factory.openSession();
Object ob = ses1.get(Student.class, new Integer(101));
Student s = (Student)ob
System.out.println(s.getStudentId());

ses.evict(s);

Object ob1 = ses.get(Student.class, new Integer(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

## 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 classin hibernate configuration file like…

```
<property name="hibernate.cache.provider_class">
org.hibernate.cache.EhCacheProvider
</property>
```

Configure cache elementfor a class in hibernate mapping file…
<cache usage="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

**Hibernate Second Level Cache Example**

Let us see the example on this hibernate second level cache. please go through the concept on this second level cache, still if you have any doubt [ Click hear ]
Files required….
•Product.java [ Pojo class]
•ForOurLogic.java
•Product.hbm.xml
•ehcache.xml
•hibernate.cfg.xml

**Product.java**

```
package str;

public class Product{

   private int productId;
   private String proName;
   private double price;
```

```
    public void setProductId(int productId)
    {
       this.productId = productId;
    }
    public int getProductId()
    {
       return productId;
    }

    public void setProName(String proName)
    {
       this.proName = proName;
    }
    public String getProName()
    {
       return proName;
    }

    public void setPrice(double price)
    {
       this.price = price;
    }
    public double getPrice()
    {
       return price;
    }
}
```

**ForOurLogic.java**

```
package str;

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

public class ForOurLogic {

    public static void main(String[] args)
    {

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

        SessionFactory factory = cfg.buildSessionFactory();
        Session session1 = factory.openSession();
```

```
        Object o=session1.load(Product.class,new Integer(105));

        Product s=(Product)o;
        System.out.println("Loaded object product name is___"+s.getProName());
        System.out.println("Object Loaded successfully.....!!");
        session1.close();

        System.out.println("-----------------------------");
        System.out.println("Waiting......");

        try{
            Thread.sleep(6000);
        }
        catch (Exception e) {
        }

        System.out.println("6 seconds compelted......!!!!!!!!");

        Session session2 = factory.openSession();
        Object o2=session2.load(Product.class,new Integer(105));

        Product s2=(Product)o2;
        System.out.println("Loaded object product name is___"+s2.getProName());
        System.out.println("Object loaded from the database...!!");
        session2.close();

        Session session3 = factory.openSession();
        Object o3=session3.load(Product.class,new Integer(105));

        Product s3=(Product)o3;
        System.out.println("Loaded object product name is___"+s3.getProName());
        System.out.println("Object loaded from global cache successfully.....!!");
        session3.close();

        factory.close();
    }

}
```

**Product.hbm.xml**

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

```
<hibernate-mapping>
<class name="str.Product" table="products">

<cache usage="read-only" />
<id name="productId" column="pid"  />
<property name="proName" column="pname" length="10"/>
<property name="price"/>

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

**hibernate.cfg.xml**

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="cache.provider_class">
org.hibernate.cache.EhCacheProvider
</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

<mapping resource="Product.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

**ehcache.xml**

```
<?xml version="1.0"?>

<ehcache>

<defaultCache maxElementsInMemory="100" eternal="false" timeToIdleSeconds="120"
timeToLiveSeconds="200" />
<cache name="str.Product" maxElementsInMemory="100" eternal="false" timeToIdleSeconds="5"
```

timeToLiveSeconds="200" />

</ehcache>

**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 assign values 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

Difference Between Merge And Update Methods In Hibernate

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**

---

---

```
SessionFactory factory = cfg.buildSessionFactory();
Session session1 = factory.openSession();

Student s1 = null;
Object o = session1.get(Student.class, new Integer(101));
s1 = (Student)o;
session1.close();

s1.setMarks(97);

Session session2 = factory.openSession();
Student s2 = null;
Object o1 = session2.get(Student.class, new Integer(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 update and merge methods will come into picture when ever we loaded the same object again and again into the database, like above.

Difference Between Hibernate Save And Persist Methods

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 Incrementmeans 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.

Hibernate Annotations Introduction

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.....**

```
public @interface Java4s
{
   // code.....
}
```

For working with annotations, our java version must be 1.5 or higher and hibernate version must behibernate 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

## Hibernate One To Many Annotation Example

Let us see an example on one to many annotations mapping…

Files required..
•Customers.java
•Vendor.java
•ForOurLogic.java
•hibernate.cfg.xml

**Customers.java**

```
package str;


import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
```

```
@Table(name = "Customers")
public class Customers{

    @Id
    @Column(name = "custid")
    private int customerId;

    @Column(name = "custName", length=10)
    private String customerName;

    public int getCustomerId() {
        return customerId;
    }

    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }

}
```

**Vendor.java**

```
package str;


import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
```

```
@Table(name = "Vendor")

public class Vendor{

    @Id
    @Column(name = "vid")
    private int vendorId;

    @Column(name = "vname", length=10)
    private String vendorName;

    @OneToMany(fetch=FetchType.LAZY, targetEntity=Customers.class, cascade=CascadeType.ALL)
    @JoinColumn(name = "venid", referencedColumnName="vid")

    private Set children;

    public int getVendorId() {
        return vendorId;
    }

    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }

    public String getVendorName() {
        return vendorName;
    }

    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
    }

    public Set getChildren() {
        return children;
    }

    public void setChildren(Set children) {
        this.children = children;
    }

}
```

**hibernate.cfg.xml**

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

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

**ForOurLogic.java**

```
package str;

import java.util.HashSet;
import java.util.Set;

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

public class ForOurLogic {

```

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

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

        SessionFactory factory = new AnnotationConfiguration().configure().buildSessionFactory();
        Session session = factory.openSession();

          Vendor v=new Vendor();
          v.setVendorId(100);
          v.setVendorName("java4s");

          Customers c1=new Customers();
          c1.setCustomerId(500);
          c1.setCustomerName("customer1");

          Customers c2=new Customers();
          c2.setCustomerId(501);
          c2.setCustomerName("customer2");

          Set s=new HashSet();
          s.add(c1);
          s.add(c2);

          v.setChildren(s);

          Transaction tx=session.beginTransaction();
              session.save(v);
          tx.commit();

        session.close();
        System.out.println("One to Many Annotatios Done...!!!!!!");
        factory.close();
    }

}
```

**Hibernate Many To One Annotation Example**

Will find the example on hibernate many to one mapping using annotations

Files required…

- Customers.java
- Vendor.java
- hibernate.cfg.xml

•ForOurLogic.java

**Customers.java**

```
package str;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;

@Entity
@Table(name = "Customers")
public class Customers{

    @Id
    @Column(name = "custid")
    private int customerId;

    @Column(name = "custName", length=10)
    private String customerName;

    @ManyToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="venid",referencedColumnName="vid")
    private Vendor parent;

    public int getCustomerId() {
        return customerId;
    }

    public void setCustomerId(int customerId) {
        this.customerId = customerId;
    }

    public String getCustomerName() {
        return customerName;
    }

    public void setCustomerName(String customerName) {
        this.customerName = customerName;
    }
```

```
    public Vendor getParent() {
        return parent;
    }

    public void setParent(Vendor parent) {
        this.parent = parent;
    }

}
```

**Vendor.java**

```
package str;

import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name = "Vendor")

public class Vendor{

    @Id
    @Column(name = "vid")
    private int vendorId;

    @Column(name = "vname", length=10)
    private String vendorName;

    public int getVendorId() {
        return vendorId;
    }

    public void setVendorId(int vendorId) {
        this.vendorId = vendorId;
    }

    public String getVendorName() {
```

```
        return vendorName;
    }

    public void setVendorName(String vendorName) {
        this.vendorName = vendorName;
    }

}
```

**hibernate.cfg.xml**

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

<hibernate-configuration>
<session-factory>
<property name="connection.driver_class">oracle.jdbc.driver.OracleDriver
</property>
<property name="connection.url">jdbc:oracle:thin:@www.java4s.com:1521:XE</property>
<property name="connection.username">system</property>
<property name="connection.password">admin</property>

<property name="dialect">org.hibernate.dialect.OracleDialect</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>

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

**ForOurLogic.java**

```
package str;

import java.util.HashSet;
import java.util.Set;

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

public class ForOurLogic {
```

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

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

    SessionFactory factory = new AnnotationConfiguration().configure().buildSessionFactory();
    Session session = factory.openSession();

    Vendor v =new Vendor();

    v.setVendorId(100);
    v.setVendorName("java4s6");

    Customers c1=new Customers();

     c1.setCustomerId(504);
     c1.setCustomerName("customer4");
     c1.setParent(v);

     Customers c2=new Customers();

     c2.setCustomerId(505);
     c2.setCustomerName("customer5");
     c2.setParent(v);

     Customers c3=new Customers();

     c3.setCustomerId(506);
     c3.setCustomerName("customer6");
     c3.setParent(v);

Transaction tx = session.beginTransaction();

            //session.save(v);
           session.save(c2);
           session.save(c2);
           session.save(c3);

tx.commit();
session.close();
System.out.println("Many to One with annotation done...!!");
factory.close();
```

```
    }
}
```