Explore More

Subcription : Premium CDAC NOTES & MATERIAL @99

Contact to Join
Premium Group

Click to Join
Telegram Group

<CODEWITHARRAY'S/>

For More E-Notes

Join Our Community to stay Updated

TAP ON THE ICONS TO JOIN!

	codewitharrays.in freelance project available to buy contact on 8007592194	
SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

	freelance_Project available to buy contact on 8007592194	
21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+NextJs+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql

41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48	Train Ticket Booking Project	React+Springboot+MySql
49	Quizz Application Project	JSP+Springboot+MySql
50	Hotel Room Booking Project	React+Springboot+MySql
51	Online Crime Reporting Portal Project	React+Springboot+MySql
52	Online Child Adoption Portal Project	React+Springboot+MySql
53	online Pizza Delivery System Project	React+Springboot+MySql
54	Online Social Complaint Portal Project	React+Springboot+MySql
55	Electric Vehical management system Project	React+Springboot+MySql
56	Online mess / Tiffin management System Project	React+Springboot+MySql
57		React+Springboot+MySql
58		React+Springboot+MySql
59		React+Springboot+MySql
60		React+Springboot+MySql

Spring Boot + React JS + MySQL Project List

Sr.No	Project Name	YouTube Link
1	Online E-Learning Hub Platform Project	https://youtu.be/KMjyBaWmgzg?si=YckHuNzs7eC84-IW
2	PG Mate / Room sharing/Flat sharing	https://youtu.be/4P9clHg3wvk?si=4uEsi0962CG6Xodp
3	Tour and Travel System Project Version 1.0	https://youtu.be/-UHOBywHaP8?si=KHHfE_A0uv725f12
4	Marriage Hall Booking	https://youtu.be/VXz0kZQi5to?si=llOS-QG3TpAFP5k7
5	Ecommerce Shopping project	https://youtu.be/vJ_C6LkhrZ0?si=YhcBylSErvdn7paq
6	Bike Rental System Project	https://youtu.be/FIzsAmIBCbk?si=7ujQTJqEgkQ8ju2H
7	Multi-Restaurant management system	https://youtu.be/pvV-pM2Jf3s?si=PgvnT-yFc8ktrDxB
8	Hospital management system Project	https://youtu.be/IynIouBZvY4?si=CXzQs3BsRkjKhZCw
9	Municipal Corporation system Project	https://youtu.be/cVMx9NVyI4I?si=qX0oQt-GT-LR_5jF
10	Tour and Travel System Project version 2.0	https://youtu.be/_4u0mB9mHXE?si=gDiAhKBowi2gNUKZ

Sr.No	Project Name	YouTube Link
11	Tour and Travel System Project version 3.0	https://youtu.be/Dm7nOdpasWg?si=P_Lh2gcOFhlyudug
12	Gym Management system Project	https://youtu.be/J8_7Zrkg7ag?si=LcxV51ynfUB7OptX
13	Online Driving License system Project	https://youtu.be/3yRzsMs8TLE?si=JRI_z4FDx4Gmt7fn
14	Online Flight Booking system Project	https://youtu.be/m755rOwdk8U?si=HURvAY2VnizIyJlh
15	Employee management system project	https://youtu.be/lD1iE3W_GRw?si=Y_jv1xV_BljhrD0H
16	Online student school or college portal	https://youtu.be/4A25aEKfei0?si=RoVgZtxMk9TPdQvD
17	Online movie booking system project	https://youtu.be/Lfjv_U74SC4?si=fiDvrhhrjb4KSlSm
18	Online Pizza Delivery system project	https://youtu.be/Tp3izreZ458?si=8eWAOzA8SVdNwlyM
19	Online Crime Reporting system Project	https://youtu.be/0UlzReSk9tQ?si=6vN0e70TVY1GOwPO
20	Online Children Adoption Project	https://youtu.be/3T5HC2HKyT4?si=bntP78niYH802I7N

Java means DURGA SOFT..

JAVA FRAMEWORKS

India's No.1 Software Training Institute

DURGASOFT

www.durgasoft.com Ph: 9246212143 ,8096969696

HIBERNATE

Syllabus:

- Key Features of Hibernate
  - Enterprise Application Architecture.
  - ORM (Object Relational Mapping).
  - Hibernate Architecture.
  - Mapping and Configuration files in Hibernate.
  - Installation, Simple Hibernate Application Requirements.
  - Introduction to IDE.
    - Hello World program in Hibernate
    - Hello World program in Hibernate using eclipse
- Simple CRUD operations.
- Important methods of Session Interface.
- <generator> element.
- State of Objects in Hibernate.
- Inheritance mapping
  - Table per class
  - Table per sub class
  - Table per concrete class
- HQL (Hibernate Query Language).
- Criteria Query.
- Native SQL Query.
- Named Query.

- Relationships in Hibernate.
    - One to One
    - One to Many
    - Many to One
    - Many to Many
- Hibernate Caching Mechanism
    - First level cache
    - Second level cache
- Hibernate Annotations.
- Integration of Hibernate with and Struts.

Notes:

Hibernate Framework

About Developers:

- Hibernate was started in 2001 by Gavin King with colleagues from Cirrus Technologies as an alternative to using EJB2-style entity beans. Its original goal was to offer better persistence capabilities than offered by EJB2 by simplifying the complexities and supplementing missing features.

- In early 2003, the Hibernate development team began Hibernate2 releases, which offered many significant improvements over the first release.
- JBoss, Inc. (now part of Red Hat) later hired the lead Hibernate developers in order to further its development.
- In 2005, Hibernate version 3.0 was released. Key features included a new Interceptor/Callback architecture, user defined filters, and JDK 5.0 Annotations (Java's metadatafeature). As of 2010, Hibernate 3 (version 3.5.0 and up) was a certified implementation of the Java Persistence API 2.0 specification via a wrapper for the Core module which provides conformity with the JSR 317 standard.[3]
- In Dec 2011, Hibernate Core 4.0.0 Final was released. This includes new features such as multi-tenancy support, introduction of ServiceRegistry (a major change in how Hibernate builds and manages "services"), better Session opening from SessionFactory, improved integration via *org.hibernate.integrator.spi.Integrator* and auto discovery,internationalization support and message codes in logging, and a clearer split between API, SPI and implementation classes.[4]
- In Dec 2012, Hibernate ORM 4.1.9 Final was released.[5]
- In Apr 2014 , Hibernate ORM 4.3.5 Final was released.[5]

Hibernate ORM

Where we can contribute as a hibernate developer?

While making **MVC** based architecture Applications

We can contribute under Data Access Layer (DAO)

Before Hibernate what we have to make DAO's?

For making DAO classes in industry they used to depends on EJB entity beans+jdbc

Features of EJBEntity beans ?

i. Portable Object Relations for making Persistence layer
ii. Caching support (bean managed ,container managed)
iii. QL(query language)database independent language support
iv. Hide Implementations and we can access through Interface references(data abstraction)

EJBEntity beans disadvantages ?

i. Dependency of application server
ii. Less productive (take more time for implementation)
iii. Compile time Exception Handling while using jdbc code
iv. Complete Invasive Strictly force us to impalement EJb classes \

Futures of Hibernate ?

i. It will provide Portable Object Relationship(associations , Inheritance)
ii. Caching support(session level ,Session Factory level)
iii. HQL support, hibernate Query Language (database independent queries developer can write ,at runtime it will fire queries on underlying Database)
iv. Criteria support (we can pass conditional query's in very easy manner)
v. Named parameters support (run time values we can pass to query's)
vi. No need of compile time Exception handling here they refitted to run time exceptions
vii. Auto DDL support (table creation , updating and deletion)
viii. Auto primary key incrimination
ix. No Application Server dependency

Usually for making enterprise applications software industry will fallows**MVC** rules. As per MVC Application should divide into 3sub layers

Presentation →Business→Data Access

For making Abstraction On each layer they will put Interfaces on top of their Implementations

Presentation→ Business Interface → Data Access Interface

↓ ↓

Business Implementations Data Access implementation

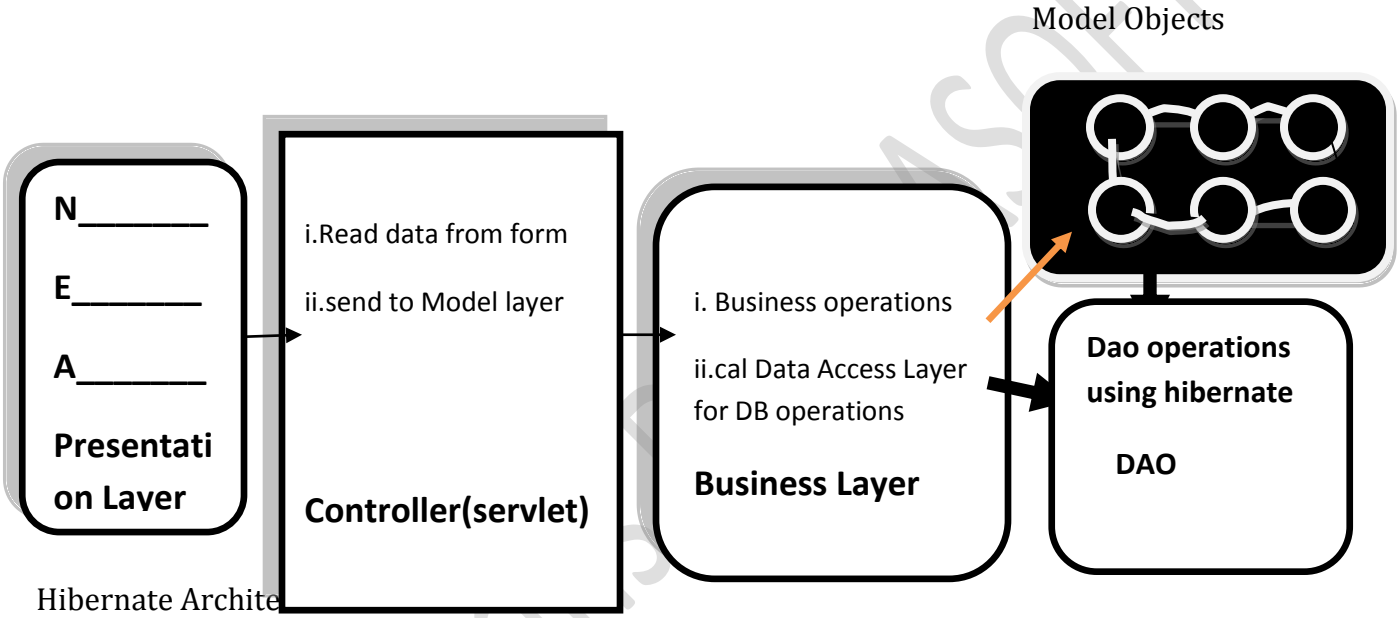
So we know as a hibernate developer we should make Data Access layer

Then The common requirements in Data Access layer is

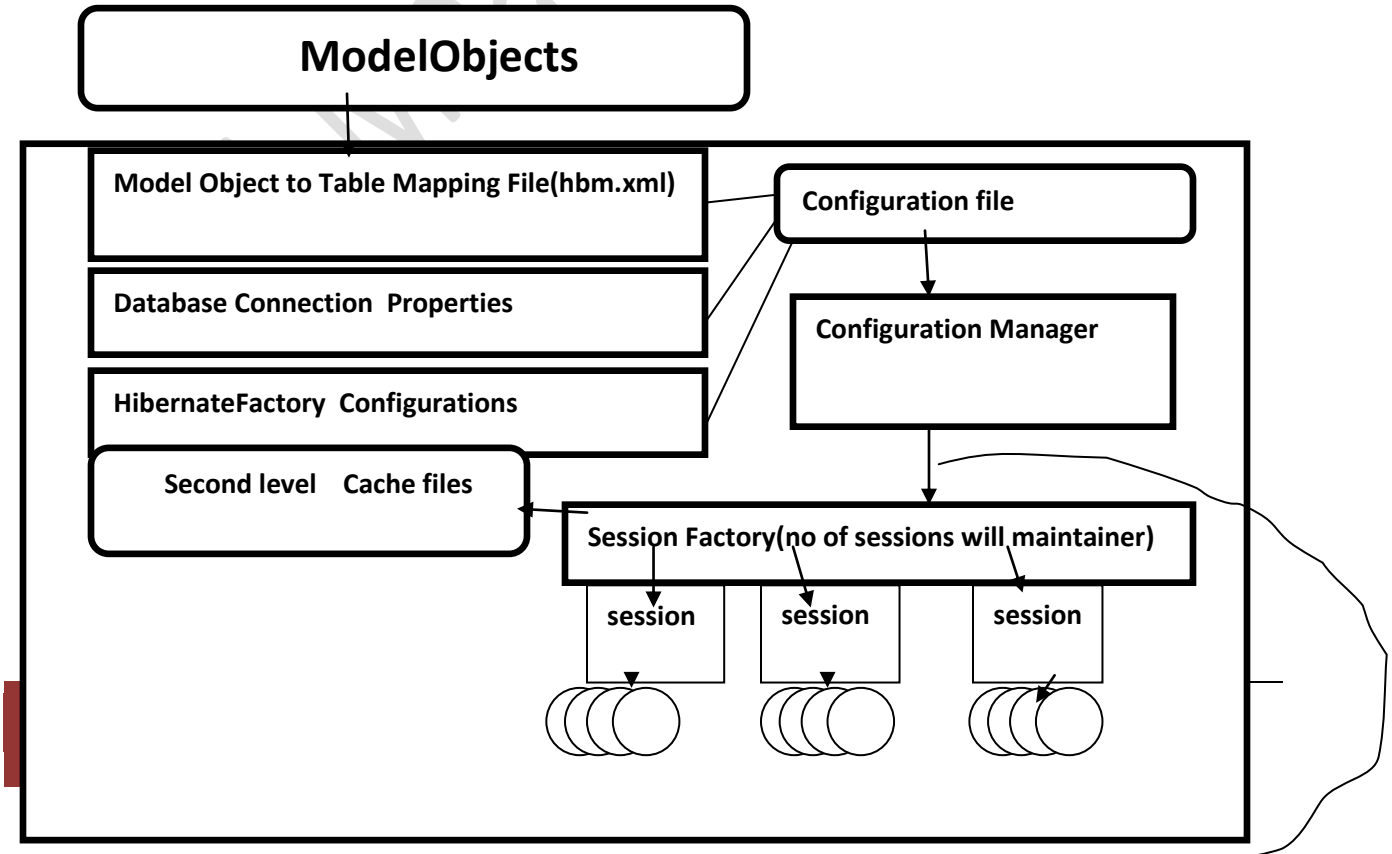
i. Create tables in database(database dba will provide these tables)
ii. save records into database(ex: student details want to store in database)
iii. update records into database(ex: student details want to update in database)
iv. delete records from database(ex: student details want to delete from database)
v. select records with condition(ex: student details want to select from database by applying condition like where sroll=1220)
vi. select some records (ex: select students details only male or select student who is having marks between 500 to 550)
vii. select All (ex: All students details want to see)
viii. store data into temp files for the caching and to reduce database calls

So if I use Hibernate Can I achieve all these ?
Yes we can provide all these futures by using hibernate

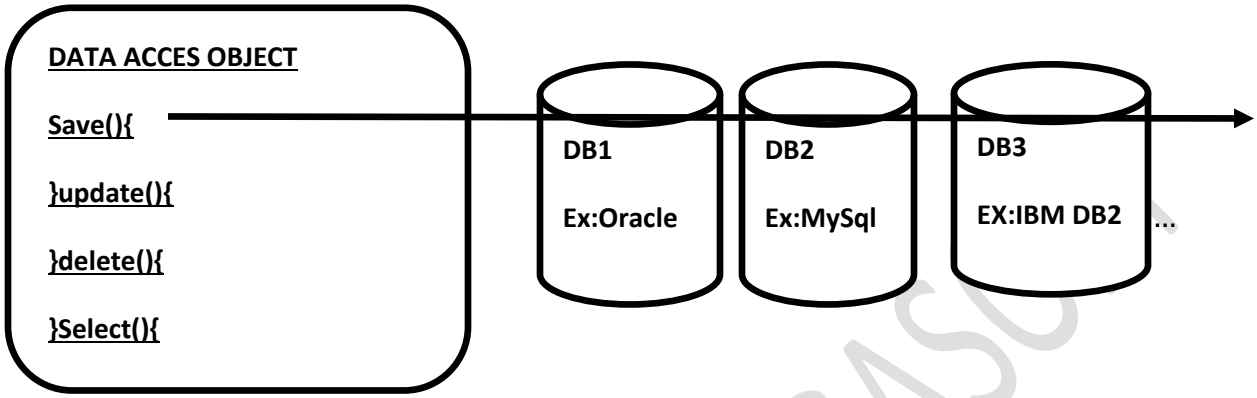
Application architecture by using MVC



Hibernate Archite



Auto schema generation



Model Object:

a. It will store data(state) in Objects state before moving object into database state .
b. It will store data into objects state while selecting data from database

With the files we have the following draw backs:-

1)Files are very complex to work.The programmer must know atleast one programming language ,in order to work with files.

2)Files does not support Query language.

3)Files are less secure.

4)Maintaining data in a file is complex.

5)In order to overcome the draw back of File manage ment System.we entered into DataBase ManageMent System(DBMS).

DrawBacks of Jdbc

1)In jdbc ,if we open a database connection then,we are responsible to close it.if the connection is not closed, later there is a chance of getting out of connection problem.

2)In jdbc,we write SQL commands at various places in the programe.after the programe has to created,if the lable structure or designe is modified then the jdbc programe does not work.in order to work it,we need to go to the each place of sql command and we need modify according to the new designe.

3)Jdbc provides error codes(eg:ORA-666)to the java programmer,whenever an exception is occurred.but java programmers are unknow to about the error codes of the datadases.

4)In enterprise applications,the data flow with the application, will be in the form of Objects.But while Storing the data in a data baseusing Jdbc then the object will be converted into text.Because Jdbc does not transfer Objects directly.

5)In order to over come the above problems, Sun Micro System provided Entity Beans in in J2EE.

EntityBeans:-

1)Entity Beans are the persistant components given under EJB technology(Enterprise Java Beans).

2)In order to overcome the draw backs of JDBC technology given under J2SE or JSE ,Sun micro Systems introduced Entity beans under J2EE or JEE.

3)Entity Beans was the first technology used ORM mechanism(Object Relational Mechanism).

4)Entity Beans are introduced,in order to too and fro the data b/w a java application and db,in the form of objects.

5)Mapping is a mediator b/w an object model and a relational model.

6)java followes Object model and DataBase follows Relational model.The principles of an object model and relational model are different.So Mapping acting as mediator.

JAVA Means DURGASOFT


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com

DrawBacks of Entity Beans:

Entity Beans are heavy weight components.

Entity Beans only run on a middle were server.so additionally we need to install a middle were server in a system.

If any problem occure while running an entity bean then we need to shutdown the server and later we need to restart the servr. Until the server is going to shutdown and until the server is restart the programmer wait it means entity beans are slow.

Unit testing of an application is difficult :-

is difficult in ordr to overcome the drawbacks, JDBC technology and entity beans technology we got hibernate

q) what is an hibernate?

1) Hibernate is an ORM tool given, to transfer back and fort the data b/w a java application and a database, in the form of objects.

2) Hibernate is an open source and a light weight tool given by gavin king of safttee (JBOSS)

Hibernate is not saying that our class extends of class that way hibernate is called non-invasive

3) Hibernate is a non-invasive framework.

4) Non-invasive Fw's means, it does not force a programmer to extends (or) to implement their classes from any base class (or) interface given by the Fw

5) Invasive Fw means,it will force the programmer to implement (or) extends their class either bas class (or) interface that is given by Framework.

Hibernate can be use either standalone app, servlet (or) spring any wre can be run.

We can not writ any b.l in hibernate.

P c myframe extends Jframe

{

Hibernate

Standlone app

}

p c myservlet extends GS

{

---

hibernate

web application

}

So hibernate can be run inside server and outside server

Hibernate aim is persistent the data in db.

NOTE:- Hibernate runs with in a server and even with out a server . it means hibernate is suitable for all types of java applications.

We can us hibernate for standalone applications and in web application and also in an enterprise application.

Hibernate is tool, used for implementing persistence logic for the real time applications.

Q).**why mapping?**

A) Generally an object contains 3 properties .

1) Identity (name)

2) State (values)

3) Behaviour (methods)

But, while Storing an object in a dbase, only values can be stored. It is not possible to store identity and behavior.

In order to inform what value of an object should be stored in what column of table, mapping is required

NOTE:-Mapping can be done using two way 1) using xml 2) using annotations.

ADVANTAGE OF JDBC:-

1) Inheritance does not support jdbc.

2) Relationships does not support jdbc.
3) Collections does not support jdbc.

ADVANTAGES OF HIBERNATE:-1) Hibernate supports inheritance, associations (relationship),collections.

i) Hibernate when you saved derived class object, then automatically the base class data also inserted in db. It means in hibernate we got inheritance support.
ii) Hibernate support relation ships like one to many, many-to-one,one-to-one and many-to-many
iii) Hibernate support the collections like list, set and map

2) In Hibernate an exception Translator is given, so it convert the checked exceptions into un checked exceptions. So in hibernate we have all unchecked Exception only

ii) if all exceptions are unchecked exception, the we no need write , try, catch, and throws exception. While writing hibernate code.

3)Hibernate support the automatic generation of primary keys.

4) In Hibernate, we got it's own Query language , called HQL (Hibernate Query Language) so we can develop db independent persistence logic.

ii) In Jdbc, we us SQL, so Jdbc provides dbase dependency persistence logic.

5)If the database Scheme does not exit then only it will insert th data otherwise throws an exception (lable (or) View does not exit)

6) Hibernate supports coaching mechanism by this, the number of round trips b/w an application and a dbase will be reduced.

ii) By using coaching technique, an application Performance will be increased automatically.

7) Hibernate support coaching mechanism by this, the number of round trips b/w an application and a dbase will be reduced.

ii) By using coaching technique, an application performance will be increased automatically.

7) Hibernate support annotations.

8) Hibernate has provided dialect classes. So w no ned write SQL Queries in hibernate. Instead, we apply methods provided by Hibernate API.

NOTE:-in hibernate, we have one draw back called, hibernate does not support distributed transactions.

ii) in order to overcome the drawback of hibernate JBOSS (Soft tree original name) communicative realses an extension of hibernate called Hibernate hards.

ii)at present Hibernate shards Befa23.0 is realsed

APPLICATION REQUIREMENTS OF HIBERNATE:-

1) Pojo Class.
2) Mapping Xml (<anyname >.hbm.xml)
3) Configuration xml (<anyname >. Cfg.xml)

POJO CLASS:-Pojo class means no need to extends from any java class (or) implement any interface

ii)this Pojo class contains private properties variable and for each property a setter and getter method.

Example: public class customer

{

Private int customerid;

Private string customername;

Private String address

Public void set customerid(int customerid)

{

This. Customerid = customerid;

}

Public int getcustomerid( )

{

Return customerid;

}

……

…….

}

2) **MAPPING FILE:-** <hibernate-mapping>

<class name=”customer” table=”customer”>

<id name=” customerid” colume=”custid”>

<propertyname = custname” column= “ customer”>

This file is an heart of an hibernate application

Every ORM tool needs this mapping.

Mapping is a mechanism of placing an object properties into columns of a table.

A mapping can be given to an ORM tool either in the form of an xml (or) in the form of annotations

The mapping file contains mapping from a pojo class name for a table name and pojo class variable names to table columns names.

While columns an hibernate application , we can construct one (or) more mapping files.

SYNTAX:-

<anyname>.hbm.xml

<! Hibernate – mapping>

<class name = “pojo classname “ table =” tablename”>

<id name =” Variable name “ column ”column name” type = javatype/hibernate type”>

<generator class =” generator classname”/>

</id>

<property name =”variable name “ column =” columnname type=” javatype/hibernate>

<property name = “variablename “ column =”columnname” type =”java type/hibernate”/>

……

……

</class>

</hibernate .mapping>

While constructing the mapping file, under every <class>element ; id element; is mandatory.

Is element is used to tep primary key mapping of the holde.

Even through a table does not have any p.k but in the mapping file id element is complusary

In this case any column of the table can be mapped as id

Property element represent non-primary key mapping.

While configuring id , we need to specify a generator class name it inform of hibernate how the primary key value for the object is going to be generate

In single mapping file if u want to write more than once and we need write the class for more than once.

EXAMPLE: <Hibernate-maping>

<class name: “table”>

……

……..

</class>

<class name “ “table”>

……..

…….

</class>

</hibernate.mapping>

Configuration file:-

1) Connection properties.
2) Hibernate properties.
3) Mapp file names.

This is the file loaded into hibernate application when working with hibernate this configuration file contain 3 types of information

1)connection properties.

2)hibernate properties.

3) mapping files.


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com

We need create one configuration file for each db separately

Suppouse if we want to connect with two dbases at a time willing hibernate then we need to create two configuration files.

In hibernate the configuration can be provided by creating either properties file (or) an xml file.

In hibernate 1.x and 2.x ,configuration file is in the form of a resource bundle(properties file) but in hibernate 3.x, the resource bundle is replace with an xml file, but still in hibernate 3.x w can use resource bundle also.

Eventhough we use annotations, but still configuration file is required

Syntax:

<anynam>.cfg.ml

<!..DTD/Schem is must..>

<hibernate-configuration>

<session-factory>

<!—connection properties→

<property name “hibernate.connection.driver-class”>

Driverclassname </properly>

<properly name=”hibernate.connection.url</properly>

<properly name=”hibernate.connection.username”>username</prop>

<properly name=”hibernate.connection.password”>password</properly>

<!—hibernate properties→

<properly name =”hibernate.show_sql>true/false</properly>

<properly name =” hibernate.dialect”>dialect classname</properly>

<properly name=”hibernate.hbmaddl.auto”>create/update</properly>

<!.. mapping file…>

<mapping resource =”hbm filename”/>

……..

……..

</session –factory>

</hibernate –configuration>

While constructing this configuration file, we need to provide either connection properties (or) JNDI properties in order to get a connection with the database.

In general we write configure file name is hibernate cfg.xml but is not mandatory to write the configuration file name as hibernate.cfg.xml instated we can put any name

NOTE: hibernate is java based middleware, for transferring the data in the form of objects b/w a java application and a Dbase server

STEPS TO USE HIBERNATE IN A JAVA APPLICATION:

1) Hibernate can be used in a java application, that is going to run either with in a server (or) out side a server, but we need to follow the standard steps.
2) In order to work with hibernate we do not required any server software mandatory but we need hibernate s/w (jar files of hibernate)

Step:-import the hibernat API.

Import org.hibernat.*;

Import org.hibernate.cfg.*;

Step:-create an object of configuration class

Configuration is a class given in org. hibernate.cfg.x;

Package.

We need to call configure() by passing configuration xfl file as parameter on configuration class object.

Whenever configuration file is loaded into a java application then we can say that hibernate environment is started this step is also called as boot strapping hibernate into a java application.

Configuration conf=new configuration ();

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

Step3:- create an high level of object called sessionFactory

sessionFactory is an interface and sessionFactory.xml is an implemented class and both are given in org. hibernate.* package

in order to get an object of sessionFactory, we need to call a method buildSessionFactory () given by configuration class

syntax:- SessionFactory Factory= conf.buildSessionFactory ();

STEP:- Create an object of session

Session is an interface and Session.impl is an implementation class, both are given in org. hibernate package.

Whenever a session is opened then internally a dbase connection is opened

In order to get a session (or) open a session we need to called openSession ( ) given SessionFactory it means SessionFactory produces a Session (open conn)

Syntax:- Session session = factory . open Session ();

Step:-5) create a logical transaction

While working with insert ,update, delete operation from an hibernate application on to the dbase. Then hibernate needs a logical transaction.

If we are loading (select command) an object from the dbase we do not required any logical transaction in hibernate.

In ordr to begin a logical transaction in hibernate then we need to call a method beginTransaction () givn by Session interface.

SYNTAX: Transaction tx = Session .beginTransaction ()

Step 6:- use the methods given by session interface for too and from transferring the data in the form of object

Session .save (obj)

Session. Delete (obj);

…….etc

Step 7:- if any Exception occurs then hibernate will automatically Rollback (cancel) on the database.

As a programmer after the operation are completed on the dbase .we need to commit the transaction .

If no exception is occur during operations then the transaction will be committed.

SYNTAX : tx. Commit ()

STEP 8:- close the Session

Whenever a Session is closed the internally the dbase connection is also closed.

SYNTAX:- Session.close ()

STEP 9:- close the sessionFactory

Syntax→ Factory.close ()

HIBERNATE INITILIZATION:-

Working with a Framework s/w is nothing but, working with (or) adding the jar file provided by that java application.

Each framework s/w is not an installable s/w.

It means we do not containe any set up file in the s/w

When we download a framework s/w , w will get a zip file and we need to on zip it, to get the jar file required.

All Framework s/w we follow some common principal

1) the Frame work s/w will be in the form of a set of jar files, where one jar file acts as main jar file and remaining jar file, will acts as dependent jar files.

2) each framework s/w application contains at least one configuration file but multip[le configuration files are also allowed.

In order to setup the Fw environment into a java application, the configuration file is the first one to be loaded into a java application.

www.sourceforge.net/projects/hibernat/files/hibernates

LIST OF JAR FILES REQUIRED:-

1. Antlr-2.7.6.jar
2. Asm.jar
3. Cglib-2.1.3.jar
4. Commons-collections-2.1.1.jar
5. Commons-logging-1.0.4.jar
6. Domuj-1.6.1.jar
7.

8. Hibernates.jar(main jar files)
9. Jta.jar
10. Ehcache.jar
11. Log4j-1.2.11.jar
12. Oidberu.jar

1)the following app is to java an object of product class in database.

Hibernat app

Product.java

Product.hbm.xml

Hibernate.cfg.xml

clientForSave.java this is standlone application.net for web application

```
// product.java (pojo)

Public class product

{

Private int productid;

Private string productnam;

Private double price;

Public void setproductid (int productid)

{

This product id = productid;

}

Public int getproductid()

{

Return productid;

}

Public void setproductname(string productname)

{

This.productname = productname;

}

Public string getproductname()

{

Return productname;
```

JAVA Means DURGASOFT

}

Public void setprice (double price)

}

This.price = price;

}

Public double get price ()

}

Return price;

}

}

PRODUCT.HBM.XML:-

<! DOCTYPE------

<! Product.html.xml..>

<hibernate – mapping >

<class name = " product" tabl = " products">

<kid name = " product" column = " pid">

Generator class = " assigned "/>

</id>

<property name = " productname column = " pname'/>

<property name =" price"/>

</ hibernate .mapping>

HIBERNATE.CFG.XML :-

<doc type hibernate – configuration

…

..

<!—hibernate.cfg.xml→

<hibernate- configuration>

<session – factory >

<! ..connection properties..>

<property name= " hibernate.connection.driver.class">

Oracle.jdbc.driver.oracledriver </ property>

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

<property name = " hibernate. Connection username" >scott</ pro>

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

< ! .. hibernate pv apelies..>

<property name = " hibernate. Dialect " > org. hibernate. Dialect- oracleDialect </ property>

< ! .. mapping file..>

< mapping resource " product.hbm.xml'/>

</session .factory>

</hibernate- configuration >

CLIENT FOR SAVE.JAVA:- This is our original java programe this application is doing to stove the object in db.fot this using hibernate concept and save the object in dbase.

Inport org.hibernate.*;

Import org.hibernate.cfg.*;

Class clint forsave

{

//step:

```
Configuration conf = new configuration ();

Con.configuration (“ hibernate.cfg.xml’);

//step 3:

sessionFactory factory = conf.build Session Factory ();

//step 4:

Session session = factory .open Session ( )

Product p = new product ();

p. setproduct id (111);

p.set product name (“Samsung”);

p.set price (1000);

// step 5 :

Transaction tx = session.beginTransaction ();

//Step6:

Session. Save (p);

//Step 7:

Tx. Commit ();

//step 8

Session .clase ( );

Sop (“object saved successfully”);

// step 9

Factory. Clase ();

}
```

JAVA Means DURGASOFT

}

In the example mapping file , for the property price, we have not included column attribute the reason is, the property name and column name both are one and same.

In the above mapping file, for <id>element , we used a sub element called generator, with class assignd it's meaning is the programmer manually insert to the value for the product id property. Hibernate does not generate any id value

If w do not provide a generate class for the < id> element then hibernate by default assumes the generator class as assigned only.

<id name = " product id " =column = "p id">

<generator class = " assigned "/>

</id>

Is equal to

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

In the above client application, we used save () given by session interface. Internally hibernate generate an insert statement and for all statements, hibernate uses preparedstatement of jdbc internally

IMPORTANT POINTS:-

If the configuration file of hibernate is hibernate.cfg.xml then it is not mandatory to call configure () by passing hibernate.cfg.xml as a parameter. Instated, we can directly all configure () with out any parameter.

If the configuration file name is hibernate.cfg.xml then following two statements are one and same.

1)load (class arg, id in the form of an object)

2) get (class arg, id in the form of an object)

By using the above two methods, it is possibl is select load only one object at a time from the dbase .becaz we are passing an id and id is unique.

If we want to load a product object, whose id is then the following statement is required.

Object o = session.load (product. Class, nw integer of (111));

When we compile a java programe, automatically java compiler acts a public static property called class of type class.

When we want a class object then we can all that static class property of the class

BEFORE COMPILING:-

Public class student

{

Private int student.id;

Private string studentname;

Private int marks;

…….

}

- Java c student.java

After compiling

Public class student

{

Public static class class;

Private int studentid;

Private string studentname;

Setters and getter

..

}

Class e = student. Class;

Q)what is diff b/w load() and get() in hibernate?

1) load() throws an exception called objectNotFoundException if the given id is not found in the db.where as get()

RETURNS NULL BUT NOT AN EXCEPTION:-

2)load() does not load an object from db immediately it waits until that object is going to use and then it loads the object at the time this is called log y loading where as get() does not wait for the usage, for instated it immediately loads from the db this is known as early loading.

2) the following example is for loading an object from the database?

R click on project→ properties

\\ select client

Import org. hibernate. X;

Import org. hibernate.cfg.*;

P class selectclient

{

p.s.v main (String…orgs)

{

```
Configuration conf = new configuration ();

Conf. configure ();

sessionFactory factory = conf. buildSessionFactory();

Session Session = Factory. Open Session ();

Object o= session.load ( product. Class, new integer (111));

Product p = (product) o;

Sop ( p. getPrice ());

Session. Close ();

Factory. Close ()

} }
```

NOTE:-in the above client application we have not started any logical transaction of hibernate becaz loading an object from a db does not belong to any transactional operation.

Instated of creating configuration object and SessionFactory separately, we can combin into a single stmt like following.

SessionFactory Factory = new configuration (). Configure (). Build SessionFactory ();

PROCEDURE TO CREATE AN HIBERNATE APPLICATION IN ECLIPSE :-

STEP 1:- start Eclipse → and enter some workspace name → ok

STEP 2:- click on file menu→ new→ project→ select java-project→ next→ enter projects name (example)→ finish.

STEP 3:- RT click an projctname → new→ class→ entername→finish

// student. Java

P class student

{

Private int studentid;

Private string StudentName;

Private int marks

Setters & getters

}

STEP 4 :- RT click on projectname → new→ file→ enter filename (student.hbm.xml)→ Finish.

Copy doctype

<hibernate- mapping>

< class name = " student" lable =" students">

< id name =" studentid" column =" sid">

<generator class = " assigned"/>

</id>

<property name = " student name" column= "sname length = "2"/>

<property name = " marks"/>

</class>

</hibernate – mapping>

STEP 5:- RT click on project→ new→ file→ enter filename

(hibernate.cfg.xml)→ finish

Hibernate code copy and paste

Hibernate properties.

<hibernate – configuration >

< Session – Factory>

< !.. connection properties..>

…

..

<!- connection properties..>

..

..

<property name = " hibernate. Hbmaddl.auto"> create </ property>

< ! mapping resource = " student hbm.xml"/>

<! Hibernate – configuration>

STEP 6:- RT click on projectname→ properties→ select java buildpath at leftside→ selectbrowes tab→ click and externalize button→ select jar files required → open this step is equal to close path setting on console)

STEP 7 :- RT click on projectname→ new→ closs→ enterclass-name (insert client)→ finish

//insert client :-

Import org.hibernate.*;

Import org.hibernate.cfg.*;

```
P class insertclient
{
Static public void main ( string[] arg1)
{
sessionFactory factory = new configuration() . configure()
buildsessionFactory ();
session session = Factory.open session ();
student s = new studnt ();
s.setstudentid( 101);
s.setstudentName (“ abcd”);
s.setmarks (u55);
Transaction tx = session. beginTransaction();
Session.sav (s);
Fx.commit ();
Session close ();
Factory .close ();
Sop (“ object saved”);
} }
```

STEP 8:- to run the client application , R T click on the client java file→ run as→ java application

IMPORTANT POINTS:- 1) in the above example, in hbm.xml file, for studentname property w have added an additional attribute called length because, while creating table by hibernate, it by default uses 255 length for string types. In order to modify that site, we use length attribute.

JAVA Means DURGASOFT

2 In the above hibernate configuration file, we have added an additional hibernate property called hbml ddl.out" its value are
   1. Create 2.update 3.create-drop.
3) If we use hbm.ddl. auto property value as create then hibernate will create a new table by dropping an existing table always and then insert a record into the tabl.
4) When first tim executing the insert operation on the db then the create value will be useful but executing the next operation, we need to put hbm.ddl. auto property in commands, otherwise hibernate will drop existing table and create a new table once againe

5) If we use hbm2.ddl.auto property as update then hibernate first verifies whether the table exist in the db not if exist then hibernate does not create a new table by dropping existing table instated hibernate uses the same table.
6) Hibernate creates a new table,. If it is not available in the db, for the property value updats.
7) If we use hbm2ddl.auto property value as create-drop then hibernate creates a new table, when sessionFactory is closed explicitly then hibernate drops, the table. It means drop value works, if we use factory.close () statement in our application.

UPDATING AN OBJECT:- in hibernate if we want to update an object, ie already persisted in the dbase then we have the following two approaches.

1) Load an object from the database and modify it's values. Now hibernate automatically modifies the valus on to dbase also, whenever a transaction is committed.

2) If we want modify an object then we can create a new object with same id and we can all update() given by session interface

APPROACH 1:- whenever an object to is loaded from the dbase then hibernate stores the loaded object in a coach memory maintained by session.

ii) once an object is loaded, if we do any modification on that object by calling it's setters methods then these modifications are stored in the object maintained by cache memeory.

iii) If we modified the loaded object for multiple times then also the modifications will be stored in the object maintains by the cache memory.

iv) Whenever we issue commit operation then hibernate verifies whether any changes are there b/w the object stored in cache memory and the object stored in dbase. If changes exist then hibernate automatically updates the dbase also by generating an update operation.

v) Hibernate automatically maintains b/w the cache memory objects and dbase tables row

FOR EXAMPLE :- object o = session . load (Product.class, new Interger (111) product p = (product) o;

Transaction tx = Sssion. Begin Transaction ();

p.setprice (1500);

p. setprice(2500);

fx. Commit ();

APPROACH 1:- in this approach we need load object from the bd we will create new object and we will assigne same id number and we will call update method explicitly in order to make changes on the object that is stored in the dbase.

Ex:- Product p = new product ()

p.set product id (111);

Transaction fx= session.begin Transaction ()

Session.update (p);

Fx.commit ();

3) Import org.hibernate. *;

Import org. hibernate.cfg.*;

Public class updateclient

{

Main ()

{

SessionFactory factory = new configuration () .configure () build SessionFactory ()

Session session = Factory.open Session ();

Object o = session. Load (product.class, new integer (111))

Product p = (product)o;

JAVA Means DURGASOFT

Transaction fx= session begin Transaction ();

p. set price (1200);

p.set price (2500);

fx.commit ()

session. Clos ()

factory. Close ();

} }

In the above app we have the product of price if a time, but hibernate generate only a single updation operation only and the object stored in dbase.

Deleting an object :-if we want delete an object from the dbase then load that object from the dbase and call delete() given by session interface.

Ex: object o = session. Load (product.class, new integer(111));

Product p = (product)o;

Transaction fx= session .begin transaction ();

Session.delete (p);

Fx. Commit ();

JAVA Means DURGASOFT

VERSIONING AN OBJECT:-1) once an object is save in a dbase, we can modify that object for any no. of times.

2 .If we want to know that how many number of times an object is modifies then w need to apply this versioning concept.

3 whenever we use versioning then hibernate insert version number as zero whenever an object is saved. Later hibernate increments that version number by one automatically whenever a modification is done on that object.

4 In order to use this versioning concept, we need the following two changes in our application.

1) Create a property (Variable) of type in the pojo class.
2) In hibernate mapping file, add an element called <version>

Example:- p c student

{

Private int studentid;

String studentname ;

Private int mark;

Private int v;

== settr and getters

}

JAVA Means DURGASOFT

STUDENT.HBM.XML

<h-m>

<class.>

<id->

<gen,>

</id>

<vrsion name "u" colum= " version">

</p.>

</p->

</h-m>

In the hibernate mapping file <version> must be immediately after id element.

1) **STUDENT . JAVA:-**

Public class student

{

Private int studentid;

Private string studentnam;

Private int marks;

JAVA Means DURGASOFT

Private int v;

Public int getv( );

Return v;

}

Public void setV (int V)

{

This. V= V;

}

STUDENT. HBM.XML

<doc type..

<hibernate – mapping>

<class name = " student" tabl= " students>

<id name = " student id " column = " sid">

<generator class =" assigned"/>

</id>

<version name = "V" column = " version"/>

JAVA Means DURGA

<property name = " student" column= "sname" length= "12"/>

<property name = marks"/>

</class>

</hibernate>

1)**INSERT CLIENT JAVA**

Import

Import

P class insert client

{

Main ()

{

JAVA Means DURGASOFT


```
sessionFactory factory = new configuration. Configuration ()

session session = factory. Open session ();

student s= new student ();

s.set student id (101);

s.set student name ("sathya "23");

s.set marks (755);

tansaction h = session. Begin transaction ();

session.save (s);

fx. Commit ();

session close ()

factory.close ()

sop ('object is saved")

}    }
```

JAVA Means DURGASOFT

After modifying the object :-

// update client

i.e s.set marks (855);

fx. Commit ();

session. Close ();

factory .close ();

sop ("object is saved")

} }

WORKING WITH TIMSTAMP:-while saving or updating on object, if we want record the current system date and time in the database then we need to us time stamp

If we use time stop the hibernate automatically records stores the current system date and time information in to the database.

Timestamp is a class given in java. Sql package

If we want to use this timestamp feature of hibernate then we need to do the following two changes.

In the pojo class take a property of type timestamp along with th other properties of the class and generate setter and getter methods for it

JAVA Means DURGASOFT

In the hbm.xml file, configure < timestamp> element and it must be configure immediately of < id>

For ex:-

P c student

{

Private int student id;

Private string studentname;

Private int marksl

Private timestamp ts;

--

--

}

Student.hbm.xml

<h- m>

<class ---→

< id --->

<generator---->.

</id->

<timestamp name = " ts" columm= "stamp"/>

</p--->

</p--->

</class>

</h—m>

While hibernate creating the table, hibernate uses the data type of timestamp column as date so in the dbase only date is saved but not time

JAVA Means DURGASOFT

If we want record both date and time then we need to explicitly alter the t stamp column type as timestamp. Otherwise a programmer can explicitly

Create the table.

We are creating pojo class object.

We are assinge some values of that object .

We are saving that object in db.

Student s= new student.

s.set student (101);

s.setStudent name (“abc”);

s.set marks (500);

session. Save (s);

session. Close ();

JAVA Means DURGA

LIFE CYCL OF A POJO OBJECT IN HIBERNATE:-

1) Transient state 2) persistent state 3) detached state

1)**TRANSINT STATE:-** whenever an object of a pojo class is created then it will be in transient state

2) when an object is in transient state it does not represent any row of the database.

3) when an object is in transient state the it does not have any association with the session object.

If we modify the data of a pojo class object when it is in transint state then it does not effect on the dbase table.

Student s = null

Object o = session .load (student. Class, new integer (101);

S = (student) o;

Session .close ()

s.set marks (555);

5) before a pojo class object enters into a session then it will be in transient state.

6) if we want to convert a transient object into persistent state then we have the following two approachs 1) by saving that object.

By loading that object from the database.

8) If we want save an object in the data base then we need to call any one of the following 3 methods. I) save ii) persist() iii) save or update ()
9) If w want to load an object from the database then we need to call eithr load () or get ()

student s= null ; -- transient

object o = sssion .get (student .class, new integer (101);

s= (student) o;

transaction fx = session. Begin ()

session. Delete (s);

f. commit ()

NOTE :- PERSISTENT STATE:- 1) when an object is in persistent state then it rep one row of underlying dbase.

When an object is in persistent state then it is associate with the session.

2) If we make any changes on the persistent object then the change are automatically effected on the database also.
3) From persistent state, we can move an object to either detached state or againe back to transient state

JAVA Means DURGASOFT

4) If we want move an object from persistent state into detached state, we need to either close that session or we need clear of that cache memory of the session or we need to remove that particular object from the coach memory of the session.
5) If we want move an object from persistent state to transient state then we need delete from the object from the dbase.

DETACHED STATE:- 1) when an object is coming out of a sssion (or) when an object living its association with the session then the object enters into detached state.

2) If an object is detached state then if we make any changes on that object then these changes are not effected on the database.
3) It is possible to convert an object from detached state into persistent state back againe by calling update () merge () or save or update ()
4) If the preview session is already closed then we need to open a new session in order to convert from detached state back is persistent state.

// states client. Java :- import org. hiber. *;

Import org. hiber .cfg. *;

P class statesclient

{

Main (string arg[])

{

Session Factory factory = new configuration (). Configure buildsession factory ();

Session session .factory .open session ();

Student s = null // transfixed

Object o = session .get (student. Class , new interger (101));

S = (student) o ; // persistent state
session 1. Close ()

s. set marks (999); // detached

state

session session = factory. Open sssion ();

Transaction fx = session2. Begin Transaction ();

Session 2. Update (s); // persistent state

Fx. Commit ();

Session. Close ();

Factory.Close();

} }

q) what is the diff b/w a primitive type and wrapper type in a pojo class of hibernate ?

a) if we use primitive type and if we are not interested to set the value for it then at the time of saving an object in database the value zero is inserted in that column.

Because zero is the value , so there is a chance of miss understanding the data

In order to overcome the above problem, we can take a wrapper type in place of a primitive type so if we do not set the value for wrapper type in dbase null values inserted into column there will be no miss understanding data.

While creating pojo class in hibernate , it is recommend to use wrapper types rather than primitive type.

Student s1= null;

Object o = session. Get (student . class. New integer (101);

S1= (student) o;

Session. Clos ();

S1.setmarks (850);

JAVA Means DURGASOFT

Session session = factory .open session();

Object o= session. Get (student. Class, new integer (101);

Student s2= (student) o1;

Trans fx = session2. Begin trans();

Session 2. Update (s2);

Fx. Commit ();

Java is 5star length

q) what Is diff b/w update () and merge () in the java?

1) if we want to convert a detached object into persistent state againe then w call update ()

2) in this conversion , if that session is already having an object with the same id then hibernate throws exception because in hibernate , a session can maintaine only one object with the same.

3) id instated of calling update () we need to call merge () in this case hibernate copies the state changes nothing but data from detached object into the persistent object and writes the changes on to the data base.

4) merge () will merge combine two object into one object , if the two objects are rep the same id.

5) update () and the merge() bothe are used for converting or moving an object from detachd state into persistent state againe.

Ex:- student s1= null;

Object o= session. Get (studentclass. Nw interger (101))

S1= (student)o;

Session. Close ();

S1.setmarks (850); \\ detached

Session sssion2= factory. Open Session();

Student s2= null;

Object o1= session2.get(studentclass, new integer (101));

S2= (student)o1;

Transaction fx= sssion2.begin Transaction ();

Session2. Update (s1)

Fx. Commit;

Sessions close ()

JAVA Means DURGASOFT

Instated of calling th update () we need to call marge () in order to convert s1 from detached state into persistent state.

Session.merge (s1)

INHERITANCE MAPPING IN HIBERNATE:-

1) Table per class hierarchy
2) Table per sub class hierarchy.
3) Table per concrete class hierarchy

TABLE PER CLASS HIERARCHY :-

In this type of hierarchy, hibernate uses a simple table in the database for storing the base class data and all it's derived classes data in the databse.

In this type of hierarchy hibernate needs an addition column in the table called discriminator column this discriminator column is mandatory, while writing with table per class hierarchy

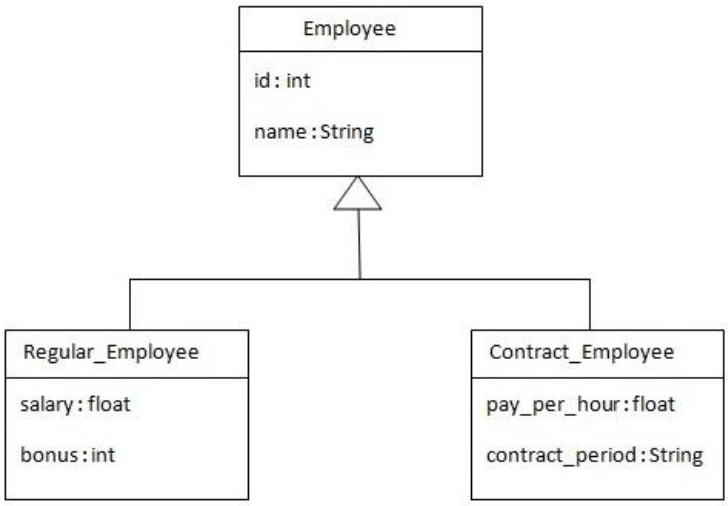
In this hierarchy hibernate uses discriminator column in order to identify white derived class object has inserted a row in the database table it means discriminator column is used for indetification purpose.

In table perclass hierarchy while writing hibernate mapping file, we need to use < discriminator > element and < subclass > element newly in the xml file

By this inheritance strategy, we can map the whole hierarchy by single table only. Here, an extra column (also known as**discriminator column**) is created in the table to identify the class.

LEARN FROM EXPERT & DIAMOND FACULTIES OF AMEERPET...
JAVA MEANS DURGASOFT
INDIA'S NO. 1 SOFTWARE TRAINING INSTITUTE
AN ISO 9001:2008 CERTIFIED DURGA SOFTWARE SOLUTIONS
#202 2nd FLOOR
www.durgasoft.com
040-64512786
+91 9246212143
+91 8096969696

Let's understand the problem first. I want to map the whole hierarchy given below into one table of the database.



FREE TRAINING VIDEOS
YouTube
3000+ VIDEOS
www.youtube.com/durgasoftware

There are three classes in this hierarchy. Employee is the super class for Regular_Employee and Contract_Employee classes. Let's see the mapping file for this hierarchy.

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6. <hibernate-mapping>
7. <class name="com.javatpoint.mypackage.Employee" table="emp121" discr
   iminator-value="emp">
8. <id name="id">
9. <generator class="increment"></generator>
10.       </id>
11.
12.       <discriminator column="type" type="string"></discriminator>
13.       <property name="name"></property>
14.
15.       <subclass name="com.javatpoint.mypackage.Regular_Employee" discr
   iminator-value="reg_emp">
16.       <property name="salary"></property>
17.       <property name="bonus"></property>
18.       </subclass>
19.
20.       <subclass name="com.javatpoint.mypackage.Contract_Employee" disc
   riminator-value="con_emp">
21.       <property name="pay_per_hour"></property>
22.       <property name="contract_duration"></property>
23.       </subclass>
24.
25.       </class>
26.
27.       </hibernate-mapping>
```

In case of table per class hierarchy an discriminator column is added by the hibernate framework that specifies the type of the record. It is mainly used to distinguish the record. To specify this, **discriminator** subelement of class must be specified.

The **subclass** subelement of class, specifies the subclass. In this case, Regular_Employee and Contract_Employee are the subclasses of Employee class.

The table structure for this hierarchy is as shown below:

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
TYPE	VARCHAR2(255)	No	-	-
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 7

Example of Table per class hierarchy

In this example we are creating the three classes and provide mapping of these classes in the employee.hbm.xml file.

1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

File: Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Employee {
4. private int id;
5. private String name;
6.
7. //getters and setters
8. }
```

File: Regular_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Regular_Employee extends Employee{
4. private float salary;
5. private int bonus;
6.
7. //getters and setters
8. }
```

File: Contract_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Contract_Employee extends Employee{
4.     private float pay_per_hour;
5.     private String contract_duration;
6.
7. //getters and setters
8. }
```

---

2) Create the mapping file for Persistent class

The mapping has been discussed above for the hierarchy.

File: employee.hbm.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
```

```
6. <hibernate-mapping>
7. <class name="com.javatpoint.mypackage.Employee" table="emp121" discri
   minator-value="emp">
8. <id name="id">
9. <generator class="increment"></generator>
10.    </id>
11.
12.    <discriminator column="type" type="string"></discriminator>
13.    <property name="name"></property>
14.
15.    <subclass name="com.javatpoint.mypackage.Regular_Employee" discrim
   inator-value="reg_emp">
16.    <property name="salary"></property>
17.    <property name="bonus"></property>
18.    </subclass>
19.
20.    <subclass name="com.javatpoint.mypackage.Contract_Employee" discri
   minator-value="con_emp">
21.    <property name="pay_per_hour"></property>
22.    <property name="contract_duration"></property>
23.    </subclass>
24.
25.    </class>
26.
27.    </hibernate-mapping>
```

3) Add mapping of hbm file in configuration file

Open the hibernate.cgf.xml file, and add an entry of mapping resource like this:

```
1. <mapping resource="employee.hbm.xml"/>
```

Now the configuration file will look like this:

File: hibernate.cfg.xml


```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6. <hibernate-configuration>
7.
8.     <session-factory>
9.         <property name="hbm2ddl.auto">update</property>
10.             <property name="dialect">org.hibernate.dialect.Oracle9Dialect<
   /property>
11.             <property name="connection.url">jdbc:oracle:thin:@localhost:1
   521:xe</property>
12.             <property name="connection.username">system</property>
13.             <property name="connection.password">oracle</property>
14.             <property name="connection.driver_class">oracle.jdbc.driver.Or
   acleDriver</property>
15.          <mapping resource="employee.hbm.xml"/>
16.          </session-factory>
17.
18.      </hibernate-configuration>
```

The hbm2ddl.auto property is defined for creating automatic table in the database.

4) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

File: StoreData.java

```
1. package com.javatpoint.mypackage;
2. import org.hibernate.*;
3. import org.hibernate.cfg.*;
4.
5. public class StoreData {
6. public static void main(String[] args) {
7.     Session session=new Configuration().configure("hibernate.cfg.xml")
8.                     .buildSessionFactory().openSession();
9.
10.          Transaction t=session.beginTransaction();
11.
12.          Employee e1=new Employee();
13.          e1.setName("sonoo");
14.
15.          Regular_Employee e2=new Regular_Employee();
16.          e2.setName("Vivek Kumar");
17.          e2.setSalary(50000);
18.          e2.setBonus(5);
19.
```


```
20.
21.         Contract_Employee e3=new Contract_Employee();
22.         e3.setName("Arjun Kumar");
23.         e3.setPay_per_hour(1000);
24.         e3.setContract_duration("15 hours");
25.
26.         session.persist(e1);
27.         session.persist(e2);
28.         session.persist(e3);
29.
30.         t.commit();
31.         session.close();
32.         System.out.println("success");
33.     }
34.     }
```

Output:

EDIT	ID	TYPE	NAME	SALARY	BONUS	PAY_PER_HOUR	CONTRACT_DURATION
	1	emp	sonoo	-	-	-	-
	2	reg_emp	Vivek Kumar	50000	5	-	-
	3	con_emp	Arjun Kumar	-	-	1000	15 hours
							row(s) 1 - 3 of 3

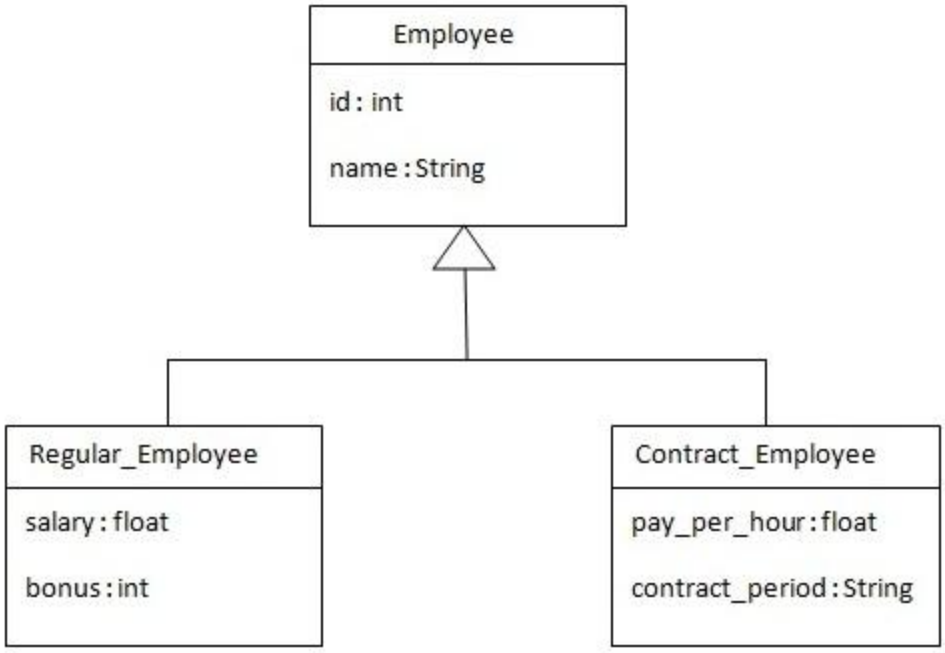
Table Per Subclass Hierarchy

In case of Table Per Subclass, subclass mapped tables are related to parent class mapped table by primary key and foreign key relationship.

The <joined-subclass> element of class is used to map the child class with parent using the primary key and foreign key relation.

In this example, we are going to use hb2ddl.auto property to generate the table automatically. So we don't need to be worried about creating tables in the database.

Let's see the hierarchy of classes that we are going to map.



Let's see how can we map this hierarchy by joined-subclass element:


```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.
4.   "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
5.   "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
6.
7.
8.   <hibernate-mapping>
```

```
9.   <class name="com.javatpoint.mypackage.Employee" table="emp123">
10.         <id name="id">
11.         <generator class="increment"></generator>
12.         </id>
13.
14.         <property name="name"></property>
15.
16.         <joined-
    subclass name="com.javatpoint.mypackage.Regular_Employee" table="rege
    mp123">
17.         <key column="eid"></key>
18.         <property name="salary"></property>
19.         <property name="bonus"></property>
20.         </joined-subclass>
21.
```


```
22.         <joined-
    subclass name="com.javatpoint.mypackage.Contract_Employee" table="con
    temp123">
23.         <key column="eid"></key>
24.         <property name="pay_per_hour"></property>
25.         <property name="contract_duration"></property>
26.         </joined-subclass>
27.
28.         </class>
29.         </hibernate-mapping>
```

In case of table per subclass class, there will be three tables in the database, each representing a particular class.

The joined-subclass subelement of class, specifies the subclass.

The key subelement of joined-subclass is used to generate the foreign key in the subclass mapped table. This foreign key will be associated with the primary key of parent class mapped table.

The table structure for each table will be as follows:

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
EID	NUMBER(10,0)	No	-	1
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 3

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
EID	NUMBER(10,0)	No	-	1
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 3

Example of Table per subclass class

In this example we are creating the three classes and provide mapping of these classes in the employee.hbm.xml file.

1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

File: Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Employee {
4. private int id;
5. private String name;
6.
7. //getters and setters
8. }
```

File: Regular_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Regular_Employee extends Employee{
4. private float salary;
5. private int bonus;
6.
7. //getters and setters
8. }
```

File: Contract_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Contract_Employee extends Employee{
4.     private float pay_per_hour;
5.     private String contract_duration;
```

```
6.
7. //getters and setters
8. }
```

---

2) Create the mapping file for Persistent class

The mapping has been discussed above for the hierarchy.

File: employee.hbm.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.
4.  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
5.  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```


```
6.
7.
8.  <hibernate-mapping>
9.  <class name="com.javatpoint.mypackage.Employee" table="emp123">
10.       <id name="id">
11.       <generator class="increment"></generator>
12.       </id>
13.
14.       <property name="name"></property>
15.
16.       <joined-
subclass name="com.javatpoint.mypackage.Regular_Employee" table="rege
mp123">
```

```
17.         <key column="eid"></key>
18.         <property name="salary"></property>
19.         <property name="bonus"></property>
20.         </joined-subclass>
21.
22.         <joined-
    subclass name="com.javatpoint.mypackage.Contract_Employee" table="con
    temp123">
23.         <key column="eid"></key>
24.         <property name="pay_per_hour"></property>
25.         <property name="contract_duration"></property>
26.         </joined-subclass>
27.
28.         </class>
29.         </hibernate-mapping>
```

3) create configuration file

Open the hibernate.cgf.xml file, and add an entry of mapping resource like this:

```
1. <mapping resource="employee.hbm.xml"/>
```

Now the configuration file will look like this:

File: hibernate.cfg.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6. <hibernate-configuration>
7.
8.     <session-factory>
9.         <property name="hbm2ddl.auto">update</property>
10.             <property name="dialect">org.hibernate.dialect.Oracle9Dialect<
   /property>
11.             <property name="connection.url">jdbc:oracle:thin:@localhost:1
   521:xe</property>
```


```
12.             <property name="connection.username">system</property>
13.             <property name="connection.password">oracle</property>
14.             <property name="connection.driver_class">oracle.jdbc.driver.Or
   acleDriver</property>
15.         <mapping resource="employee.hbm.xml"/>
16.         </session-factory>
17.
18.     </hibernate-configuration>
```

The hbm2ddl.auto property is defined for creating automatic table in the database.

---

4) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

File: StoreData.java

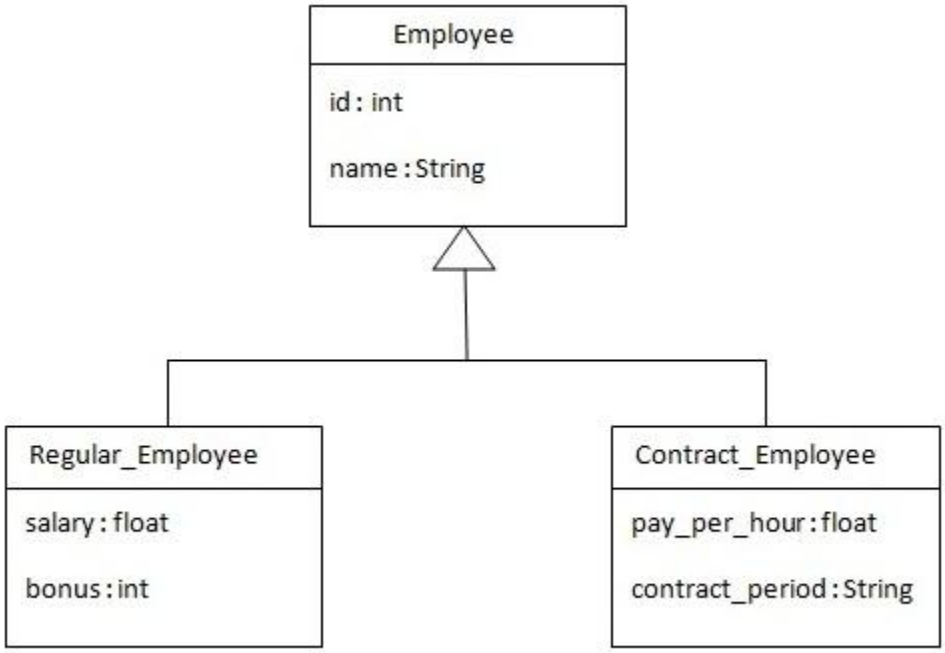
```
1. package com.javatpoint.mypackage;
2.
3. import org.hibernate.*;
4. import org.hibernate.cfg.*;
5.
6. public class StoreData {
7. public static void main(String[] args) {
8.     Session session=new Configuration().configure("hibernate.cfg.xml")
9.                     .buildSessionFactory().openSession();
10.
11.         Transaction t=session.beginTransaction();
12.
13.         Employee e1=new Employee();
14.         e1.setName("sonoo");
15.
16.         Regular_Employee e2=new Regular_Employee();
17.         e2.setName("Vivek Kumar");
18.         e2.setSalary(50000);
19.         e2.setBonus(5);
20.
21.         Contract_Employee e3=new Contract_Employee();
22.         e3.setName("Arjun Kumar");
23.         e3.setPay_per_hour(1000);
24.         e3.setContract_duration("15 hours");
25.
26.         session.persist(e1);
27.         session.persist(e2);
28.         session.persist(e3);
29.
30.         t.commit();
31.         session.close();
32.         System.out.println("success");
33.     }
34.     }
```

Table Per Concrete class Hierarchy

In case of Table Per Concrete class, there will be three tables in the database having no relations to each other. There are two ways to map the table with table per concrete class strategy.

- By union-subclass element
- By Self creating the table for each class

Let's understand what hierarchy we are going to map.



Let's see how can we map this hierarchy by union-subclass element:

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">


5.   <hibernate-mapping>
6.   <class name="com.javatpoint.mypackage.Employee" table="emp122">
7.   <id name="id">
8.   <generator class="increment"></generator>
9.   </id>
10.
11.         <property name="name"></property>
12.
13.         <union-
    subclass name="com.javatpoint.mypackage.Regular_Employee" table="rege
    mp122">
14.         <property name="salary"></property>
15.         <property name="bonus"></property>
16.         </union-subclass>
17.
18.         <union-
    subclass name="com.javatpoint.mypackage.Contract_Employee" table="con
    temp122">
19.         <property name="pay_per_hour"></property>
20.         <property name="contract_duration"></property>
21.         </union-subclass>
22.
23.         </class>
24.
25.         </hibernate-mapping>
```

In case of table per concrete class, there will be three tables in the database, each representing a particular class.

The union-subclass subelement of class, specifies the subclass. It adds the columns of parent table into this table. In other words, it is working as a union.


LEARN FROM EXPERTS ...
COMPLETE JAVA
CORE JAVA, ADV. JAVA, ORACLE, STRUTS, HIBERNATE, SPRING, WEB SERVICES,...
COMPLETE .NET
C#.NET, ASP.NET, SQL SERVER, MVC 5 & WCF
TESTING TOOLS
MANUAL + SELENIUM
ORACLE
D2K
MSBI
SHARE POINT
HADOOP
ANDROID
C, C++, DS, UNIX
CRT & APTITUDE TRAINING
AN ISO 9001:2008 CERTIFIED
DURGA
Software Solutions®
202, 2nd Floor, HUDA Maitrivanam,
Ameerpet, Hyd. Ph: 040-64512786,
9246212143, 8096969696
www.durgasoft.com

The table structure for each table will be as follows:

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 4

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 4

Example of Table per concrete class

In this example we are creating the three classes and provide mapping of these classes in the employee.hbm.xml file.

1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

File: Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Employee {
4. private int id;
5. private String name;
6.
7. //getters and setters
8. }
```

File: Regular_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Regular_Employee extends Employee{
4. private float salary;
5. private int bonus;
6.
7. //getters and setters
8. }
```

File: Contract_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Contract_Employee extends Employee{
4.     private float pay_per_hour;
5.     private String contract_duration;
6.
7. //getters and setters
8. }
```

---

2) Create the mapping file for Persistent class

The mapping has been discussed above for the hierarchy.

File: employee.hbm.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6.
7.   <hibernate-mapping>
8.   <class name="com.javatpoint.mypackage.Employee" table="emp122">
9.   <id name="id">
10.        <generator class="increment"></generator>
11.        </id>
12.
13.        <property name="name"></property>
14.
15.        <union-
   subclass name="com.javatpoint.mypackage.Regular_Employee" table="rege
   mp122">
16.        <property name="salary"></property>
17.        <property name="bonus"></property>
18.        </union-subclass>
19.
20.        <union-
   subclass name="com.javatpoint.mypackage.Contract_Employee" table="con
   temp122">
21.        <property name="pay_per_hour"></property>
22.        <property name="contract_duration"></property>
```

```
23.         </union-subclass>
24.
25.         </class>
26.
27.         </hibernate-mapping>
```

3) Add mapping of hbm file in configuration file

Open the hibernate.cgf.xml file, and add an entry of mapping resource like this:

```
1. <mapping resource="employee.hbm.xml"/>
```

Now the configuration file will look like this:

File: hibernate.cfg.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6. <hibernate-configuration>
```


```
    <session-factory>
        <property name="hbm2ddl.auto">update</property>
        <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
            <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
            <property name="connection.username">system</property>
            <property name="connection.password">oracle</property>
            <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
          <mapping resource="employee.hbm.xml"/>
          </session-factory>

       </hibernate-configuration>
```

The hbm2ddl.auto property is defined for creating automatic table in the database.

---

4) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

File: StoreData.java

```
package com.javatpoint.mypackage;

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

```

```
6. public class StoreData {
7. public static void main(String[] args) {
8.     Session session=new Configuration().configure("hibernate.cfg.xml")
9.                     .buildSessionFactory().openSession();
10.
11.         Transaction t=session.beginTransaction();
12.
13.         Employee e1=new Employee();
14.         e1.setName("sonoo");
15.
```


```
16.         Regular_Employee e2=new Regular_Employee();
17.         e2.setName("Vivek Kumar");
18.         e2.setSalary(50000);
19.         e2.setBonus(5);
20.
21.         Contract_Employee e3=new Contract_Employee();
22.         e3.setName("Arjun Kumar");
23.         e3.setPay_per_hour(1000);
24.         e3.setContract_duration("15 hours");
25.
26.         session.persist(e1);
27.         session.persist(e2);
28.         session.persist(e3);
29.
30.         t.commit();
31.         session.close();
32.         System.out.println("success");
33.     }
34.     }
```

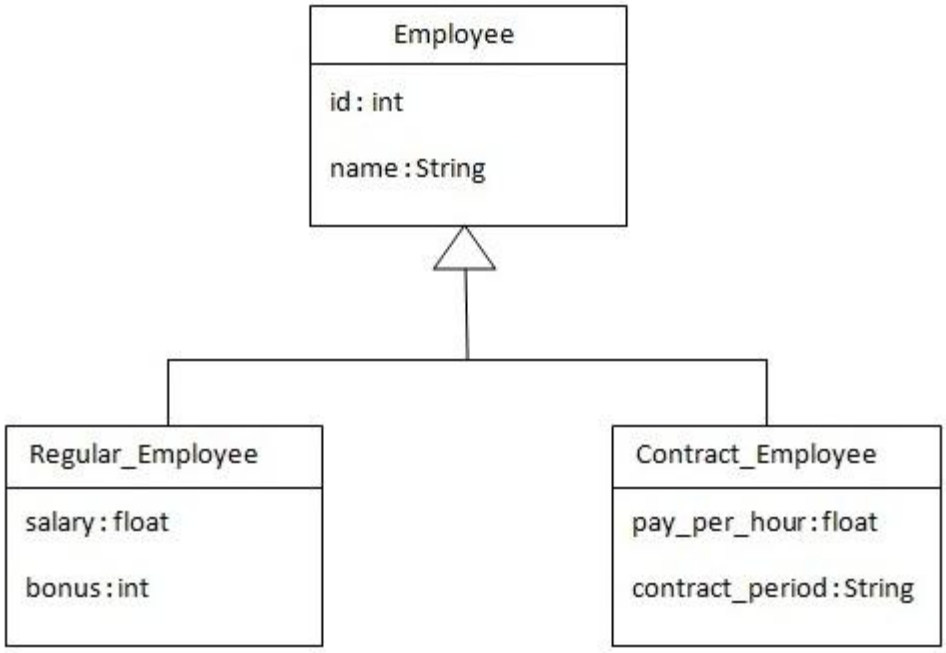
Table Per Concrete class

In case of Table Per Concrete class, there will be three tables in the database having no relations to each other. There are two ways to map the table with table per concrete class strategy.

- By union-subclass element
- By Self creating the table for each class

Let's understand what hierarchy we are going to map.

JAVA Means DURGASOFT



Let's see how can we map this hierarchy by union-subclass element:

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6.   <hibernate-mapping>
7.   <class name="com.javatpoint.mypackage.Employee" table="emp122">
8.   <id name="id">
9.   <generator class="increment"></generator>
10.         </id>
11.
12.         <property name="name"></property>
13.
14.         <union-
    subclass name="com.javatpoint.mypackage.Regular_Employee" table="rege
    mp122">
15.         <property name="salary"></property>
16.         <property name="bonus"></property>
17.         </union-subclass>
18.
```

```
19.         <union-
    subclass name="com.javatpoint.mypackage.Contract_Employee" table="con
    temp122">
20.         <property name="pay_per_hour"></property>
21.         <property name="contract_duration"></property>
22.         </union-subclass>
23.
24.         </class>
25.
26.         </hibernate-mapping>
```

In case of table per concrete class, there will be three tables in the database, each representing a particular class.

The union-subclass subelement of class, specifies the subclass. It adds the columns of parent table into this table. In other words, it is working as a union.

The table structure for each table will be as follows:

Table structure for Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
				1 - 2

Table structure for Regular_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
SALARY	FLOAT	Yes	-	-
BONUS	NUMBER(10,0)	Yes	-	-
				1 - 4

Table structure for Contract_Employee class

Column Name	Data Type	Nullable	Default	Primary Key
ID	NUMBER(10,0)	No	-	1
NAME	VARCHAR2(255)	Yes	-	-
PAY_PER_HOUR	FLOAT	Yes	-	-
CONTRACT_DURATION	VARCHAR2(255)	Yes	-	-
				1 - 4

Example of Table per concrete class

In this example we are creating the three classes and provide mapping of these classes in the employee.hbm.xml file.

1) Create the Persistent classes

You need to create the persistent classes representing the inheritance. Let's create the three classes for the above hierarchy:

File: Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Employee {
4. private int id;
5. private String name;
6.
7. //getters and setters
8. }
```

File: Regular_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Regular_Employee extends Employee{
4. private float salary;
5. private int bonus;
6.
7. //getters and setters
```


```
8. }
```

File: Contract_Employee.java

```
1. package com.javatpoint.mypackage;
2.
3. public class Contract_Employee extends Employee{
4.     private float pay_per_hour;
```

```
5.     private String contract_duration;
6.
7. //getters and setters
8. }
```

2) Create the mapping file for Persistent class

The mapping has been discussed above for the hierarchy.

File: employee.hbm.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6.
7.   <hibernate-mapping>
8.   <class name="com.javatpoint.mypackage.Employee" table="emp122">
9.   <id name="id">
10.         <generator class="increment"></generator>
11.         </id>
12.         <property name="name"></property>
13.
14.         <union-
    subclass name="com.javatpoint.mypackage.Regular_Employee" table="rege
    mp122">
15.         <property name="salary"></property>
16.         <property name="bonus"></property>
17.         </union-subclass>
18.
19.         <union-
    subclass name="com.javatpoint.mypackage.Contract_Employee" table="con
    temp122">
20.         <property name="pay_per_hour"></property>
21.         <property name="contract_duration"></property>
22.         </union-subclass>
23.
24.         </class>
25.
26.         </hibernate-mapping>
```

3) Add mapping of hbm file in configuration file

Open the hibernate.cgf.xml file, and add an entry of mapping resource like this:

```
1. <mapping resource="employee.hbm.xml"/>
```

Now the configuration file will look like this:

File: hibernate.cfg.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6. <hibernate-configuration>
7.
8.     <session-factory>
9.         <property name="hbm2ddl.auto">update</property>
10.             <property name="dialect">org.hibernate.dialect.Oracle9Dialect<
   /property>
11.             <property name="connection.url">jdbc:oracle:thin:@localhost:1
   521:xe</property>
12.             <property name="connection.username">system</property>
13.             <property name="connection.password">oracle</property>
14.             <property name="connection.driver_class">oracle.jdbc.driver.Or
   acleDriver</property>
15.         <mapping resource="employee.hbm.xml"/>
16.         </session-factory>
```

```
17.
18.         </hibernate-configuration>
```

The hbm2ddl.auto property is defined for creating automatic table in the database.

---

4) Create the class that stores the persistent object

In this class, we are simply storing the employee objects in the database.

File: StoreData.java

```
1. package com.javatpoint.mypackage;
2.
3. import org.hibernate.*;
4. import org.hibernate.cfg.*;
5.
6. public class StoreData {
7. public static void main(String[] args) {
8.     Session session=new Configuration().configure("hibernate.cfg.xml")
9.                     .buildSessionFactory().openSession();
10.
11.          Transaction t=session.beginTransaction();
12.
13.          Employee e1=new Employee();
14.          e1.setName("sonoo");
```

```
15.
16.         Regular_Employee e2=new Regular_Employee();
17.         e2.setName("Vivek Kumar");
18.         e2.setSalary(50000);
19.         e2.setBonus(5);

20.         Contract_Employee e3=new Contract_Employee();
21.         e3.setName("Arjun Kumar");
22.         e3.setPay_per_hour(1000);
23.         e3.setContract_duration("15 hours");
24.
25.         session.persist(e1);
26.         session.persist(e2);
27.         session.persist(e3);
28.
29.         t.commit();
30.         session.close();
31.         System.out.println("success");
32.     }
33.     }
```

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

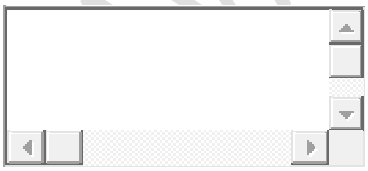
- If the table has a primary key then in the hibernate mapping file we need to configure that column by using <id /> element right..!
- Even though the database table doesn't have any primary key, we must configure one column as id (one primary key is must)
- If the database table has more than one column as primary key then we call it ascomposite 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



packagestr;


```
publicclassProduct implementsjava.io.Serializable{
        privatestaticfinallongserialVersionUID=1L;

        privateintproductId;
        privateStringproName;
        privatedoubleprice;

        publicvoidsetProductId(intproductId)
        {
           this.productId=productId;
        }

        publicintgetProductId()
        {
           returnproductId;
        }
```

JAVA Means DURGASOFT

```
    publicvoidsetProName(StringproName)
    {
       this.proName=proName;
    }
    publicStringgetProName()
    {
       returnproName;
    }
    publicvoidsetPrice(doubleprice)
    {
       this.price=price;
    }
    publicdoublegetPrice()
    {
       returnprice;
    }
}
```

JAVA Means DURGASOFT

hibernate.cfg.xml

```
<?xmlversion='1.0'encoding='UTF-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

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


<hibernate-configuration>

<session-factory>

<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

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

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

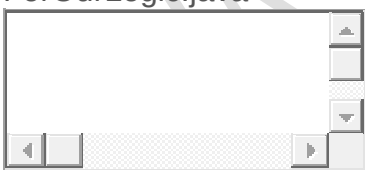
JAVA Means DURGASOFT


```
<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



```
packagestr;
```

JAVA Means DURGASOFT


```
importorg.hibernate.*;

importorg.hibernate.cfg.*;


publicclassForOurLogic{


publicstaticvoidmain(String[]args)

{

Configuration cfg=newConfiguration();

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


SessionFactory factory=cfg.buildSessionFactory();

                    Session session=factory.openSession();


                        Productp=newProduct();


                        p.setProductId(101);

                       p.setProName("iPhone");
```

JAVA Means DURGASOFT

```
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.Serializable` interface, this is the first time am writing this implementation for the pojo class right…! we will see the reason why we use this serializable interface later.
- But remember, if we want to use the composite primary keys we must implement our pojo class with Serializable interface
- hibernate.cfg.xml is normal as previous programs, something like hello world program

JAVA Means DURGASOFT

- 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)

Hibernate Query Language(HQL)

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries which in turns perform action on database.

Although you can use SQL statements directly with Hibernate using Native SQL but I would recommend to use HQL whenever possible to avoid database portability hassles, and to take advantage of Hibernate's SQL generation and caching strategies.

Keywords like SELECT , FROM and WHERE etc. are not case sensitive but properties like table and column names are case sensitive in HQL.

FROM Clause

You will use FROM clause if you want to load a complete persistent objects into memory. Following is the simple syntax of using FROM clause:

```
String hql ="FROM Employee";
Query query =session.createQuery(hql);
List results =query.list();
```

If you need to fully qualify a class name in HQL, just specify the package and class name as follows:


```
String hql ="FROM com.hibernatebook.criteria.Employee";
Query query =session.createQuery(hql);
List results =query.list();
```

AS Clause

The AS clause can be used to assign aliases to the classes in your HQL queries, specially when you have long queries. For instance, our previous simple example would be the following:

```
String hql ="FROM Employee AS E";
Query query =session.createQuery(hql);
List results =query.list();
```

The AS keyword is optional and you can also specify the alias directly after the class name, as follows:

```
String hql ="FROM Employee E";
Query query =session.createQuery(hql);
List results =query.list();
```

SELECT Clause

The SELECT clause provides more control over the result set than the from clause. If you want to obtain few properties of objects instead of the complete object, use the SELECT clause. Following is the simple syntax of using SELECT clause to get just first_name field of the Employee object:

```
String hql ="SELECT E.firstName FROM Employee E";
Query query =session.createQuery(hql);
```

```
List results =query.list();
```

It is notable here that Employee.firstName is a property of Employee object rather than a field of the EMPLOYEE table.

WHERE Clause

If you want to narrow the specific objects that are returned from storage, you use the WHERE clause. Following is the simple syntax of using WHERE clause:

```
String hql ="FROM Employee E WHERE E.id = 10";
Query query =session.createQuery(hql);
List results =query.list();
```

ORDER BY Clause

To sort your HQL query's results, you will need to use the ORDER BY clause. You can order the results by any property on the objects in the result set either ascending (ASC) or descending (DESC). Following is the simple syntax of using ORDER BY clause:

```
String hql ="FROM Employee E WHERE E.id > 10 ORDER BY E.salary
DESC";
Query query =session.createQuery(hql);
List results =query.list();
```

If you wanted to sort by more than one property, you would just add the additional properties to the end of the order by clause, separated by commas as follows:

```
String hql ="FROM Employee E WHERE E.id > 10 "+
"ORDER BY E.firstName DESC, E.salary DESC ";
Query query =session.createQuery(hql);
List results =query.list();
```

GROUP BY Clause

This clause lets Hibernate pull information from the database and group it based on a value of an attribute and, typically, use the result to include an aggregate value. Following is the simple syntax of using GROUP BY clause:

```
String hql ="SELECT SUM(E.salary), E.firtName FROM Employee E "+
"GROUP BY E.firstName";
Query query =session.createQuery(hql);
List results =query.list();
```

Using Named Paramters

Hibernate supports named parameters in its HQL queries. This makes writing HQL queries that accept input from the user easy and you do not have to defend against SQL injection attacks. Following is the simple syntax of using named parameters:

```
String hql ="FROM Employee E WHERE E.id = :employee_id";
Query query =session.createQuery(hql);
query.setParameter("employee_id",10);
List results =query.list();
```

UPDATE Clause

Bulk updates are new to HQL with Hibernate 3, and deletes work differently in Hibernate 3 than they did in Hibernate 2. The Query interface now contains a method called executeUpdate() for executing HQL UPDATE or DELETE statements.

The UPDATE clause can be used to update one or more properties of an one or more objects. Following is the simple syntax of using UPDATE clause:

```
String hql ="UPDATE Employee set salary = :salary "+
"WHERE id = :employee_id";
Query query =session.createQuery(hql);
query.setParameter("salary",1000);
query.setParameter("employee_id",10);
int result = query.executeUpdate();
System.out.println("Rows affected: "+ result);
```

DELETE Clause

The DELETE clause can be used to delete one or more objects. Following is the simple syntax of using DELETE clause:

```
String hql ="DELETE FROM Employee "+
"WHERE id = :employee_id";
Query query =session.createQuery(hql);
query.setParameter("employee_id",10);
int result = query.executeUpdate();
System.out.println("Rows affected: "+ result);
```

INSERT Clause

HQL supports INSERT INTO clause only where records can be inserted from one object to another object. Following is the simple syntax of using INSERT INTO clause:

```
String hql ="INSERT INTO Employee(firstName, lastName, salary)"+
"SELECT firstName, lastName, salary FROM old_employee";
Query query =session.createQuery(hql);
int result = query.executeUpdate();
System.out.println("Rows affected: "+ result);
```

Criteria API

HCQL (Hibernate Criteria Query Language)

The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria. The Criteria interface provides methods to apply criteria such as retreiving all the records of table whose salary is greater than 50000 etc.

Advantage of HCQL

The HCQL provides methods to add criteria, so it is easy for the java programmer to add criteria. The java programmer is able to add many criteria on a query.

Criteria Interface

The Criteria interface provides many methods to specify criteria. The object of Criteria can be obtained by calling thecreateCriteria() method of Session interface.

Syntax of createCriteria() method of Session interface

1. public Criteria createCriteria(Class c)

The commonly used methods of Criteria interface are as follows:

1. public Criteria add(Criterion c) is used to add restrictions.
2. public Criteria addOrder(Order o) specifies ordering.
3. public Criteria setFirstResult(int firstResult) specifies the first number of record to be retreived.
4. public Criteria setMaxResult(int totalResult) specifies the total number of records to be retreived.
5. public List list() returns list containing object.
6. public Criteria setProjection(Projection projection) specifies the projection.

Restrictions class

Restrictions class provides methods that can be used as Criterion. The commonly used methods of Restrictions class are as follows:

1. public static SimpleExpression lt(String propertyName,Object value) sets the less than constraint to the given property.
2. public static SimpleExpression le(String propertyName,Object value) sets the less than or equalconstraint to the given property.
3. public static SimpleExpression gt(String propertyName,Object value) sets the greater than constraint to the given property.
4. public static SimpleExpression ge(String propertyName,Object value) sets the greater than or equalthan constraint to the given property.

5. public static SimpleExpression ne(String propertyName,Object value) sets the not equal constraint to the given property.
6. public static SimpleExpression eq(String propertyName,Object value) sets the equal constraint to the given property.
7. public static Criterion between(String propertyName, Object low, Object high) sets the betweenconstraint.
8. public static SimpleExpression like(String propertyName, Object value) sets the like constraint to the given property.

Order class

The Order class represents an order. The commonly used methods of Restrictions class are as follows:

1. public static Order asc(String propertyName) applies the ascending order on the basis of given property.
2. public static Order desc(String propertyName) applies the descending order on the basis of given property.

Examples of Hibernate Criteria Query Language

There are given a lot of examples of HCQL.

Example of HCQL to get all the records

```
1. Crietria c=session.createCriteria(Emp.class);//passing Class class argument

2. List list=c.list();
```

Example of HCQL to get the 10th to 20th record

```
1. Crietria c=session.createCriteria(Emp.class);
2. c.setFirstResult(10);
3. c.setMaxResult(20);
4. List list=c.list();
```

Example of HCQL to get the records whose salary is greater than 10000

```
1. Crietria c=session.createCriteria(Emp.class);
2. c.add(Restrictions.gt("salary",10000));//salary is the propertyname
3. List list=c.list();
```

Example of HCQL to get the records in ascending order on the basis of salary

```
1. Crietria c=session.createCriteria(Emp.class);
2. c.addOrder(Order.asc("salary"));
3. List list=c.list();
```

HCQL with Projection

We can fetch data of a particular column by projection such as name etc. Let's see the simple example of projection that prints data of NAME column of the table only.

```
1. Criteria c=session.createCriteria(Emp.class);
2. c.setProjection(Projections.property("name"));
3. List list=c.list();
```

One to One Mapping in Hibernate by one-to-one example

As We have discussed in the previous example, there are two ways to perform one to one mapping in hibernate:

- By many-to-one element
- By one-to-one element

Here, we are going to perform one to one mapping by one-to-one element. In such case, no foreign key is created in the primary table.

In this example, one employee can have one address and one address belongs to one employee only. Here, we are using bidirectional association. Let's look at the persistent classes.

1) Persistent classes for one to one mapping

There are two persistent classes Employee.java and Address.java. Employee class contains Address class reference and vice versa.

Employee.java

```
1. package com.javatpoint;
2.
```

```
3. public class Employee {
4. private int employeeId;
5. private String name,email;
6. private Address address;
7. //setters and getters
8. }
```

Address.java

```
1. package com.javatpoint;
2.
3. public class Address {
4. private int addressId;
5. private String addressLine1,city,state,country;
6. private int pincode;
7. private Employee employee;
8. //setters and getters
9. }
```

---

2) Mapping files for the persistent classes

The two mapping files are employee.hbm.xml and address.hbm.xml.

employee.hbm.xml

In this mapping file we are using one-to-one element in both the mapping files to make the one to one mapping.

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
5.
6.          <hibernate-mapping>
7.          <class name="com.javatpoint.Employee" table="emp212">
8.          <id name="employeeId">
9.          <generator class="increment"></generator>
10.                 </id>
11.                 <property name="name"></property>
12.                 <property name="email"></property>
13.
14.                 <one-to-one name="address" cascade="all"></one-to-one>
15.                 </class>
16.
17.                 </hibernate-mapping>
```

address.hbm.xml

This is the simple mapping file for the Address class. But the important thing is generator class. Here, we are using foreigngenerator class that depends on the Employee class primary key.

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.          "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.          "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6.          <hibernate-mapping>
7.          <class name="com.javatpoint.Address" table="address212">
8.          <id name="addressId">
9.          <generator class="foreign">
```

```
10.             <param name="property">employee</param>
11.             </generator>
12.             </id>
```


```
13.             <property name="addressLine1"></property>
14.             <property name="city"></property>
15.             <property name="state"></property>
16.             <property name="country"></property>
17.
18.             <one-to-one name="employee"></one-to-one>
19.             </class>
20.
21.             </hibernate-mapping>
```

3) Configuration file

This file contains information about the database and mapping file.

hibernate.cfg.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6. <!-- Generated by MyEclipse Hibernate Tools.                   -->
7. <hibernate-configuration>
```

8.


```
9.     <session-factory>
10.             <property name="hbm2ddl.auto">update</property>
11.             <property name="dialect">org.hibernate.dialect.Oracle9Dialect<
    /property>
12.             <property name="connection.url">jdbc:oracle:thin:@localhost:1
    521:xe</property>
13.             <property name="connection.username">system</property>
14.             <property name="connection.password">oracle</property>
15.             <property name="connection.driver_class">oracle.jdbc.driver.Or
    acleDriver</property>
16.           <mapping resource="employee.hbm.xml"/>
17.           <mapping resource="address.hbm.xml"/>
18.           </session-factory>
19.
20.       </hibernate-configuration>
```

4) User classes to store and fetch the data

Store.java

```
1. package com.javatpoint;
2. import org.hibernate.cfg.*;
3. import org.hibernate.*;
4.
5. public class Store {
6. public static void main(String[] args) {
7.     Configuration cfg=new Configuration();
```

```
8.      cfg.configure("hibernate.cfg.xml");
9.      SessionFactory sf=cfg.buildSessionFactory();
10.             Session session=sf.openSession();
11.             Transaction tx=session.beginTransaction();
12.
13.             Employee e1=new Employee();
14.             e1.setName("Ravi Malik");
15.             e1.setEmail("ravi@gmail.com");
16.             Address address1=new Address();
17.             address1.setAddressLine1("G-21,Lohia nagar");
18.             address1.setCity("Ghaziabad");
19.             address1.setState("UP");
20.             address1.setCountry("India");
21.             address1.setPincode(201301);
22.
23.
24.             e1.setAddress(address1);
25.             address1.setEmployee(e1);
26.
27.             session.persist(e1);
28.             tx.commit();
29.
30.             session.close();
31.             System.out.println("success");
32.         }
33.         }
```

Fetch.java

```
1. package com.javatpoint;
2. import java.util.Iterator;
3. import java.util.List;
```

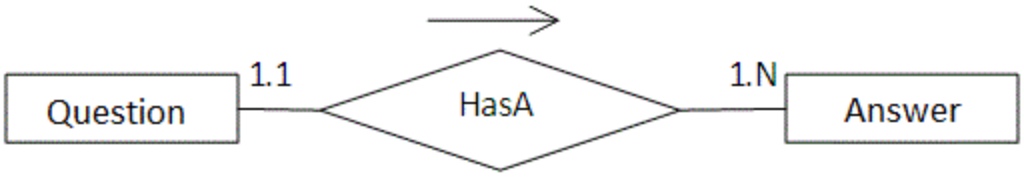
```
4. import org.hibernate.Query;
5. import org.hibernate.Session;
6. import org.hibernate.SessionFactory;
7. import org.hibernate.cfg.Configuration;
8.
9. public class Fetch {
10.       public static void main(String[] args) {
11.          Configuration cfg=new Configuration();
12.          cfg.configure("hibernate.cfg.xml");
13.          SessionFactory sf=cfg.buildSessionFactory();
14.          Session session=sf.openSession();
15.
16.          Query query=session.createQuery("from Employee e");
17.          List<Employee> list=query.list();
18.
```


```
19.          Iterator<Employee> itr=list.iterator();
20.          while(itr.hasNext()){
21.           Employee emp=itr.next();
22.           System.out.println(emp.getEmployeeId()+" "+emp.getName()+" "
    +emp.getEmail());
23.           Address address=emp.getAddress();
24.           System.out.println(address.getAddressLine1()+" "+address.getCity
    ()+" "+
25.              address.getState()+" "+address.getCountry());
26.          }
27.
28.          session.close();
29.          System.out.println("success");
30.       }
31.       }
```

One to Many mapping in Hibernate by List Example (using xml file)

If the persistent class has list object that contains the entity reference, we need to use one-to-many association to map the list element.

Here, we are using the scenario of Forum where one question has multiple answers.



In such case, there can be many answers for a question and each answer may have its own informations that is why we have used list in the persistent class (containing the reference of Answer class) to represent a collection of answers.

Let's see the persistent class that has list objects (containing Answer class objects).

```
1. package com.javatpoint;
2.
3. import java.util.List;
4.
5. public class Question {
6. private int id;
7. private String qname;
```

```
8. private List<Answer> answers;
9. //getters and setters
10.
11.       }
```

The Answer class has its own informations such as id, answername, postedBy etc.

```
1. package com.javatpoint;
2.
3. public class Answer {
4. private int id;
5. private String answername;
6. private String postedBy;
7. //getters and setters
8.
9. }
10.       }
```

The Question class has list object that have entity reference (i.e. Answer class object). In such case, we need to use one-to-many of list to map this object. Let's see how we can map it.

```
1. <list name="answers" cascade="all">
2.          <key column="qid"></key>
3.          <index column="type"></index>
4.          <one-to-many class="com.javatpoint.Answer"/>
5. </list>
```

Full example of One to Many mapping in Hibernate by List

In this example, we are going to see full example of mapping list that contains entity reference.

1) create the Persistent class

This persistent class defines properties of the class including List.

Question.java

```
1. package com.javatpoint;
2.
3. import java.util.List;
4.
5. public class Question {
6. private int id;
7. private String qname;
8. private List<Answer> answers;
9.
10.       //getters and setters
11.
12.       }
```

Answer.java

```
1. package com.javatpoint;
2.
3. public class Answer {
```

```
4. private int id;
5. private String answername;
6. private String postedBy;
7. //getters and setters
8.
9. }
10.       }
```

2) create the Mapping file for the persistent class

Here, we have created the question.hbm.xml file for defining the list.

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6.         <hibernate-mapping>
7.         <class name="com.javatpoint.Question" table="q501">
8.         <id name="id">
9.         <generator class="increment"></generator>
10.              </id>
11.              <property name="qname"></property>
12.
13.              <list name="answers" cascade="all">
14.              <key column="qid"></key>
15.              <index column="type"></index>
16.              <one-to-many class="com.javatpoint.Answer"/>
17.              </list>
18.
19.              </class>
20.
21.              <class name="com.javatpoint.Answer" table="ans501">
22.              <id name="id">
23.              <generator class="increment"></generator>
24.              </id>
25.              <property name="answername"></property>
26.              <property name="postedBy"></property>
27.              </class>
28.
29.              </hibernate-mapping>
```

3) create the configuration file

This file contains information about the database and mapping file.

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-configuration PUBLIC
3.         "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6. <!-- Generated by MyEclipse Hibernate Tools.                   -->
7. <hibernate-configuration>
```


```
8.
9.     <session-factory>
10.             <property name="hbm2ddl.auto">update</property>
11.             <property name="dialect">org.hibernate.dialect.Oracle9Dialect<
    /property>
12.             <property name="connection.url">jdbc:oracle:thin:@localhost:1
    521:xe</property>
13.             <property name="connection.username">system</property>
14.             <property name="connection.password">oracle</property>
15.             <property name="connection.driver_class">oracle.jdbc.driver.Or
    acleDriver</property>
16.          <mapping resource="question.hbm.xml"/>
17.          </session-factory>
18.
19.       </hibernate-configuration>
```

---

4) Create the class to store the data

In this class we are storing the data of the question class.

```
1. package com.javatpoint;
2. import java.util.ArrayList;
3. import org.hibernate.*;
4. import org.hibernate.cfg.*;
5.
6. public class StoreData {
7. public static void main(String[] args) {
8.     Session session=new Configuration().configure("hibernate.cfg.xml")
9.                         .buildSessionFactory().openSession();
10.         Transaction t=session.beginTransaction();
11.
12.         Answer ans1=new Answer();
13.         ans1.setAnswername("java is a programming language");
14.         ans1.setPostedBy("Ravi Malik");
15.
16.         Answer ans2=new Answer();
17.         ans2.setAnswername("java is a platform");
18.         ans2.setPostedBy("Sudhir Kumar");
```


```
19.         Answer ans3=new Answer();
20.         ans3.setAnswername("Servlet is an Interface");
21.         ans3.setPostedBy("Jai Kumar");
22.
23.         Answer ans4=new Answer();
24.         ans4.setAnswername("Servlet is an API");
25.         ans4.setPostedBy("Arun");
26.
27.         ArrayList<Answer> list1=new ArrayList<Answer>();
28.         list1.add(ans1);
29.         list1.add(ans2);
```

```
30.
31.         ArrayList<Answer> list2=new ArrayList<Answer>();
32.         list2.add(ans3);
33.         list2.add(ans4);
34.
35.         Question question1=new Question();
36.         question1.setQname("What is Java?");
37.         question1.setAnswers(list1);
38.
39.         Question question2=new Question();
40.         question2.setQname("What is Servlet?");
41.         question2.setAnswers(list2);
42.
43.         session.persist(question1);
44.         session.persist(question2);
45.
46.         t.commit();
47.         session.close();
48.         System.out.println("success");
49.     }
50.     }
```

OUTPUT

ID	QNAME
1	What is Java?
2	What is Servlet?

ID	ANSWERNAME	POSTEDBY	QID	TYPE
1	java is a programming language	Ravi Malik	1	0
2	java is a platform	Sudhir Kumar	1	1
3	Servlet is an Interface	Jai Kumar	2	0
4	Servlet is an API	Arun	2	1

How to fetch the data of List

Here, we have used HQL to fetch all the records of Question class including answers. In such case, it fetches the data from two tables that are functional dependent. Here, we are direct printing the object of answer class, but we

have overridden thetoString() method in the Answer class returning answername and poster name. So it prints the answer name and postername rather than reference id.

FetchData.java


```
1. package com.javatpoint;
2. import java.util.*;
3. import org.hibernate.*;
4. import org.hibernate.cfg.*;
5.
6. public class FetchData {
7. public static void main(String[] args) {
8.
9.     Session session=new Configuration().configure("hibernate.cfg.xml")
10.                         .buildSessionFactory().openSession();
11.
12.         Query query=session.createQuery("from Question");
13.         List<Question> list=query.list();
14.
15.         Iterator<Question> itr=list.iterator();
16.         while(itr.hasNext()){
17.            Question q=itr.next();
18.            System.out.println("Question Name: "+q.getQname());
19.
20.            //printing answers
21.            List<Answer> list2=q.getAnswers();
22.            Iterator<Answer> itr2=list2.iterator();
23.            while(itr2.hasNext()){
```

```
24.                 System.out.println(itr2.next());
25.             }
26.
27.         }
28.         session.close();
29.         System.out.println("success");
30.     }
31.     }
```

Many to Many Mapping in hibernate by map Example (using xml file)

We can map many to many relation either using set, bag, map etc. Here, we are going to use map for many-to-many mapping. In such case, three tables will be created.

Example of Many to Many Mapping

You need to create following pages for mapping map elements.

- Question.java
- User.java
- question.hbm.xml
- user.hbm.xml
- hibernate.cfg.xml
- StoreTest.java
- FetchTest.java

Question.java

```
1. package com.javatpoint;
2.
3. import java.util.Map;
4.
5. public class Question {
6. private int id;
7. private String name;
8. private Map<String,User> answers;
9.
10.    public Question() {}
11.    public Question(String name, Map<String, User> answers) {
12.        super();
13.        this.name = name;
14.        this.answers = answers;
15.    }
16.    public int getId() {
17.        return id;
18.    }
```


```
19.
20.    public void setId(int id) {
21.        this.id = id;
22.    }
23.    public String getName() {
24.        return name;
25.    }
26.    public void setName(String name) {
27.        this.name = name;
28.    }
29.    public Map<String, User> getAnswers() {
30.        return answers;
31.    }
32.    public void setAnswers(Map<String, User> answers) {
33.        this.answers = answers;
```

```
34.    }
35.
36.
37.    }
```

User.java

```
1. package com.javatpoint;
2.
3. public class User {
4. private int id;
5. private String username,email,country;
6.
7. public User() {}
8. public User(String username, String email, String country) {
9.     super();
10.          this.username = username;
11.          this.email = email;
12.          this.country = country;
13.      }
14.      public int getId() {
15.          return id;
16.      }
17.
18.      public void setId(int id) {
19.          this.id = id;
20.      }
21.
22.
23.      public String getUsername() {
24.          return username;
25.      }
26.
27.      public void setUsername(String username) {
28.          this.username = username;
29.      }
30.
31.      public String getEmail() {
32.          return email;
33.      }
34.
35.      public void setEmail(String email) {
```

```
36.         this.email = email;
37.     }
38.
39.     public String getCountry() {
40.         return country;
41.     }
42.
43.     public void setCountry(String country) {
44.         this.country = country;
45.     }
46.     }
```

question.hbm.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
5.
6. <hibernate-mapping>
7. <class name="com.javatpoint.Question" table="question738">
8. <id name="id">
9. <generator class="native"></generator>
10.     </id>
11.     <property name="name"></property>
12.
13.     <map name="answers" table="answer738" cascade="all">
14.     <key column="questionid"></key>
15.     <index column="answer" type="string"></index>
16.     <many-to-
    many class="com.javatpoint.User" column="userid"></many-to-many>
17.     </map>
18.     </class>
19.
20.     </hibernate-mapping>
```

user.hbm.xml

```
1. <?xml version='1.0' encoding='UTF-8'?>
2. <!DOCTYPE hibernate-mapping PUBLIC
3.         "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
4.         "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
```

```
5.
6.  <hibernate-mapping>
7.  <class name="com.javatpoint.User" table="user738">
8.  <id name="id">
9.  <generator class="native"></generator>
10.         </id>
11.         <property name="username"></property>
12.         <property name="email"></property>
13.         <property name="country"></property>
14.         </class>
15.
16.         </hibernate-mapping>
```

hibernate.cfg.xml

```
1.  <?xml version='1.0' encoding='UTF-8'?>
2.  <!DOCTYPE hibernate-configuration PUBLIC
3.          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
4.          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
5.
6.  <!-- Generated by MyEclipse Hibernate Tools.                   -->
7.  <hibernate-configuration>
8.
9.    <session-factory>
10.         <property name="hbm2ddl.auto">update</property>
11.         <property name="dialect">org.hibernate.dialect.Oracle9Dialect</pr
    operty>
```

```
12.         <property name="connection.url">jdbc:oracle:thin:@localhost:1521
    :xe</property>
13.         <property name="connection.username">system</property>
14.         <property name="connection.password">oracle</property>
15.         <property name="connection.driver_class">oracle.jdbc.driver.Oracl
    eDriver</property>
16.
17.          <mapping resource="question.hbm.xml"/>
18.          <mapping resource="user.hbm.xml"/>
19.          </session-factory>
20.
21.       </hibernate-configuration>
```

StoreTest.java

```
1. package com.javatpoint;
2.
3. import java.util.HashMap;
4. import org.hibernate.*;
5. import org.hibernate.cfg.*;
6. public class StoreTest {
7. public static void main(String[] args) {
8. Session session=new Configuration().configure().buildSessionFactory().open
    Session();
9. Transaction tx=session.beginTransaction();
10.
11.       HashMap<String,User> map1=new HashMap<String,User>();
12.       map1.put("java is a programming language",
13.       new User("John Milton","john@gmail.com","usa"));
14.
15.       map1.put("java is a platform",
```

```
16.         new User("Ashok Kumar","ashok@gmail.com","india"));
17.
18.         HashMap<String,User> map2=new HashMap<String,User>();
19.         map2.put("servlet technology is a server side programming",
20.
21.         new User("John Milton","john@gmail.com","usa"));
22.         map2.put("Servlet is an Interface",
23.         new User("Ashok Kumar","ashok@gmail.com","india"));
24.
25.         Question question1=new Question("What is java?",map1);
26.         Question question2=new Question("What is servlet?",map2);
27.
28.         session.persist(question1);
29.         session.persist(question2);
30.
31.         tx.commit();
32.         session.close();
33.         System.out.println("successfully stored");
34.         }
35.         }
```

FetchTest.java

```
1. package com.javatpoint;
2. import java.util.*;
3. import org.hibernate.*;
4. import org.hibernate.cfg.*;
5. public class FetchTest {
6. public static void main(String[] args) {
7.  Session session=new Configuration().configure().buildSessionFactory().ope
   nSession();
```

```
8.  Query query=session.createQuery("from Question ");
9.  List<Question> list=query.list();
10.
11.        Iterator<Question> iterator=list.iterator();
12.        while(iterator.hasNext()){
13.         Question question=iterator.next();
14.         System.out.println("question id:"+question.getId());
15.         System.out.println("question name:"+question.getName());
16.         System.out.println("answers.....");
17.         Map<String,User> map=question.getAnswers();
18.         Set<Map.Entry<String,User>> set=map.entrySet();
19.
20.         Iterator<Map.Entry<String,User>> iteratoranswer=set.iterator();
21.         while(iteratoranswer.hasNext()){
22.          Map.Entry<String,User> entry=(Map.Entry<String,User>)iteratoran
    swer.next();
23.          System.out.println("answer name:"+entry.getKey());
24.          System.out.println("answer posted by.....");
25.          User user=entry.getValue();
26.          System.out.println("username:"+user.getUsername());
27.          System.out.println("user emailid:"+user.getEmail());
28.          System.out.println("user country:"+user.getCountry());
29.         }
30.        }
31.       session.close();
32.       }
33.       }
```

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/ccee2025notes

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project