HIBERNATE FRAMEWORK

by

Mr. Ashok

HIBERNATE

❖ **ORM:-** (ORM stands for): Object Relational Mapping. It is a theory concept used at database programming to perform operations like insert. Update, delete and select .in object format only ie. JDBC converts object to primitive data and SQL Query shuld be written by programmer using primitives, which is not following OOPs.

ORM says "Do not convet object data, do operations in OOPs. Format only".

- For this concept programmer gend follow mapping rule. Given as

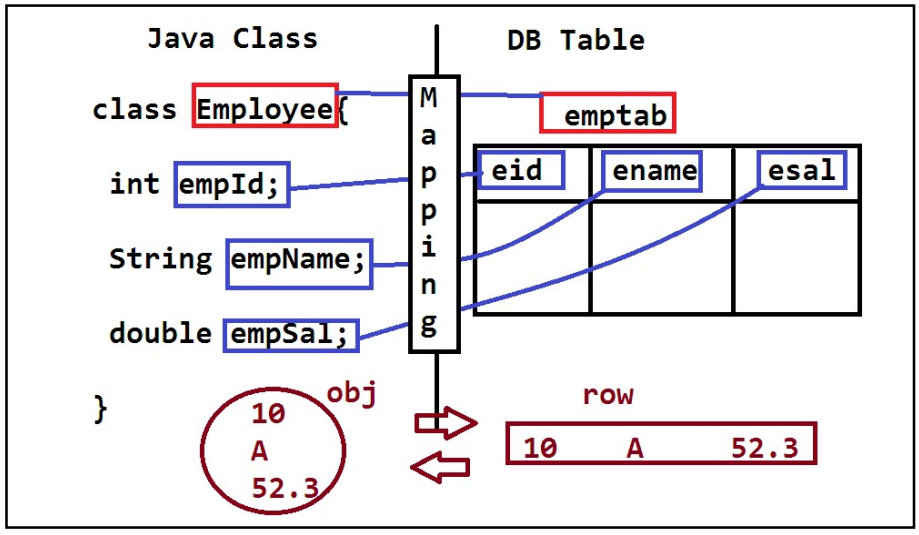
1. className- Must be mapped with – tableName
2. VariableName- Must be mapped with – columnName

** should be done by programmer using XML/Annotatios concept.

** Then ORM convert

Object ⇔ROW

** Here , ORM only generates SQL Query.



FrameWork:-

FrameWork is a Combination of diierent technologies and design patterns by using framework we can develop complete application or a part of application faster. It is also called as RAD.

RAD= Rapid Application Development (faster coding)

Technology:-

It is a concept used for fixed type of coding. In case of java JDBC.servlet. JSP, JMS, JAXB, JSF, EJB etc….

are few technologies.

---JDBC used for only DB Operations

--- Servlets used for only Dynamic Web Applications.

Design Pattern:-

This concept is used to improve applications performance by reducing code , time, and memory of on application.

$$\text{Performance} \propto \frac{1}{\text{Code, time, memory}}$$

Example design Pattern are:-
Singleton, factry, Abstract,Prototype, context, Proxy etc…..

Hiberante 5 .x

Hibernate is a framework which has inbuilt technologies and design pattern. It follows ORM concept. (ORM=Object Relational Mapping)

Hibernate has 3 Java Technologies

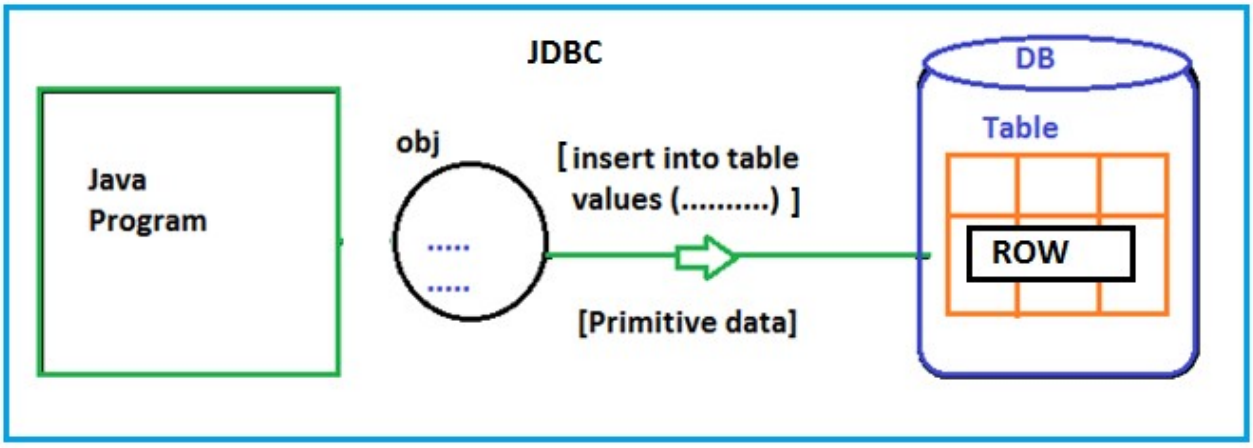
1. JDBC
2. JTA
3. JNDI

And one Non-Java Technology "XML".

1. **JDBC =Java Database Connectivity :-**
   It is used to perform database operations. Like insert, update, delete and select.
2. **JTA = Java Transaction API**
   It is used to either confirm (commit) changes or cancel (rollback) changes done at database.
3. **JNDI= Java Naming Directory Interface**
   It is used to create connection-pool ot execute SQL Statement faster.

** **POOL =** it is a group of similar type object.

Hibernate Programming design:-
Below design explain how JDBC and hibernate works.



JDBC works on primitive data even object is given, it converts Object Data to Primitive Data then SQL Query return manually by programmer.
Hibernate follows ORM Concept so, it converts object to Row even Row to Object. SQL query also generated by ORM (Hibernate), But Programmer has to follow coding files in below order.

1. Configuration file

2. Model/Entity/POJO Class
3. Mapping code(XML/Annotation)
4. Test Class [POJI-POJO]

1. Configuration file:-

It is used to provide all details related to DB and Hibernate.

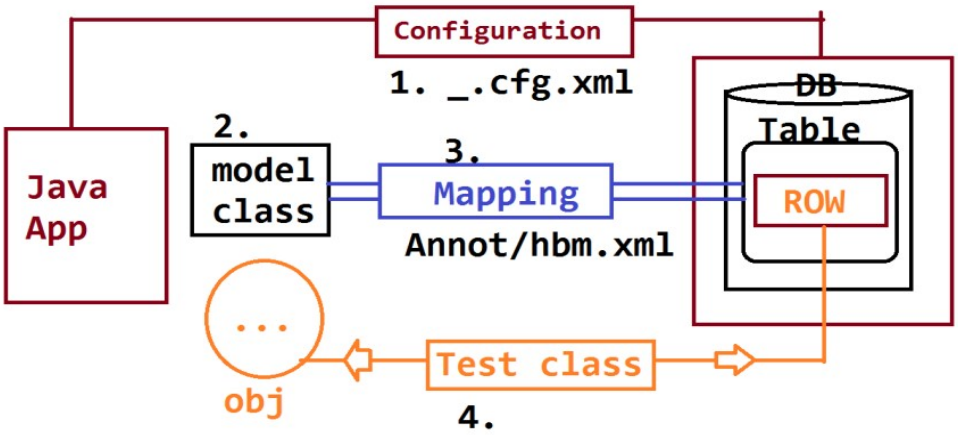
a). it will store data in key = value format.

b). file name should have extension____.cfg.xml

c). recommended name is:-

hibernate.cfg.xml

d). Few keys are:- driver_Class, url, dialect , show_sql



----------**KEYs:**----------

***) Connection keys:-**

We must specify 4 key for DB Connection. Those are:

1. Driver_class
2. url
3. username
4. password

***) Hibernate keys:-**

5. Dialect: it is a pre-defined class in hibernate.

under package: **org.hibernate.dialect**

>> It will generate SQL Query when Programmer performs any operations (**save,update,delete,……**)

>> DB to DB dialect also changes.

Ex:- Dialects are

OracleDialect, MySQLDialect, DB2Dialect, SybaseDialect, etc….

Model Class + Rules given by hibernate:-

1. class must have package statement
2. Class must have public (no. of table =no. of classes)

3. variables must be private // (no. of column =no .of variables)
4. Default constructor with setters and getters (mutators).
5. Can override methods from Object , those are(3)
   toString(), equals(), hashCode()non-final ,non-static,non-private.
6. Annotations:-JPA Annotations(Java Persistency API=JPA)

Hibernate Application Files:-

To write one application we should write 4 files, given as below:

1. **Model class** **(Class)**
2. **Mapping Code** **(XML/ANNOTATION)**
3. **Configuration file** **(XML)**
4. **Test Class** **(class)**

1. **Model Class:-** here Model mean Data.

- It can also be called as Entity/POJO
- It is c class follows rules given by Hibernate Framework. Those are:

a) Class must have package statement
b) Class must be public Type.
   (Number of Tables = No. Of Model Classes)
c) Class can have variables, must be type private
   [No. of Columns= No. Of Variables]
d) Class should have default Constructor and setter-getters methods (Mutators)(Hiberante uses def.const,set/gets)
e) Class can override methods from java.lang.Object(C), Those are:
   toString(), equals(), and hashCode()

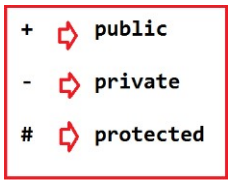
*** These methods are non-final, non-private and non-static, So we can override methods in models class.

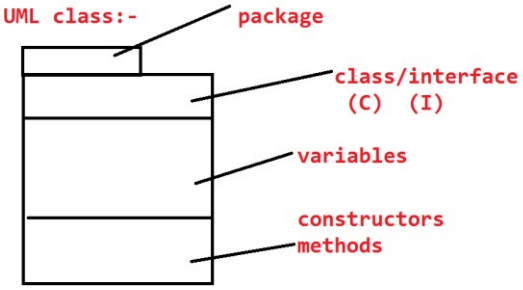
f) Class can have annotations given by JPA (Java Persistency API) and also Core Annotation(=An annotation defined in java.lang package)
g) Class can inherit (IS-A) [extends/implements] only Hibernate API (HibernateClasses and interfaces)

***** One special interface is allowed that is java.io.Serializable(I) on model class.**

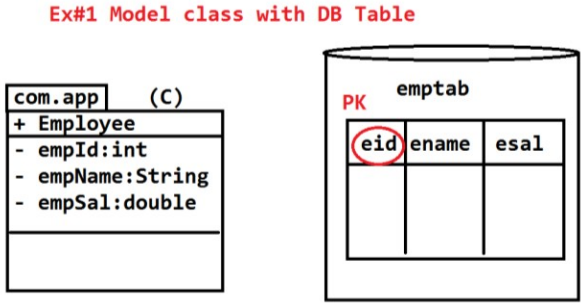
**** Model class Coding**

Every model class must be mapped with DB Table using JPA Annotations.
Follow UML Class and DB Table Design, given as
+ Means Public
- Means Private
#Means Protected





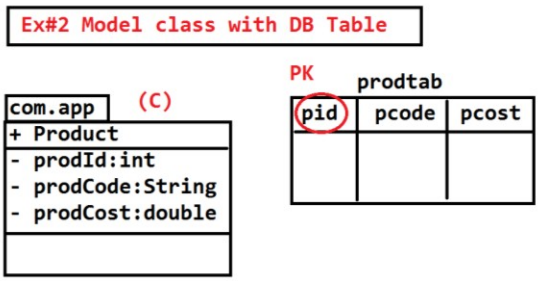
EX#! Model class with DB Table



Equal java code is:

```
@Entity
@Table(name="empTab")
publicclass Employee {
        @Id
        @Column(name="eid")
        privateintempId;
        @Column(name="ename")
        private String empName;
        @Column(name="esal")
        privatedoubleempSal;
```

Ex2 Model class with DB Table




```
package com.app.model;
// ctlr+shift+o(imports)
@Entity
@Table(name="prodtab")
publicclass Product {
        @Id
        @Column(name="pid")
        privateintProdId;
        @Column(name="pcode")
        private String prodCode;
        @Column(name="pcost")
        privatedoubleProdCost;
        }
```

@Entity : It maps model class with DB Table and Variables with columns
@Id: It indicates primary Key. Every Table must Contain Primary key.
@Table :- It is Optional , We can provide column details like column name.
*** if **@Table, @Column** are not provided then **className** is **TableName,variableName** is **ColumnName**(taken by Hibernate**);**
Hibernate Configuration File:- for one application we should provide one configuration file
***) it is XML Format**
*) it contains details like properties + mapping classes
Cfg= property + mapping class
*) Here Property means data in key= value format.

Keys are given below (8 keys)

1) **hibernate .connection.driver_class**=oracle.jdbc.driver.OracleDriver
2) **hibernate.connection.url**=jdbc:oracle:thin@localhost:1521:xe
3) **hibernate.connection.username**=system
4) **hibernate.connection.password**=system
5) **hibernate.dialect=**org.hibernate.dialect.OracleDialect
6) **hibernate.show_sql**=true
7) **hibernate.format_sql**=true
8) **hibernate.hbm2ddl.auto**=update

1. **dialect:** dialect is class it will be generate the SQL Query when programmer performs operation, for every database dialect is different

EX:- Oracle DB= OracleDialect
MySQL DB= MySQLDialect
Sybase DB= SybaseDialect
H2 DB = H2Dialect
All dialect are defined in package org.hibernate.dialect

2. **show_sql:-** it is a Boolean property default value is false. To see generated SQL on Console make value as true.
3. **format_sql:-** it is a Boolean property default value is false. It will display sql clause by clause (part by part)

Ex:- Select
Eid ,nam
From
Emptab
Where
Eid=10
Value must be set to 'true'

4. hbm2ddl.auto:- here hbm= hiberante mapping
ddl=Data definition language(create /alter/drop in SQL)
it has four possible value. **Those are:-**

a. validate (default value)
b. create
c. update
d. create-drop

A. **Validate:-** in this case hibernate creates no tables programmer has to create or modify tables manually. It is only default value.
B. **Create:-** hibernate creates always new tables, if table exist then it will be drop.
C. **Update**:- it creates new table if table not exits else uses same tables.
D. **Create-drop:-** this option is used for testing process not in development, it's new table and performs operation at last table will be drop.
*) configuration file must follow naming rule given by hibernate for auto-detection.

Naming rule is: hibernate.cfg.xml
***) it look like:**
Hibernate.cfg.xml

```
<hibernate-configuration>
<session-factory>
<property name="key">value</property>
……
……
<mapping class="-------"/>
</session-factory>
</hibernate-configuration>
```

***) Example code with key=value(Oracle)**

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

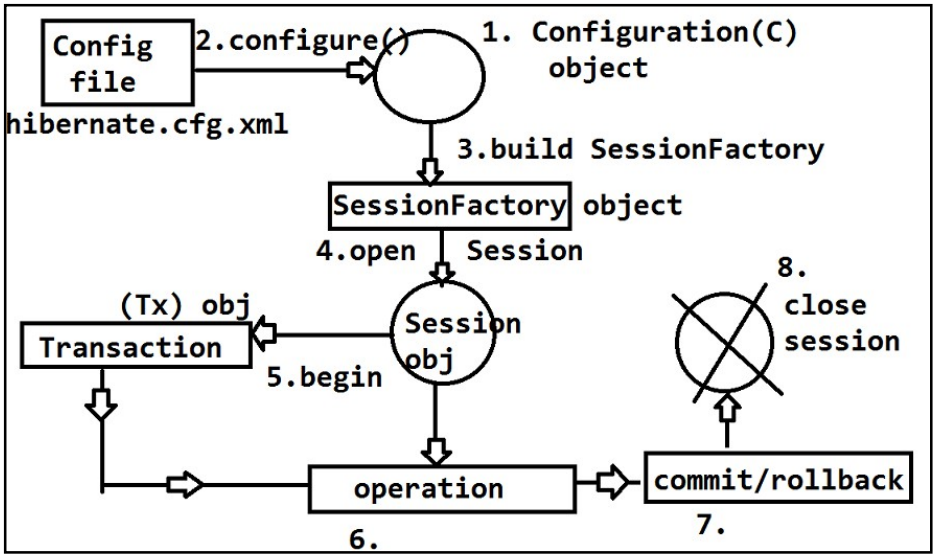
```
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hiberante.dialect"> org.hibernate.dialect.OracleDialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Product"/>
</session-factory>
</hibernate-configuration>
```

4). Test Class:-

To perform any operation in hibernate we must write test class. It is used to perform operations like:
select and non-select (insert, update, delete).
*** Transaction object is required for non-select operation. For “Select” it is not required, even written no exception but memory is wasted.

*****Test class coding and Execution flow*****

1. Create object to configuration (c)
2. Load .cfg.xml file into configuration using configure () method.
3. Build SessionFactory using cfg, which handles
   a. Loading driver class
   b. Creating connection
   c. Prepare statements
4. **Open Session to perform on operation (**either select or non-select)
5. Begin Transaction(Tx) if non-select operation is to be performed.
6. Now perform operation using session.
7. Commit or rollback transaction if tx started.
8. Close session at last.



Coding Steps with Concept:-

1. Create empty configuration object using class "configuration" given by hibernate.
   Configuration cfg=new Configuration();
2. Load hibernate.cfg.xml file into above object using method configure()
   cfg.configure();
   ** if XML file name or location is different then code will be:
   Cfg.configure("abcd.cfg.xml");
   Cfg.configure("com/app/one.cfg.xml");
3. Create object to SessionFactory using cfg, which load driver class and creates connection and statement type.

SessionFactory sf=cfg.buildSessionFactory();

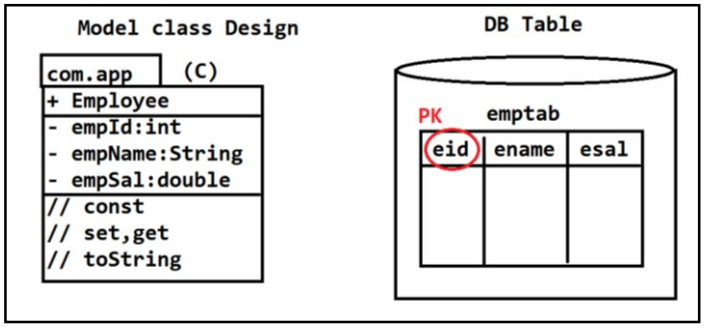
4. To perform operations (Task) create one Session object using SF.
   Session ses=sf.openSession();
5. Start one Transaction if operation type is non-select (insert, update and delete). If select operation then Tx not required.
   Transaction tx=ses.beginTransaction();
6. Perform operation using session

……..
……..

7. Either commit or rollback if Tx is started.

   tx.commit ()/tx.rollback();
8. Finally close Session
   ses.close();

Example code for below UML-ER Design:-



Model class & 2. Mapping code

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
// ctlr+shift+o(imports)
@Entity
@Table(name="empTab")
publicclass Employee {
	@Id
	@Column(name="eid")
privateintempId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	privatedoubleempSal;
	public Employee() {
		super();
	}
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
```

```
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + "]";
	}
}
```

1. **Configuration file**

```
	<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC"-//Hibernate/Hibernate Configuration DTD
3.0//EN""http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

	<hibernate-configuration>
	<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
	<propertyname="hibernate.connection.username">system</property>
	<propertyname="hibernate.connection.password">system</property>
	<propertyname="hiberante.dialect"> org.hibernate.dialect.OracleDialect</property>
	<propertyname="hibernate.show_sql">true</property>
	<propertyname="hibernate.format_sql">true</property>
	<propertyname="hibernate.hbm2ddl.auto">update</property>
	<mappingclass="com.app.model.Product"/>
	</session-factory>
</hibernate-configuration>
```

2. **Test Class**

```
	package com.app.model;
	import  org.hibernate.Session;
	import org.hibernate.SessionFactory;
	import org.hibernate.Transaction;
	import org.hibernate.cfg.Configuration;
	publicclass Test {
		publicstaticvoid main(String[] args) {
			Configuration cfg=new Configuration();
			cfg.configure();
			SessionFactory sf=cfg.buildSessionFactory();
			Session ses=sf.openSession();
			Transaction tx=ses.beginTransaction();
			Employee e=new Employee();
			e.setEmpId(1001);
```

```
                e.setEmpName("Ramjatan kumar");
                e.setEmpSal(22.22);
                ses.saveOrUpdate(e);
                tx.commit();
                ses.close();
        }
    }
```

Software Required:-

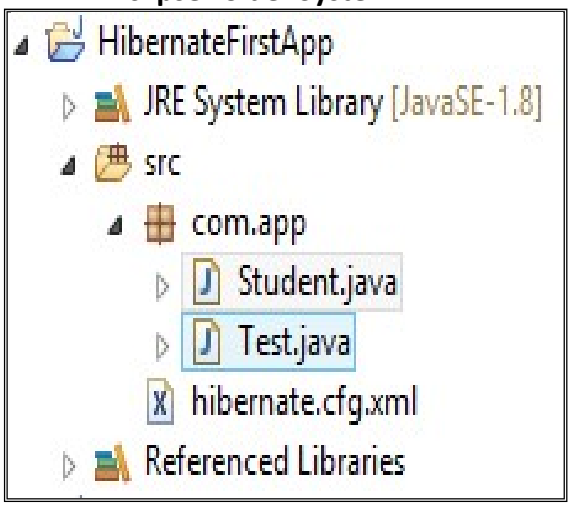
1. JDK 8/9/10
   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Eclipse Java EE | STS(Strng Tool Suite) Tool
   https://spring.io/tools/sts/all
   http://download.springsource.com/release/STS/3.9.5.RELEASE/dist/e4.8/spring-tool-suite-3.9.5.RELEASE-e4.8.0-win32-x86_64.zip
3. Oracle10gXE | MySQL5
   http://www.mediafire.com/file/roc48z1i6iitpu4/OracleXE.exe/file
4. **Jars:**

Hibernate Jars 3.x/4.x/5.x

https://sourceforge.net/projects/hibernate/files/hibernate-orm/5.2.17.Final/hibernate-release-5.2.17.Final.zip/download

DB Jars: ojdbc.jar/myusql-connector jars

----------Eclipse Folder System----------



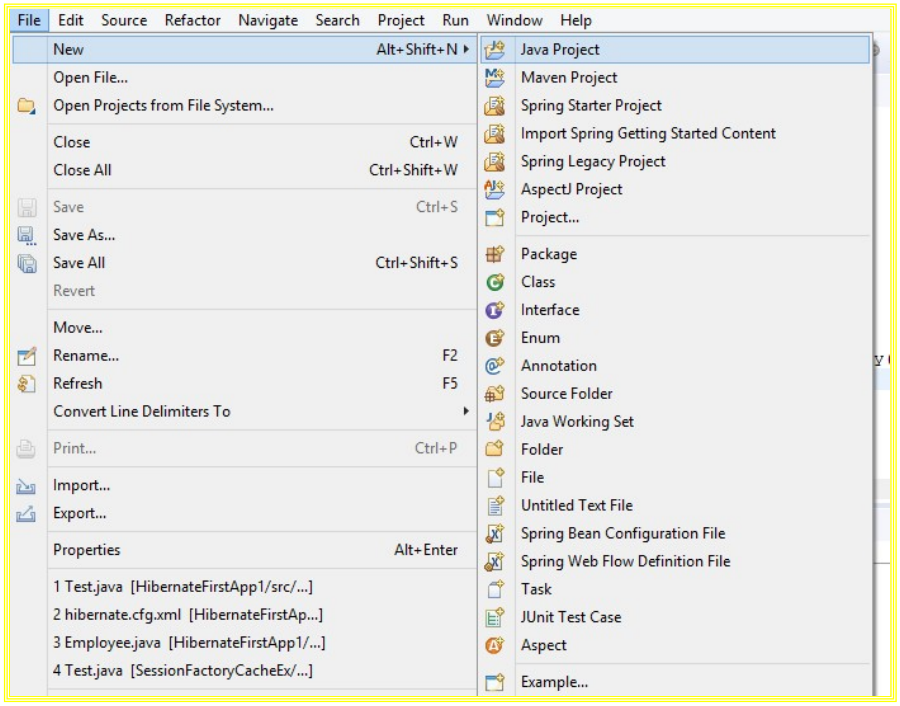
Hibernate Jars Download Link:
http://sourceforge.net/projects/hibernate/files/hibernate-orm/5.2.17.Final/
hibernate.cfg.xmlabove content download
https://gist.github.com/yusufcakmak/aefef12a7fe4518043ec

- **Choose .Zip > download > Extract**
- **Open Folder > goto lib folder**
- **Open required folder > 10 jars**

Show Images Process of creates Step By Step Project in Eclips Id

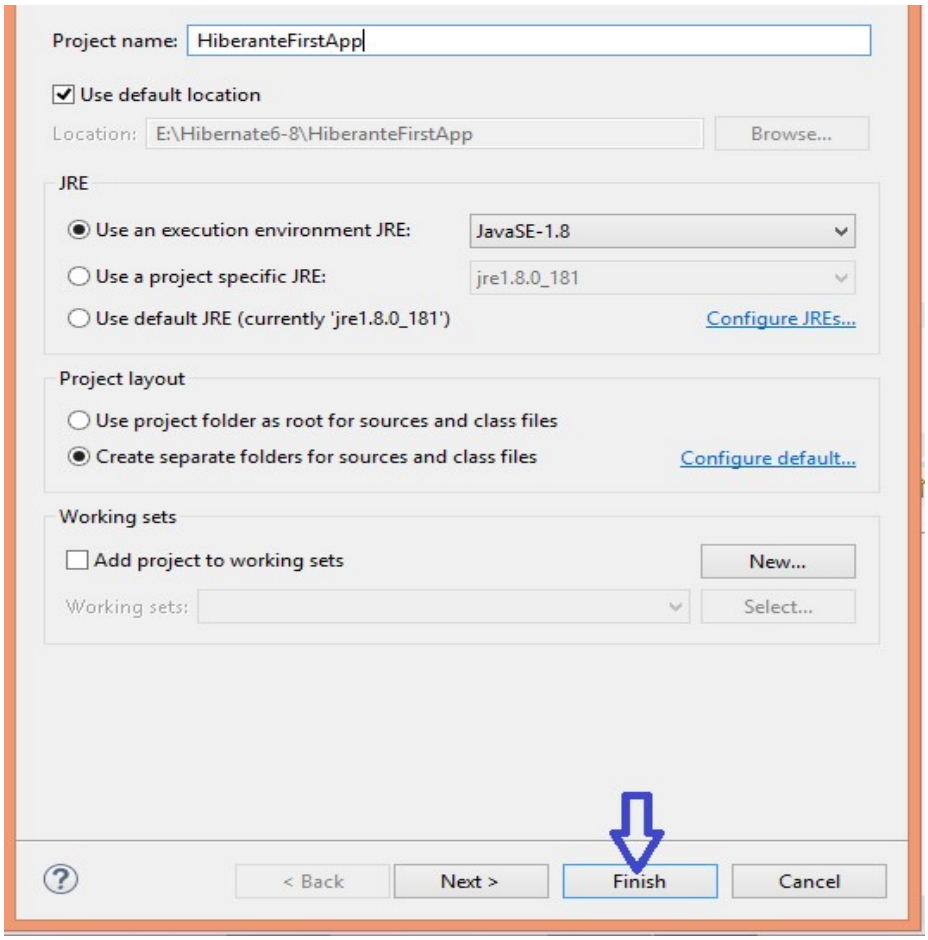
Step 1.

- **a.** click on File Menu
- **b.** Go to Java Project and Click



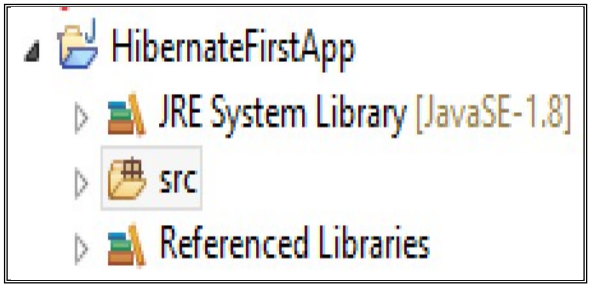
Step 2.

- **a.** Enter the Project Name In Text Area
- **b.** After that Click on Finish Button



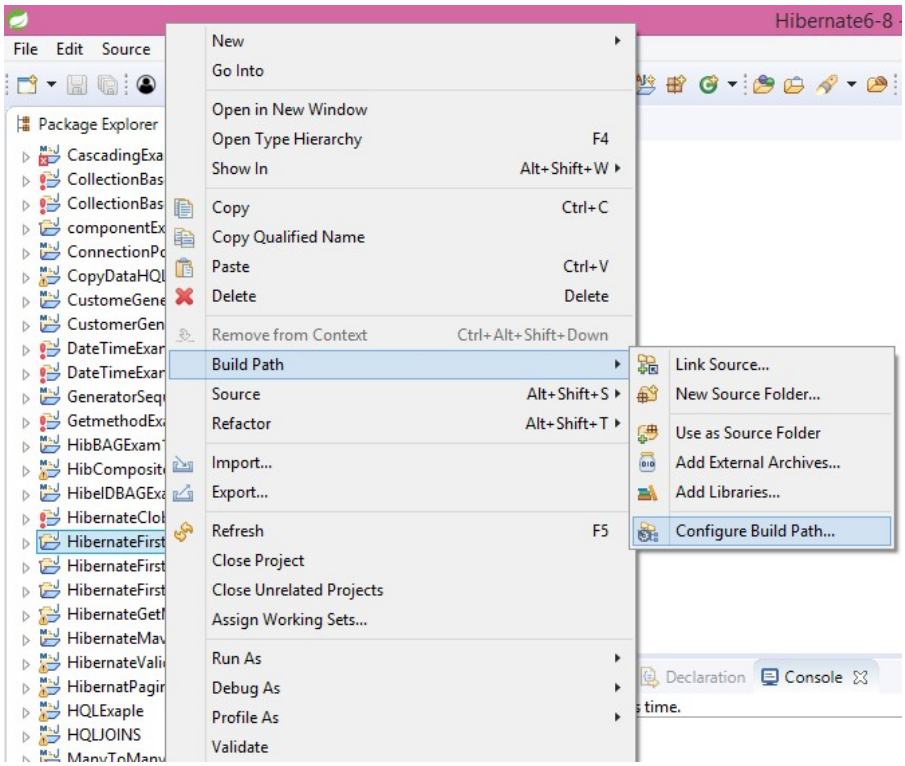
Steps 3.

a. Double Click on Project Name

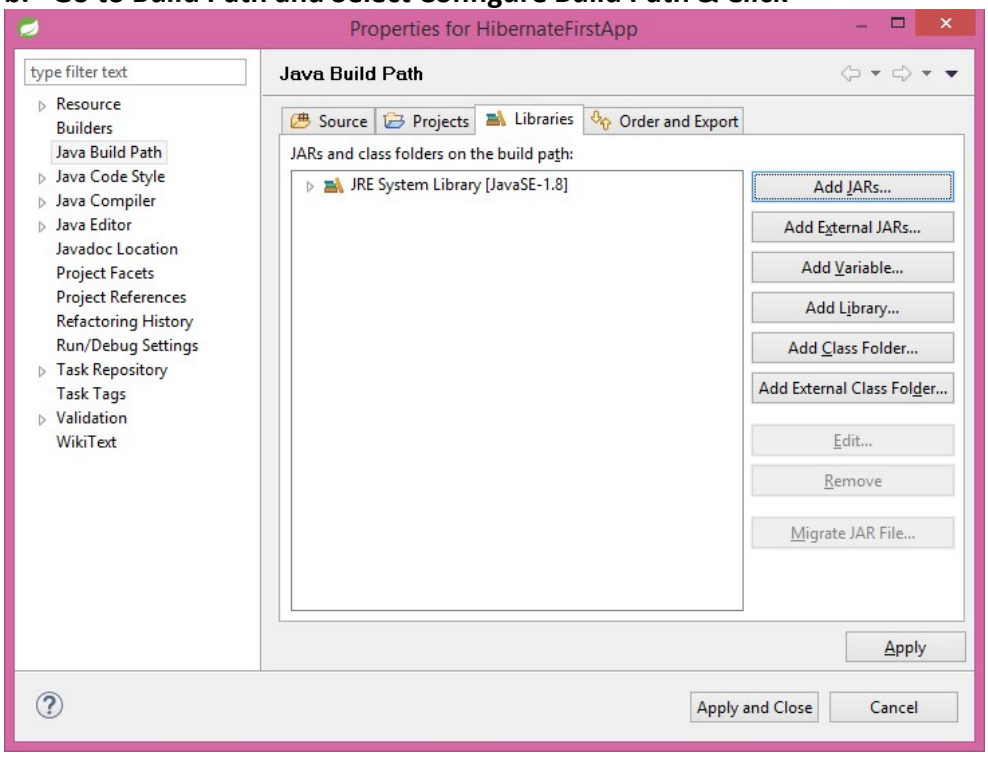


Steps 4. (build path)

a. Right click on ProjectName

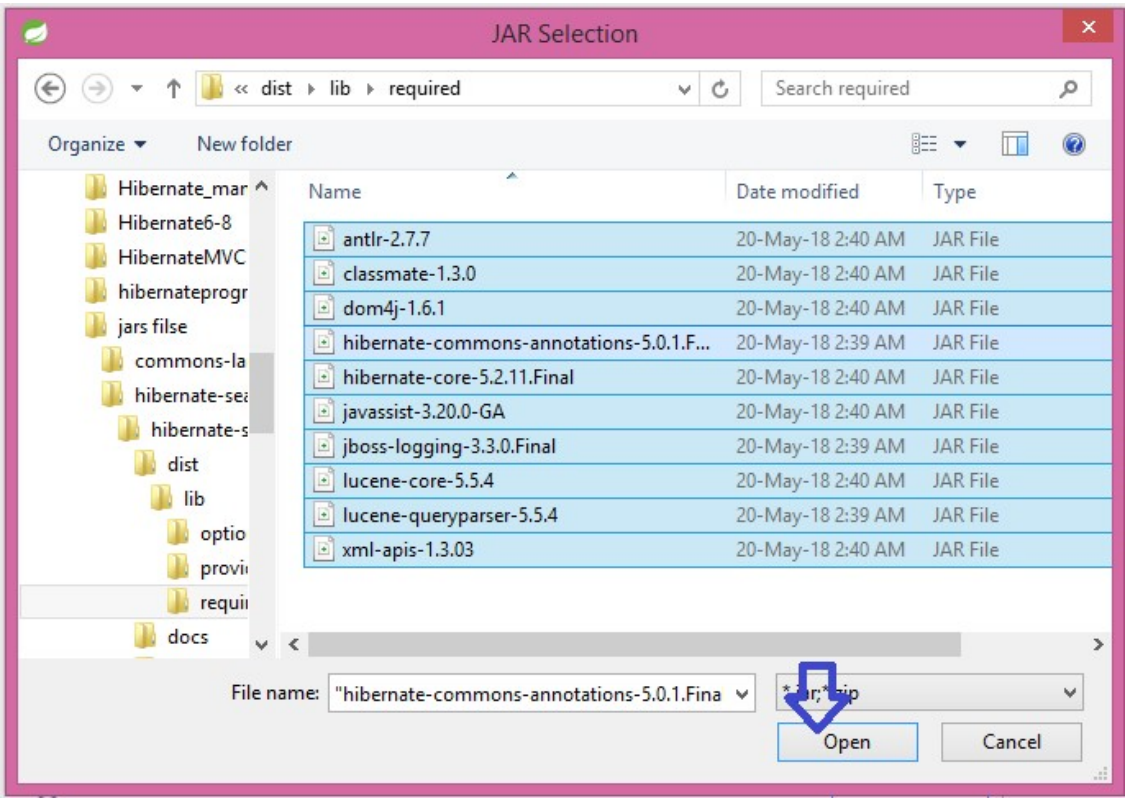


b. Go to Build Path and Select Configure Build Path & Click



c. Click on Libraries tab & Add External JARS…

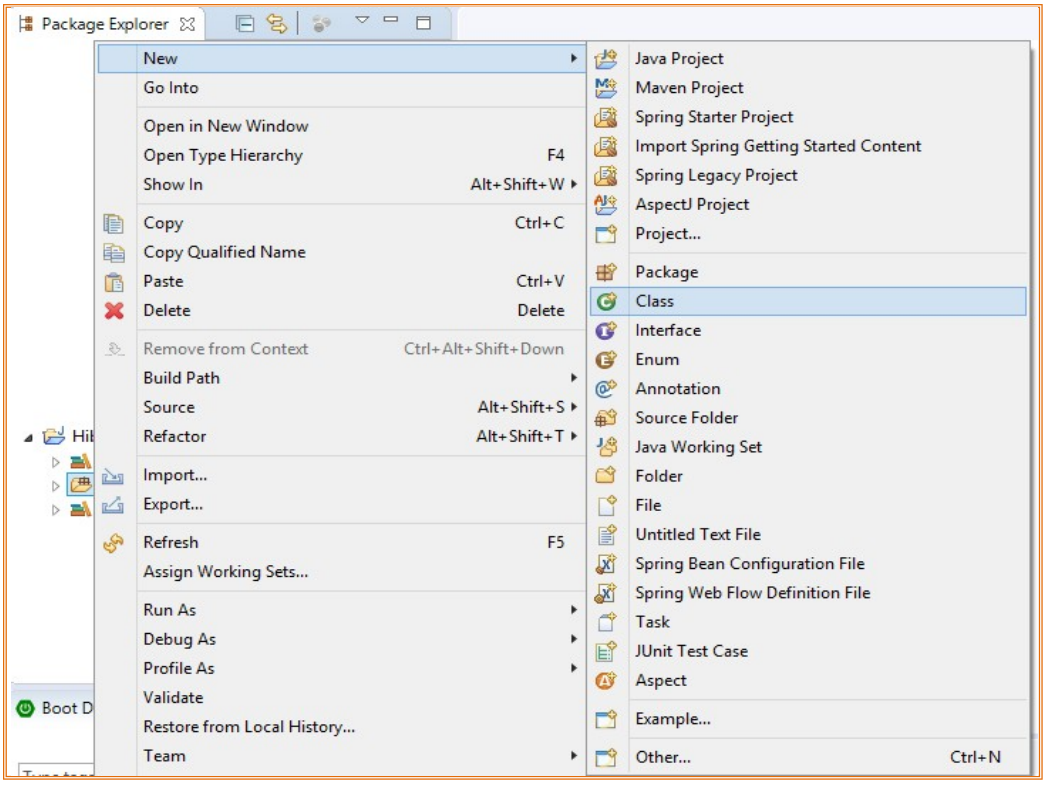
d. Search for Jars Location



e. **Click on Apply and Apply close**

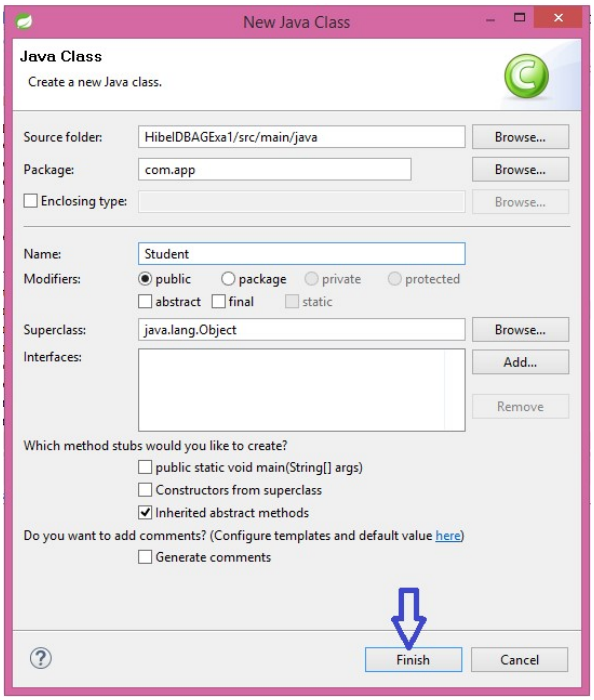
Steps 5.

a. **Right Click on Src Folder**
b. **Go to New Option**
c. **Select the Class and Click**



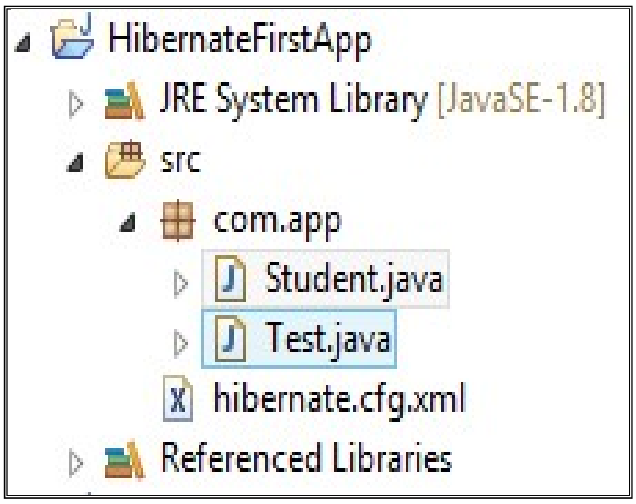
Steps 5.

a. **Enter Package Name in Text Area**
b. **Enter Class Name and click Finish Button**



(same as like test class also create)

Folder structure



1. **Model Class**

```
package com.app;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table()
publicclass Student {
@Id
@Column(name="sid")
privateintstdId;
@Column(name="sname")
private String stdName;
@Column(name="sfee")
privatedoublestdFee;
@Column(name="age")
privateintAge;
public Student() {
        super();
}
publicint getStdId() {
        returnstdId;
}
publicvoid setStdId(intstdId) {
        this.stdId = stdId;
}
public String getStdName() {
        returnstdName;
```

```
}
publicvoid setStdName(String stdName) {
        this.stdName = stdName;
}
publicdouble getStdFee() {
        returnstdFee;
}
publicvoid setStdFee(doublestdFee) {
        this.stdFee = stdFee;
}
publicint getAge() {
        returnAge;
}
publicvoid setAge(intage) {
        Age = age;
}
@Override
public String toString() {
        return"Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" + stdFee + ",
Age=" + Age + "]";
}
```

1. }**configuration file**

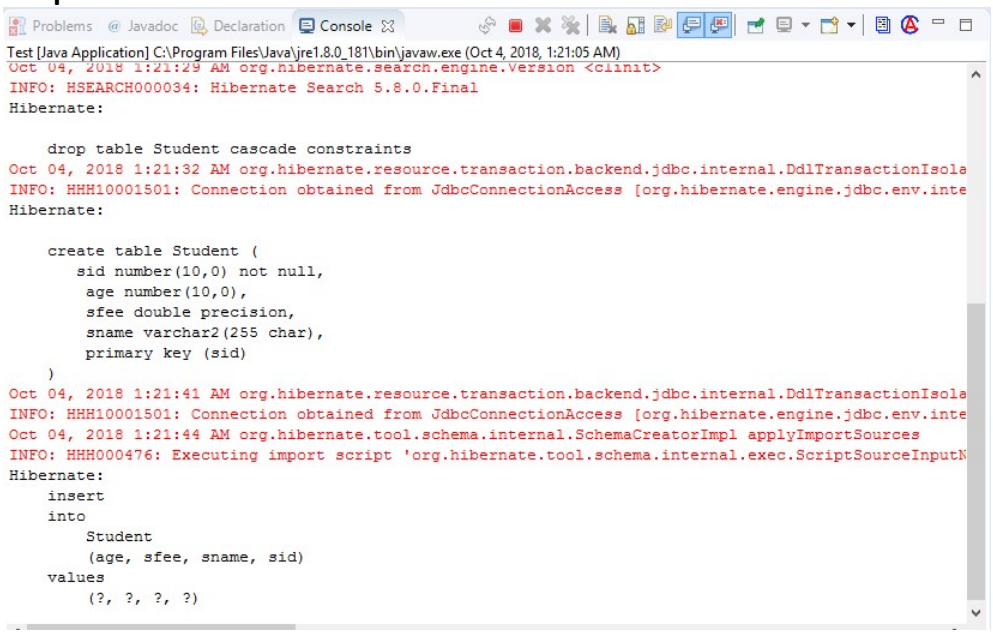
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC"-//Hibernate/Hibernate Configuration DTD
3.0//EN""http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hiberante.dialect"> org.hibernate.dialect.OracleDialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Student"/>
</session-factory>
</hibernate-configuration>
```

Test class

```
package com.app;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
```

```
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
publicclass Test {
        publicstaticvoid main(String[] args) {
                Configuration cfg=new Configuration();
                cfg.configure();
                SessionFactory sf=cfg.buildSessionFactory();
                Session ses=sf.openSession();
                Transaction tx=ses.beginTransaction();
                Student st=new Student();
                st.setStdId(1001);
                st.setStdName("Ramjatan");
                st.setStdFee(22.2);
                st.setAge(24);
                ses.save(st);
                tx.commit();
                ses.close();
        }
}
```

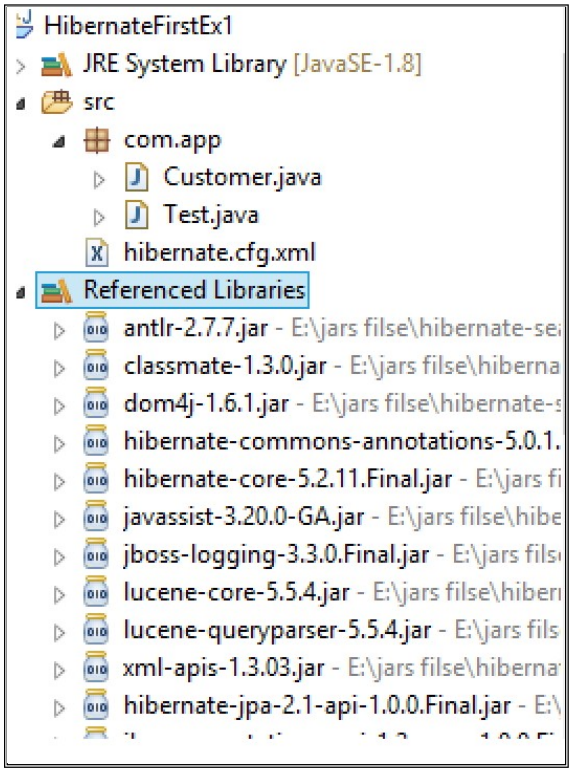
Output:-



Oracle Data base

Hibernate FirstAppExa2

Folder Structure



Model class:-

```
package com.app;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="custTab")
public class Customer {
```

```
@Id
@Column(name="cid")
        private int CustId;
@Column(name="cname")
        private String CustName;
@Column(name="cadd")
        private String CustAdd;
public Customer() {
        super();
}
public int getCustId() {
        return CustId;
}
public void setCustId(int custId) {
        CustId = custId;
}
public String getCustName() {
        return CustName;
}
public void setCustName(String custName) {
        CustName = custName;
}
public String getCustAdd() {
        return CustAdd;
}
public void setCustAdd(String custAdd) {
        CustAdd = custAdd;
}
@Override
public String toString() {
        return "Customer [CustId=" + CustId + ", CustName=" + CustName + ", CustAdd=" +
CustAdd + "]";
}
}
```

1. Configuration file (hibernate.cfg.xml)

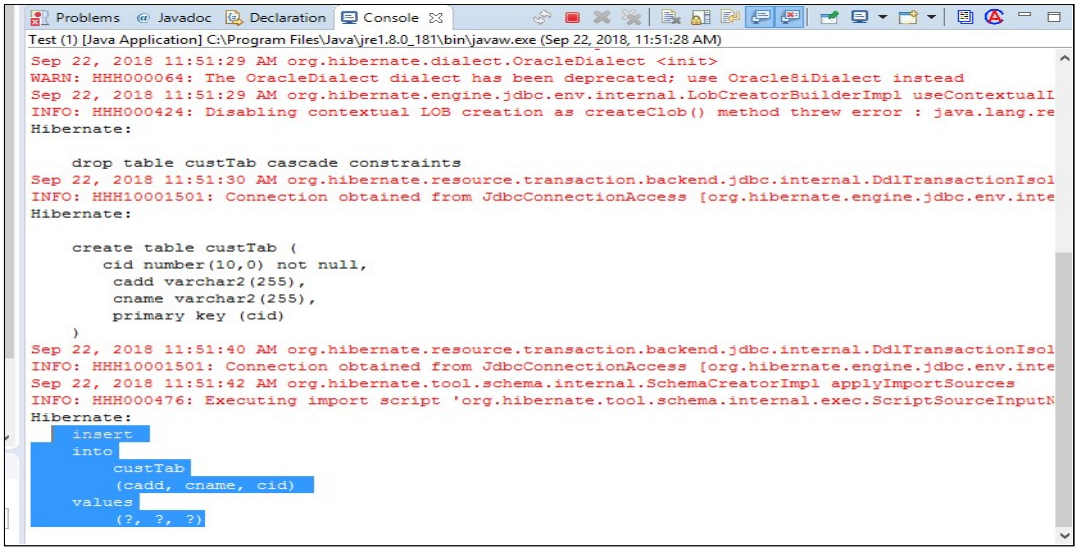
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">
oracle.jdbc.driver.OracleDriver</property>
```

```
<property name="hibernate.connection.url">
jdbc:oracle:thin:@localhost:1521:xe</property>
<property  name="hibernate.connection.username">system</property>
<property  name="hibernate.connection.password">system</property>
<property name="hibernate.dialect"> org.hibernate.dialect.OracleDialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.Customer"/>
</session-factory>
</hibernate-configuration>
```

2. Test class://

```
package com.app;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Test {
	public static void main(String[] args) {
		Configuration cfg=new Configuration();
		cfg.configure();
		SessionFactory sf=cfg.buildSessionFactory();
		Session ses=sf.openSession();
		Transaction tx=ses.beginTransaction();
		Customer cust=new Customer();
		cust.setCustId(1003);
		cust.setCustName("Mohan");
		cust.setCustAdd("BIHAR SHARIF");
		ses.save(cust);
		tx.commit();
		ses.close();
	}
}
```

OutPut:-



Session(I) in hibernate:-

Session is an interface defined in Hibernate Framework. It is used to perform any operation in Hibernate.

New Framework-5.x and JDK 1.7 supports auto close of session using try with resource.

Syntax:

```
Try(Session ses= sf.openSession()){
      ……..//logic
        }catch(Exception e)
 {
     ……..
  }
```

*) it means java executes code like:finally { ses.close();}

2. Session(I) has extended java.lang.AutoClosebale. So, try-with-resource is working.

3. Session (I) support operations both select and non-seelct.

Given few examples as:

Non-select: **Save(obj):Serializable**
Update(obj):void
Delete(obj):void
saveOrUpdate(obj):void

Select: **get(T.class,Id):T obj**
load(T.class,Id):T obj

Here T=Model class Name=type

Above operations are one row ⇔ one object (Single row) operations, it means at a time delete one row , select one row, update one row ……etc.

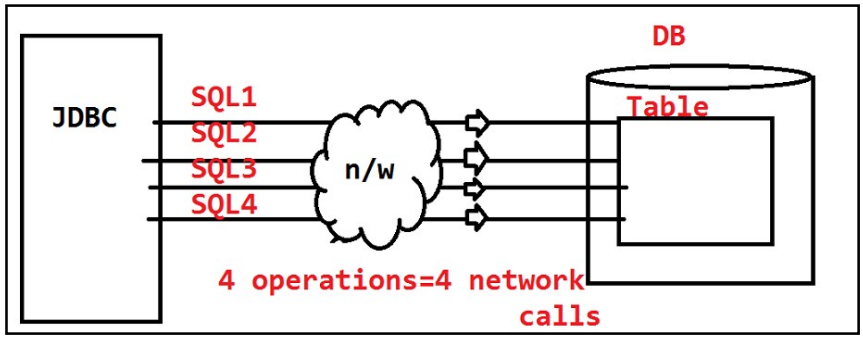
To perform multi-row operations Session(I) has provided

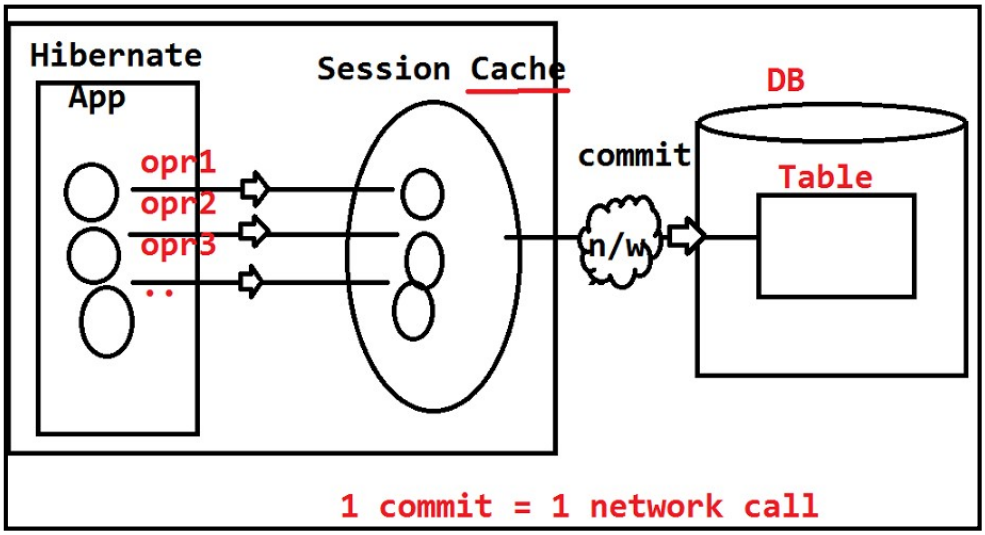
Query

Criteria APIs

***Transaction (tx) must be created using Session(I) in hibernate for non-select operations. It is not applicable for select operations.**

Cache :- it is a Temporary memory implemented by Hibernate framework(not in JDBC) which reduces network calls between Application and Database , that improves performance.
Hibernate supports 2 type of Cache Management.
Session Cache (First Level)
Session Factory Cache(Second level)
(disabled by default),Which is handled by programmer.
*** **Session Cache also has sup-type Query Cache.**
----------**JDBC VS Hibernate in Cache**----------





Session Operations:-

1. **Save(obj):-** this method is given from Session (I). It is used to convert model class object to DB Table Row.

- Model class must have at least **@Entity**, **@Id** Annotations
- @Table, @Column are optional

*** **To avoid variable mapping column, use @Transient**

(Package: **javax.persistence**)

a). We can't save new row into DB table with existed **Primary key** Id using Save() method Hibernate will throw **ConstraintViolationException**

==========code==========

Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import  javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
// ctlr+shift+o(imports)
@Entity
@Table(name="empTab")
publicclass Employee {
	@Id
	@Column(name="eid")
	privateintempId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	privatedoubleempSal;
	@Transient
	privateintempCode;
	public Employee() {
		super();
	}
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
```

```
	}
	publicint getEmpCode() {
		returnempCode;
	}
	publicvoid setEmpCode(intempCode) {
		this.empCode = empCode;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", empCode=" + empCode
				+ "]";
	}
}
```

Configuration File

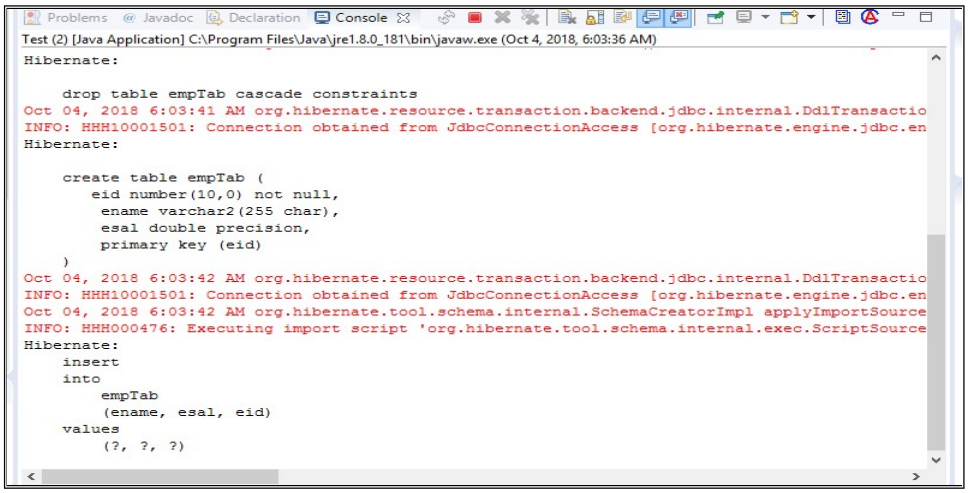
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDri
ver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</
property>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">system</property><propertyname="
hiberante.dialect"> org.hibernate.dialect.OracleDialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">create</property>
<mappingclass="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

Test Class:

```
package com.app.model;
import  org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
publicclass Test {
	publicstaticvoid main(String[] args) {
		Configuration cfg=new Configuration();
		cfg.configure();
		SessionFactory sf=cfg.buildSessionFactory();
		Session ses=sf.openSession();
		Transaction tx=ses.beginTransaction();
    Employee e=new Employee();
```

```
e.setEmpId(100);
e.setEmpName("Ram");
e.setEmpSal(22.2);
e.setEmpCode(2222);
ses.save(e);
tx.commit();
ses.close();
        }
}
```

Output:-



Save() and persist()

Save() – Persist the given transient instance, first assigning a generated identifier. (Or using the current value of the identifier property if the assigned generator is used.) This operation cascades to associated instances if the association is mapped with cascade="save-update".
persist() – Make a transient instance persistent. This operation cascades to associated instances if the association is mapped with cascade="persist". The semantics of this method are defined by JSR-220.

save()	persist()
It stores object in database	It also stores object in database
It return generated id and return type is serializable	It does not return anything. Its void return type.
It can save object within boundaries and outside boundaries	It can only save object within the transaction boundaries

save()	persist()
It will create a new row in the table for detached object	It will throw persistence exception for detached object
It is only supported by Hibernate	It is also supported by JPA

Update(obj):void:-

This method is used to update all columns data based on Primary key column.
Ex: if Table has 50 columns, in that one is Primary key, then is we use ses.update(obj), it will update 49 columns based one 1 Primary key Column.
If Given Primary key Id is not found then Hibernate throws (Unknown object State)
StateObjectStateException: Row was updated or deleted by another transaction (or unsaved-value mapping….)
Ex:……:code:----

Model class:

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
// ctlr+shift+o(imports)
@Entity
@Table(name="empTab")
public class Employee {

	@Id
	@Column(name="eid")
	private int empId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	private double empSal;
	@Transient
	private int empCode;
	public Employee() {
		super();
	}
	public int getEmpId() {
		return empId;
```

```
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	public int getEmpCode() {
		return empCode;
	}
	public void setEmpCode(int empCode) {
		this.empCode = empCode;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName + ", empSal="
+ empSal + ", empCode=" + empCode
					+ "]";
	}
	}
```

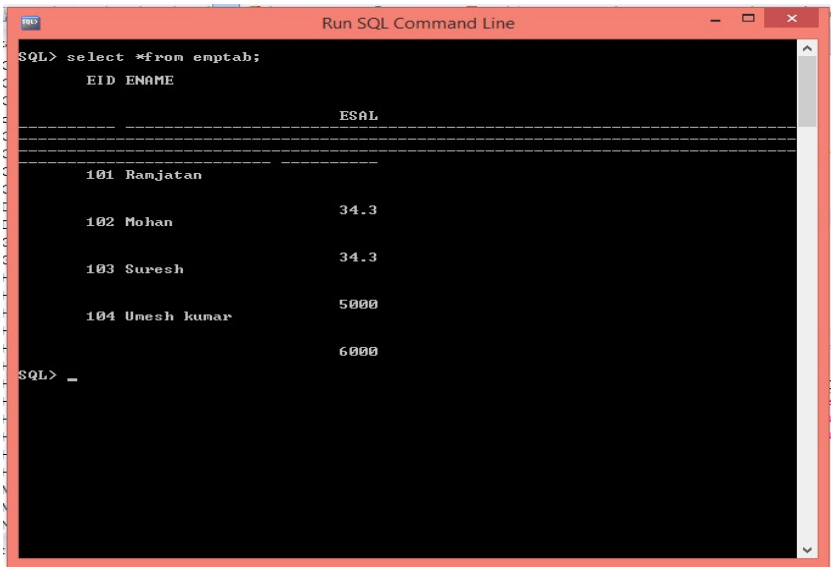
Configuration file

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDri
ver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</pr
operty>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hiberante.dialect">org.hibernate.dialect.OracleDialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

Test Class:

//insert one employee object, then

```
package com.app.model;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Test {
public static void main(String[] args) {
                Configuration cfg=new Configuration();
                cfg.configure();
                SessionFactory sf=cfg.buildSessionFactory();
                Session ses=sf.openSession();
                Transaction tx=ses.beginTransaction();
                Employee e =new Employee();
                e.setEmpId(101);
                e.setEmpName("Amit");
                e.setEmpSal(6000);
                ses.update(e);
                //ses.save(e);
                tx.commit();
        }
}
```



After Update Method

Hibernate Save()/update() for example

1. **Model class:-**

```
package com.app;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
import javax.persistence.Table;
@Entity
@Table()
publicclass Student {
@Id
@Column(name="sid")
privateintstdId;
@Column(name="sname")
private String stdName;
@Column(name="sfee")
privatedoublestdFee;
@Column(name="age")
privateintAge;
public Student() {
	super();
}
publicint getStdId() {
	returnstdId;
}
publicvoid setStdId(intstdId) {
	this.stdId = stdId;
}
public String getStdName() {
	returnstdName;
}
publicvoid setStdName(String stdName) {
	this.stdName = stdName;
}
publicdouble getStdFee() {
	returnstdFee;
}
publicvoid setStdFee(doublestdFee) {
	this.stdFee = stdFee;
}
publicint getAge() {
	returnAge;
}
publicvoid setAge(intage) {
	Age = age;
}
@Override
public String toString() {
	return"Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" + stdFee + ",
Age=" + Age + "]";
```

```
}
}
```

2. Configuration file//hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property
name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<property name="hibernate.connection.username">system</property>
<property name="hibernate.connection.password">system</property>
<property name="hiberante.dialect"> org.hibernate.dialect.OracleDialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.Student"/>
</session-factory>
</hibernate-configuration>
```

3. Test class:

```
package com.app;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Test {
        public static void main(String[] args) {
                Configuration cfg=new Configuration();
                cfg.configure();
                SessionFactory sf=cfg.buildSessionFactory();
                Session ses=sf.openSession();
                Transaction tx=ses.beginTransaction();
                 Student st=new Student();
                 st.setStdId(1001);
                 st.setStdName("Ramjatan");
                 st.setStdFee(22.2);
                 st.setAge(24);
                 ses.save(st);
                tx.commit();
                ses.close();
        }
```

}

----Generated SQL Looks like:--------

Update emptab set ename=?,esal=? Were eid=?

saveOrUpdate(obj):void:-

This method will execute “Select ..... Where PKColumn=?” first in DB

*** if given Primary key row exist then Hibernate calls update else Hibernate calls insert (save).

Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
// ctlr+shift+o(imports)
@Entity
@Table(name="empTab")
publicclass Employee {
	@Id
	@Column(name="eid")
privateintempId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")

	privatedoubleempSal;
	public Employee() {
		super();
	}
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
```

```
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + "]";
	}
	}
```

Cfg file are same as before

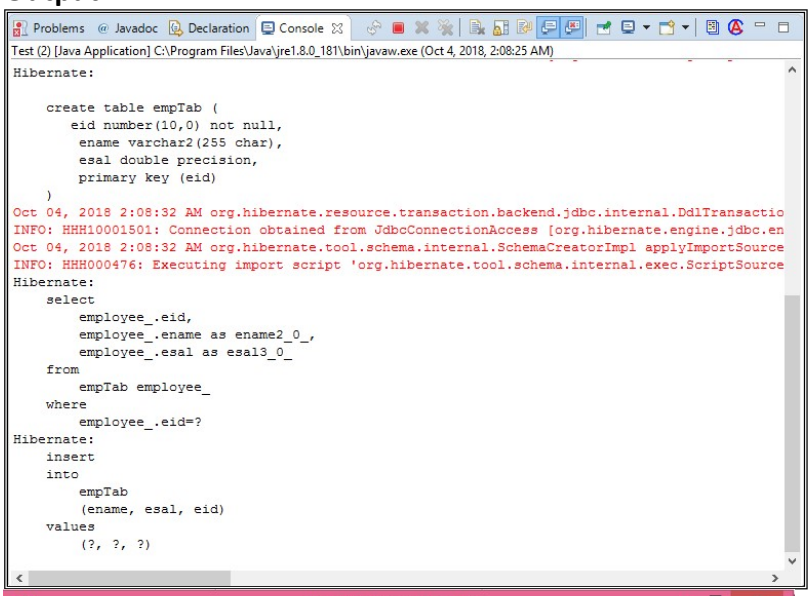
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hiberante.dialect"> org.hibernate.dialect.OracleDialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Employee"/>
</session-factory>
	</hibernate-configuration>
```

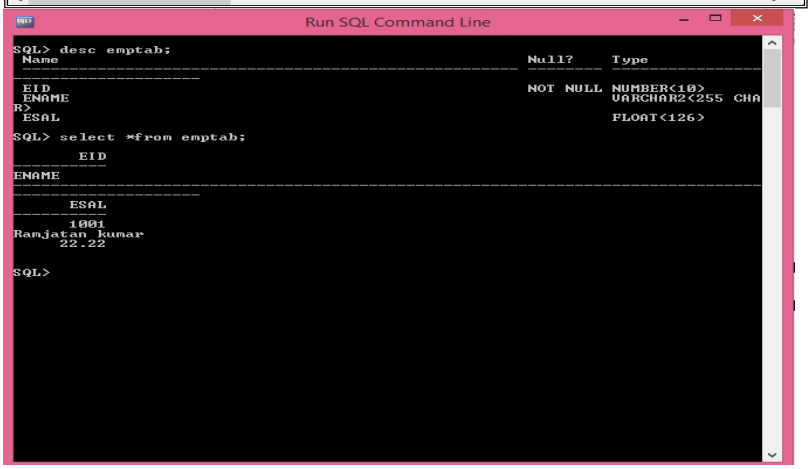
Test class:

```
package com.app.model;
import  org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
publicclass Test {
	publicstaticvoid main(String[] args) {
		Configuration cfg=new Configuration();
		cfg.configure();
		SessionFactory sf=cfg.buildSessionFactory();
		Session ses=sf.openSession();
		Transaction tx=ses.beginTransaction();
		Employee e=new Employee();
		e.setEmpId(1001);
		e.setEmpName("Ramjatan kumar");
```

```
            e.setEmpSal(22.22);
            ses.saveOrUpdate(e);
            tx.commit();
            ses.close();
        }
    }
```

Output:-





ses.delete(obj):void:-this method is used to delete one row from DB Table based on Primary key only.

a. This method will take input as "model class object having primary key value"
b. First it will execute select query and loads data into cache(I-Level) if exist then delete query will be called, else does nothing.

----------Code----------

Model class + mapping

```
package com.app.model;
import javax.persistence.Column;
```

```
import javax.persistence.Entity;
import  javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Transient;
// ctlr+shift+o(imports)
@Entity
@Table(name="empTab")
publicclass Employee {
	@Id
	@Column(name="eid")
	privateintempId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	privatedoubleempSal;
	public Employee() {
		super();
	}
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	publicint getEmpCode() {
		returnempCode;
	}
	publicvoid setEmpCode(intempCode) {
		this.empCode = empCode;
	}
	@Override
	public String toString() {
```

```
return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", empCode=" + empCode
                + "]";
    }
    }
```

Configuration file

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDri
ver</property>
<propertyname="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:xe</
property>
<propertyname="hibernate.connection.username">system</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hiberante.dialect">
org.hibernate.dialect.OracleDialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

Test class

```
package com.app.model;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Test {
    public static void main(String[] args) {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session ses=sf.openSession();
        Transaction tx=ses.beginTransaction();
        Employee e =new Employee();
        e.setEmpId(102);
        e.setEmpName("Amit");
        e.setEmpSal(6000);
        //ses.update(e);
        //ses.save(e);
        ses.delete(e);
```

```
                tx.commit();
        }
}
```

Output:-

enum(JDK 1.5):- enum is keyword in java, introduced in jdk1.5

- it is used to create programmer/User Defined data Type with set of possible values.

Enum:- set of possible values

Syntax:-

Enum<enum_name> { // properties…}

Ex: enum ExamResult {**PASS, FAIL,ABS** }

***)** Every enum property is buy default

Public static final.

It is only variable and value.

*) enum property can be accessed using direct way enumName.property or use static import to save memory.

1. Create in Project > Src > right click > new > enum > enter pack and enum name

Ex:- Package: com.app.one

Name : ExamResult

.....code.....

```
package comm.app.one;
public enum ExamResult {
PASS, FAIL,ABS
}
```

2.using enum (Direct access)

```
Package com.app.test;
Import com.app.one.ExamREsult;
Public class Test {
Public static void main(String arr[]){
System.out.println(ExamREsult.PASS);
System.out.println(ExamResult.FAILL);
}
}
```

*** static import:** This feature is used to save memory by loading only required static properties/method in to JVM(RAM)

> Normal import loads static and instance data into JVM even we use it or not.
>But static iimport will not load all.

Syntax:

1) Import static pack.cls.property
2) Import static pack.cls.method
3) Import static pack.enum.property

.....reading enum using static import.....

```
Package com.app.test;
Import static com.app.one.ExamResult.FAILL;
Public class Test {
psv main(String arr[]){
Sysout(FAILL);
```

*) To get all possible values in a enum call default method values() which returns same enum type array.

*) use ordinal (): int method to find index nmber of enum property (starts from zero by default)

.....code :Test class.....

```
package com.app.test;
Import com.app.one.ExamResult;
public class Test {
psv main(String arr[])
ExamResult[] vals=ExamResult.values();
For(examResult e:vals){
Sysout(e+","+e.ordinal());
}
}
}
```

❖ **Using MySQL DB in Hibernate:-**

Softwares:- MySQL 5.x + SQL

To migrate from one DB to another DB in hibernate effects only code change in Hibernate Configuration file. Model class and test classes remains same (No code Change).

*) In cfg file modify 5 keys data. Those are driver_class, url,username,password and dialect.

>> keys-vals for MySQL DB

Driver_class=**com.mysql.jdbc.Driver**
url=**jdbc:mysql://localhost:3306/test**
username:**system**
password:**system**
dialect=**org.hibernate.dialect.MySQL5Dialect**
***** In build path add mysql-connector-5.1.jar**

❖ **DATE and TIME in Hibernate:-**

Hibernate has provided enum to handle Date and time concept as column data named as "TemporalType" given in Package:**javax.persistence**

Code looks like:-

```
Enum TemporalType {
        DATE,TIME,TIMESTAMP
        }
```

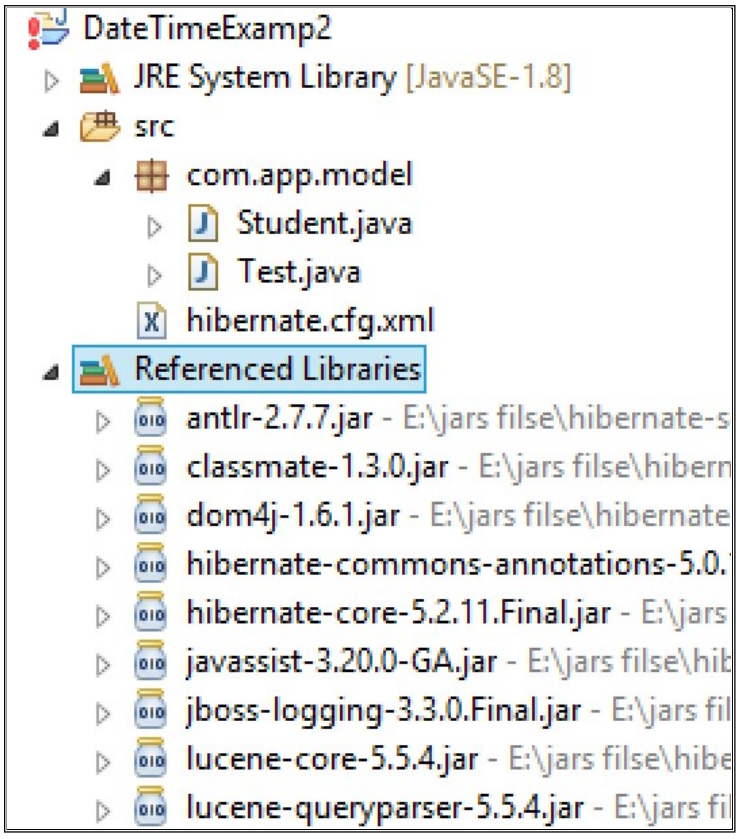
Example:-

Output----------option to choose

19/07/2018 DATE
6:51 PM TIME
19/07/2018 6:51 PM TIMESTAMP

- Use java.util.DATE or java.sql.Date as variable type

==========code==========



hibernate Date and Time

1. Model class:-

```
package com.app.model;
import java.util.Date;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
@Entity
@Table(name="StdTable")
public class Student {
	@Id
	@Column(name="sid")
	private int stdId;
	@Column(name="sname")
	private String stdName;
	@Temporal(TemporalType.DATE)
	@Column(name="date")
	private Date dt1;
	@Temporal(TemporalType.TIME)
	@Column(name="time")
	private Date dt2;
	@Temporal(TemporalType.TIMESTAMP)
	@Column(name="daytime")
	private Date dt3;
	public Student() {
		super();
	}
	public int getStdId() {
		return stdId;
	}
	public void setStdId(int stdId) {
		this.stdId = stdId;
	}
	public String getStdName() {
		return stdName;
	}
	public void setStdName(String stdName) {
		this.stdName = stdName;
	}
	public Date getDt1() {
		return dt1;
```

```
	}
	public void setDt1(Date dt1) {
		this.dt1 = dt1;
	}
	public Date getDt2() {
		return dt2;
	}
	public void setDt2(Date dt2) {
		this.dt2 = dt2;
	}
	public Date getDt3() {
		return dt3;
	}
	public void setDt3(Date dt3) {
		this.dt3 = dt3;
	}
	@Override
	public String toString() {
		return "Student [stdId=" + stdId + ", stdName=" + stdName + ", dt1=" + dt1 + ",
dt2=" + dt2 + ", dt3=" + dt3
				+ "]";
	}
}
```

Configuration file hibernate.cfg.xml

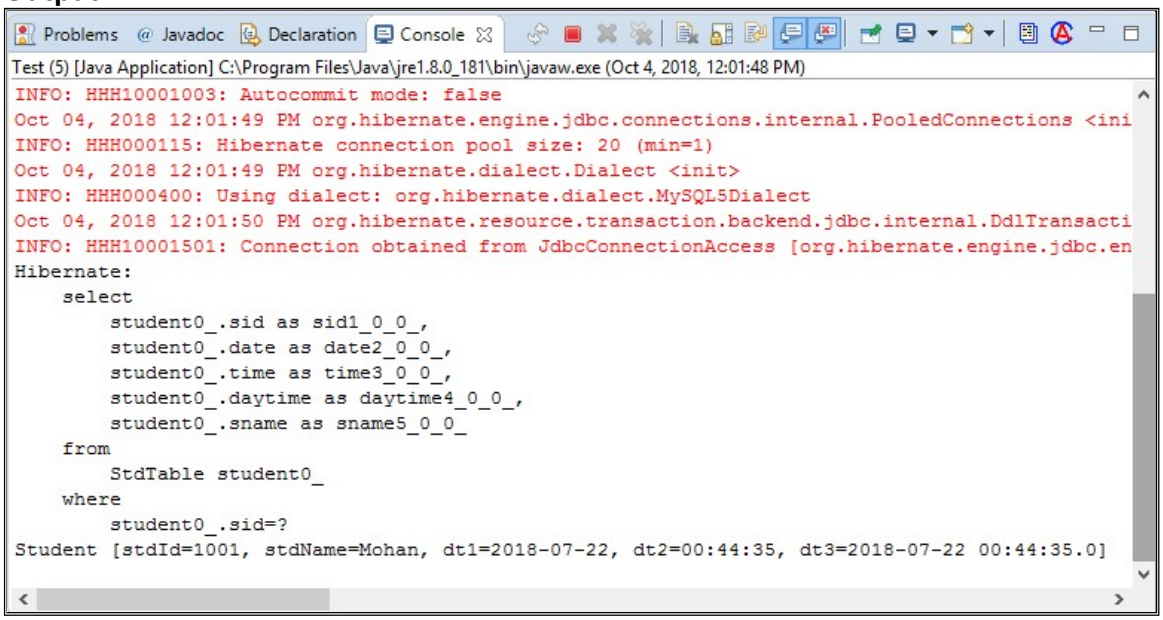
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/test</pro
perty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Student"/>
</session-factory>
</hibernate-configuration>
```

Test class:

```
package com.app.model;
```

```
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
publicclass Test {
publicstaticvoid main(String[] args) throws Exception {
	Configuration cfg=new Configuration();
	cfg.configure();
	SessionFactory sf=cfg.buildSessionFactory();
	Session ses=sf.openSession();
	Transaction tx=ses.beginTransaction();
	//Student st=new Student();
	//Student st=ses.get(Student.class, 1001);
Student st=(Student) ses.load(Class.forName("com.app.model.Student"),1001);
	System.out.println(st);
	ses.close();
//insert record
	/*st.setStdId(1001);
	st.setStdName("Mohan");
	st.setDt1(new Date());
	st.setDt2(new Date());
	st.setDt3(new Date());
	ses.save(st);
	*/
	tx.commit();
}
}
```

Output:-



- **WORKING WITH CLOB AND BLOB TYPES IN HIBERNATE:-**

  Database support BLOB and CLOB to store large object in tables

  BLOB:- it is used to store large objects like audio, video,images,files,like documents PPT,excel files etc….

  it also support pure text in hibernate code should be written as

@Lob
Private byte[] img;
CLOB: Character Large Object used to store only text data. Code is
@Lob
Private char[] mydata;

fis.available():- this method return int(file size in bytes)
str.toCharArray():- this method converts String to char[]
==========code==========

hibernate CLOB And BLOB

1. Model class:-

```
package com.app.model;
import java.util.Arrays;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Lob;
import javax.persistence.Table;
@Entity
@Table(name="studTab")
public class Student {
	@Id
	@Column(name="sid")
	private int stdId;
	@Column(name="sname")
	private String stdName;
	@Column(name="sfee")
	private double stdFee;
	@Lob
	@Column(name="img")
	private byte[] simg;
	@Lob
	@Column(name="doc")
	private char[] stxt;
	public Student() {
		super();
	}
	public int getStdId() {
		return stdId;
	}
	public void setStdId(int stdId) {
		this.stdId = stdId;
	}
```

```
	public String getStdName() {
		return stdName;
	}
	public void setStdName(String stdName) {
		this.stdName = stdName;
	}
	public double getStdFee() {
		return stdFee;
	}
	public void setStdFee(double stdFee) {
		this.stdFee = stdFee;
	}
	public byte[] getSimg() {
		return simg;
	}
	public void setSimg(byte[] simg) {
		this.simg = simg;
	}
	public char[] getStxt() {
		return stxt;
	}
	public void setStxt(char[] stxt) {
		this.stxt = stxt;
	}
	@Override
	public String toString() {
return "Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" + stdFee + ", simg="+
Arrays.toString(simg) + ", stxt=" + Arrays.toString(stxt) + "]";
	}
}
```

2 Configuration file //hibernate.cfg.xml

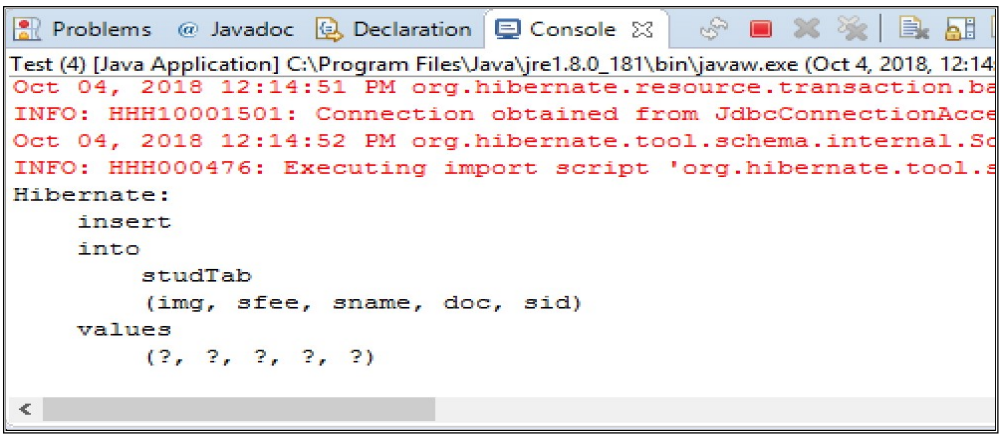
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect"> org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
```

```
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.model.Student"/>
</session-factory>
</hibernate-configuration>
```

3 Test class:-

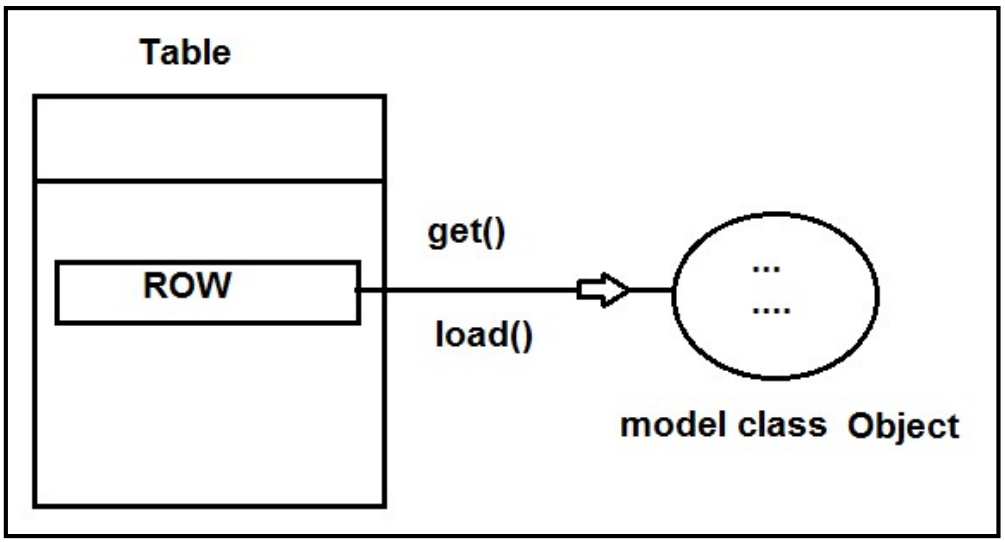
```
package com.app.model;
import java.io.FileInputStream;
importjava.io.FileNotFoundException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
publicclass Test {
publicstaticvoid main(String[] args) throws Exception {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session ses=sf.openSession();
        Transaction tx=ses.beginTransaction();
        Student st=new Student();
        st.setStdId(1001);
        st.setStdName("Ram");
        st.setStdFee(22.2);
        FileInputStream fis=new FileInputStream("D:\\Ram.jpg");
        byte[] arr=newbyte[fis.available()];
        fis.read(arr);
        st.setSimg(arr);
        String str="Welcome to Ramjatan ";
        char[] arr1=str.toCharArray();
                        st.setStxt(arr1);
                        ses.save(st);
                        tx.commit();
                        ses.close();
        }
}
```

Output:-



Selecting/Fetching Data From DB Table using Hibernate:-

Use get() or load() methods to fetch one row data based on Primary key. It will return in Object format.



1. On calling get() method,
   - Hibernate search for Row based on Primary key
   - If found, Hibernate create object and set the data taken from Table row.
   - So we need to pass Class type input (Java.lang.Class) to get method.

Class (java.lang):- This is used to give information of a class like name, package, const, variables, methods etc. **to JVM /Framework/Containers**.

Syntax:- to create Class obj:-

a. **Class c=Class.forName(" ------");**
b. **Class c= ------.class**

This method is used to fetch one row to one object based on primary key.

Consider Emplyee as Example then code can be written as

Exa1:-

Employee e=ses.get(Class.forName("com.app.Employee"),10);

Exam2:-

Employee e=ses.get(Employee.class,10);

- If Hibernate finds row based on Primary given then row will be converted to object else null value is returned.

code

1. Model class , mapping
2. Cfg file are same as before
3. Test class ://cfg,sf,ses

Employee e ses.get(Employee.class,101);
System.out.println(e);
Ses.close();

AutoBoxing/UnBoxing :- it is process of converting primitive type to object (AutoBoxing) find Object to primitive (AutoUnBoxing) without writing external converter code.

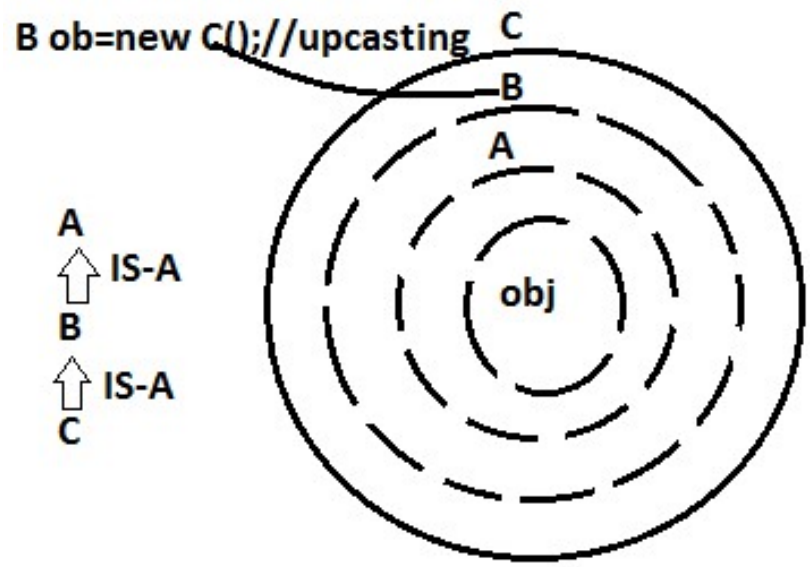
Exa:-

```
Int sid=5;
Integer s=sid;///AutoBoxing
Int p=s;//AutoUnboxing
```

UpCasting/Downcasting:-

UpCasting:- Converting Sub Type object to it's Super Type is called as UpCasting.

Consider blow example:-



To do upcasting classes must be in Inheritance Reletion(IS-A)

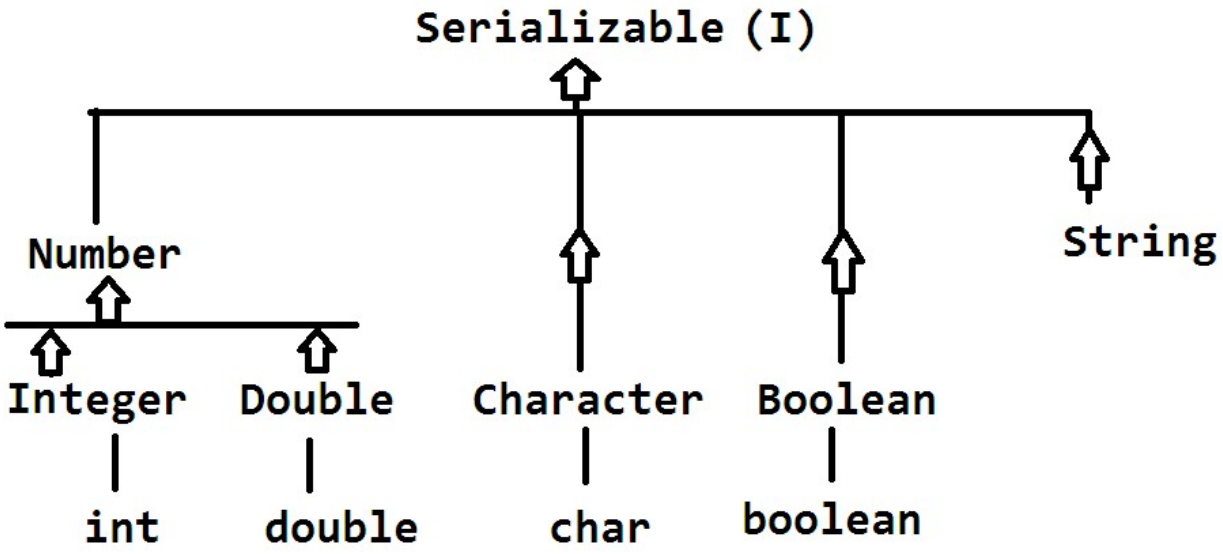
DownCasting:- Converting up casted object to subtype or previous Type is known as down casting.

To do downcasting class must be in (IS-A) and Object should already being Upcasted.

Q). **which datatype can be used as Primary key data Type in Hibernate?**

Ans.:- A class / Primitive which directly/indirectly connected to Serializable(I).

Few Are: **int, Integer, Number,String**



In hibernate Serializable indicate Primary key

- In get method() 2nd parameter indicates Primarykey is Serializable. 1st parameter is model class details.

Exa:-
Employee e=ses.get(Employee.class,101);
Here 101 is int type converted to Integer (AutoBoxing) and casted to Number > Serializable.
So, finally 101 can be stored in Serializable.

LOAD() method:- This method also used to select one row data from DB table and converts to model class object.

Syntax:

Load(Class<T> Clz,
Serializable id):T Obj

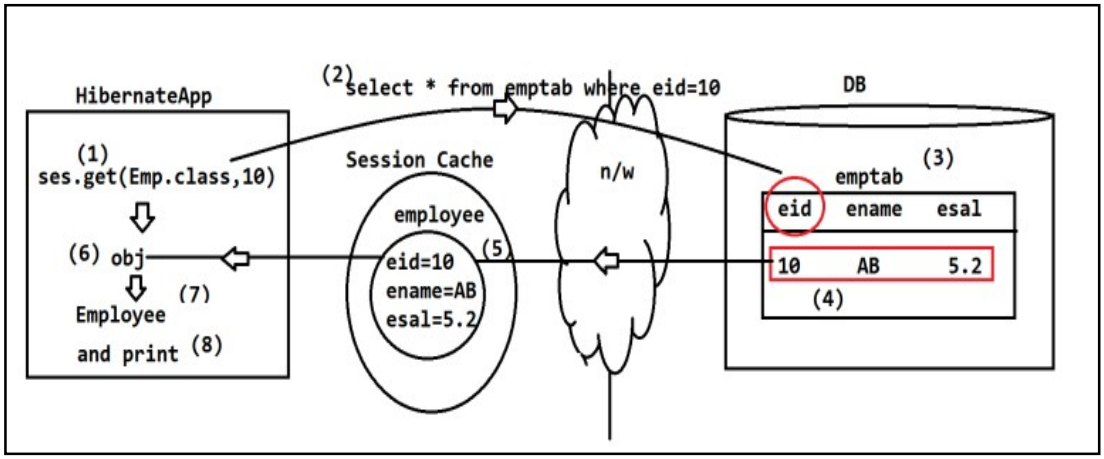
==========Example==========

1. Model class and mapping code
2. Cfg file are same as before
3. Test class:// cfg,sf,ses,

//save one object berfor this
Employee e=ses.load(Employee.class,100);
Sysout(e);
Ses.close();
*) if row 101 is not existed then **ObjectNOtfoundException** is thrown by Hibernate.

Get() execution flow:-

1. Call get method from hibernate method
2. On calling first time it will make network call by executing select query
3. In row exist it is converted to object and placed in Session Cache else object value is null.
4. Finally object return to Application.



Example for Get method

Model class:-

```
 package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="stdtab3")
publicclass Student {
	@Id
	@Column (name="sid")
	privateintstdId;
	@Column(name="sname")
	private String stdName;
	@Column(name="sfee")
	privatedoublestdFee;
	public Student() {
		super();
	}
	publicint getStdId() {
		returnstdId;
	}
	publicvoid setStdId(intstdId) {
		this.stdId = stdId;
	}
	public String getStdName() {
		returnstdName;
	}
	publicvoid setStdName(String stdName) {
		this.stdName = stdName;
```

```
	}
	publicdouble getStdFee() {
		returnstdFee;
	}
	publicvoid setStdFee(doublestdFee) {
		this.stdFee = stdFee;
	}
	@Override
	public String toString() {
		return"Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" + stdFee
+ "]";
	}
}
```

1. Configuration file // hibernate.cfg.xml

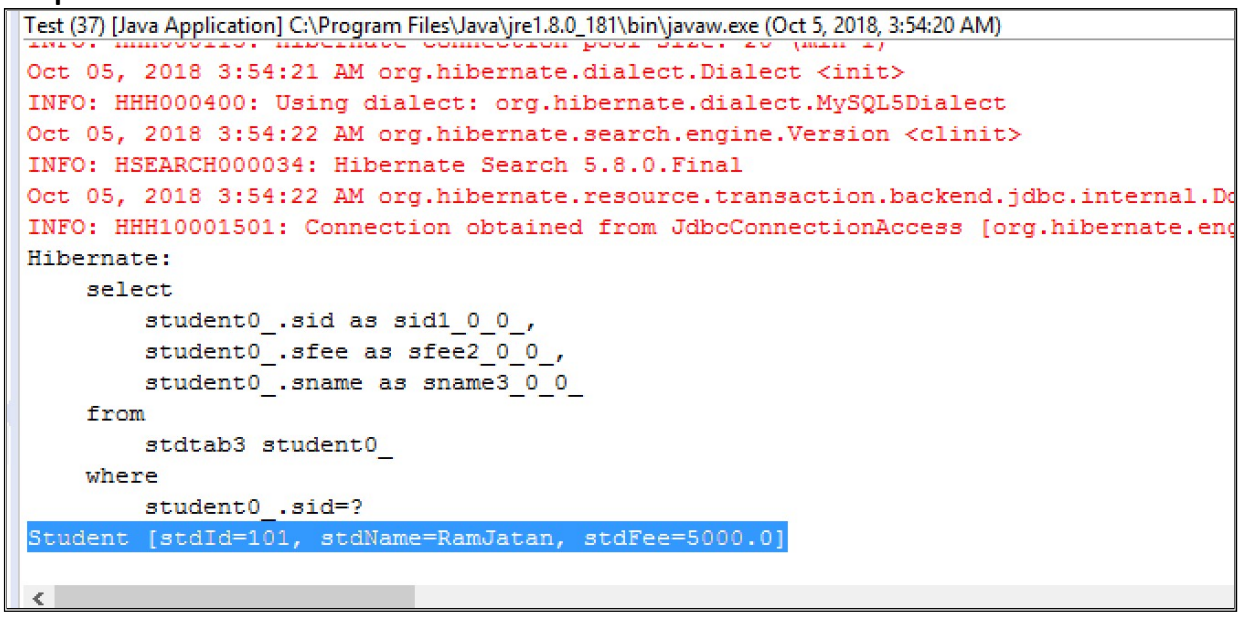
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Student"/>
</session-factory>
</hibernate-configuration>
```

Test class:

```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.app.model.Student;
public class Test {
public static void main(String[] args) throws Exception {
	Configuration cfg=new Configuration();
```

```
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session ses=sf.openSession();
/*Transaction tx=ses.beginTransaction();
        Student st=new Student();
        st.setStdId(102);
        st.setStdName("Mohan");
        st.setStdFee(6000);
        ses.save(st);
        tx.commit();
        ses.close();*/
        Student s=ses.get(Student.class, 101);
        System.out.println(s);
        ses.close();
}
}
```

Output:-

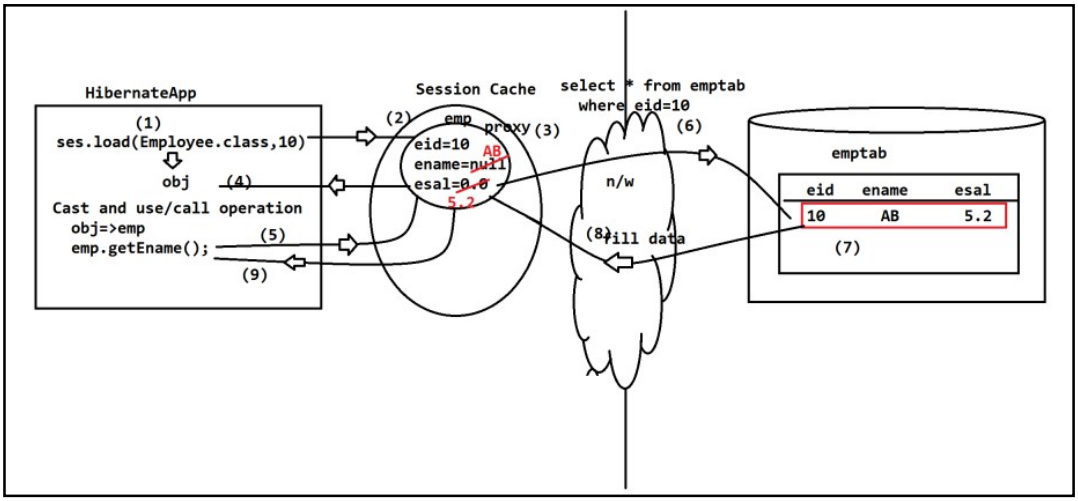


Load() Execution flow:-

1. Call load method in hibernate application
2. to application communication with cache
3. Cache create proxy object of model class and it will be set primary key value.
   Proxy means dummy object created by hibernate.
4. This proxy is return back to application
5. On performing any operations over object (Print or any get method call) then it will communicate to proxy for data.

6. Hibernate application makes Networks call by executing select query.
7. If row exist it is return back to Cache as object data.
8. Data return to applications.

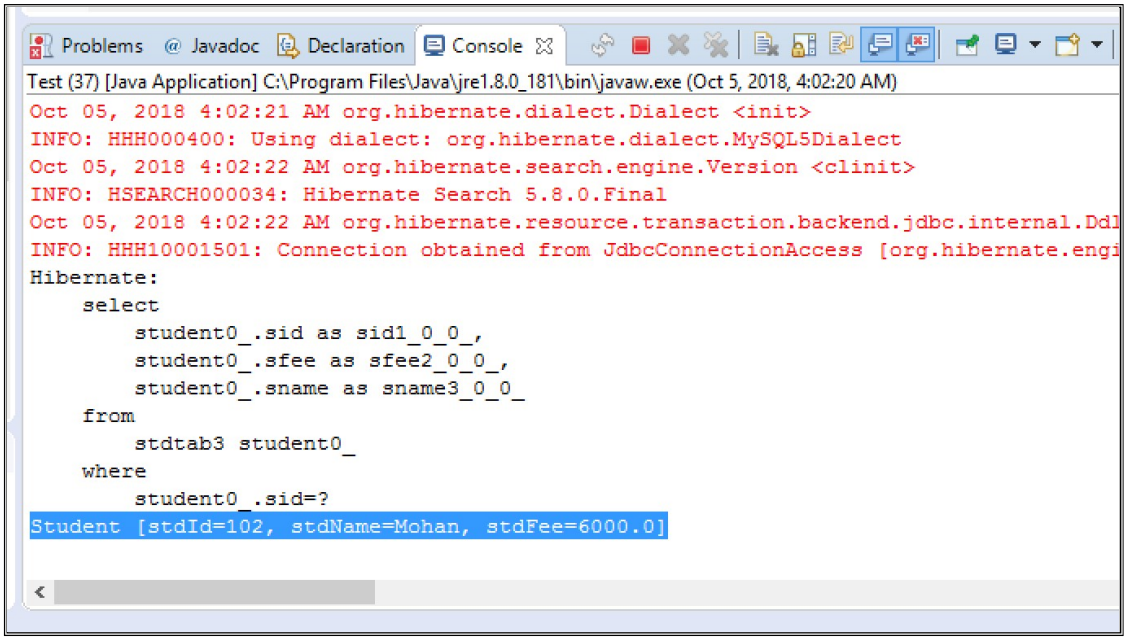
Test class :



1. **Model class (same as get method)**
2. **Configuration file (hibernate.cfg.xml)==same as get method code)**

```
package com.app.model;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Test {
public static void main(String[] args) throws Exception {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session ses=sf.openSession();
        Transaction tx=ses.beginTransaction();
Student st=(Student) ses.load(Class.forName("com.app.model.Student"),102);
        System.out.println(st);
        ses.close();
        }
}
```

Output:-



get()	load()
get hits DB on call	load hits cache first
get return null if record not exist	load throws Exception if record not exist (ObjectNotFound)

Collection Mapping in Hibernate:

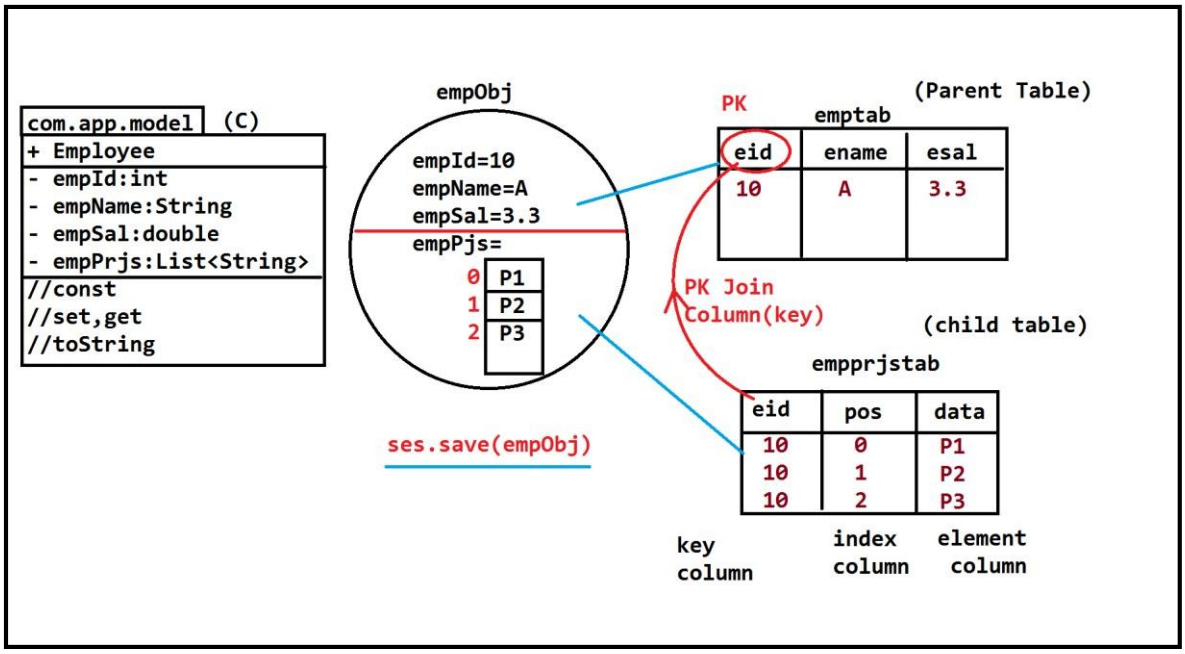
Hibernate framework supports working with collection like **List,Map,Set**.....etc. **for every collection** hibernate creates one new child table which will be linked with parent Table. These collections mapping are two types in hibernate.

1. Index-based collections **:- List,Map, and Id-Bag**
2. Non=index based collections :- **Set,Bag**
   - For every child table **min=2** and **max=3** columns are provide by hibernate. Those are:
     Key=column= PK-FK link column
     Index column= Position number element column=data of collection.

Consider List Type Example below,

For List hibernate creates a child table with 3 columns (index based)

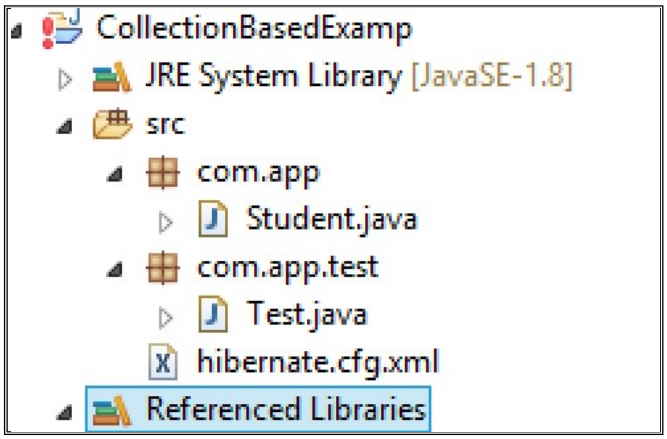
Design with Table Operation



Here, Hibernate has provided collection mapping annotation: ie
@ElementCollection it must be applied over collection type variable in model class.

- If no child table details are provided then hibernate creates child table with it's own column, table names.
- To provide child table details use annotation **@Collectiontable**
- **** To provide index number for List type use **@OrderColumn**

==========code ==========



1. **Model class**

```
package  com.app;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CollectionTable;
import javax.persistence.Column;
```

```
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OrderColumn;
import javax.persistence.Table;
@Entity
@Table(name="StudTab")
public class Student {
@Id
@Column(name="sid")
private int stdId;
@Column(name="sname")
private String stdName;
@ElementCollection
@CollectionTable(name="stdproj", joinColumns=@JoinColumn(name="sid") )
@OrderColumn(name="pos")
@Column(name="Data")
	private List<String>stdData=new ArrayList<String>();
public Student() {
super();
}
public int getStdId() {
return stdId;
}
public void setStdId(int stdId) {
this.stdId = stdId;
}
public String getStdName() {
return stdName;
}
	public void setStdName(String stdName) {
this.stdName = stdName;
}
public List<String> getStdData() {
return stdData;
}
	public void setStdData(List<String> stdData) {
this.stdData = stdData;
}
@Override
public String toString() {
	return "Student [stdId=" + stdId + ", stdName=" + stdName + ", stdData=" + stdData + "]";
}
```

```
}
```

Configuration file (hibernate.cfg.xml)

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

<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.Student"/>
</session-factory>
</hibernate-configuration>
```

Test class:-

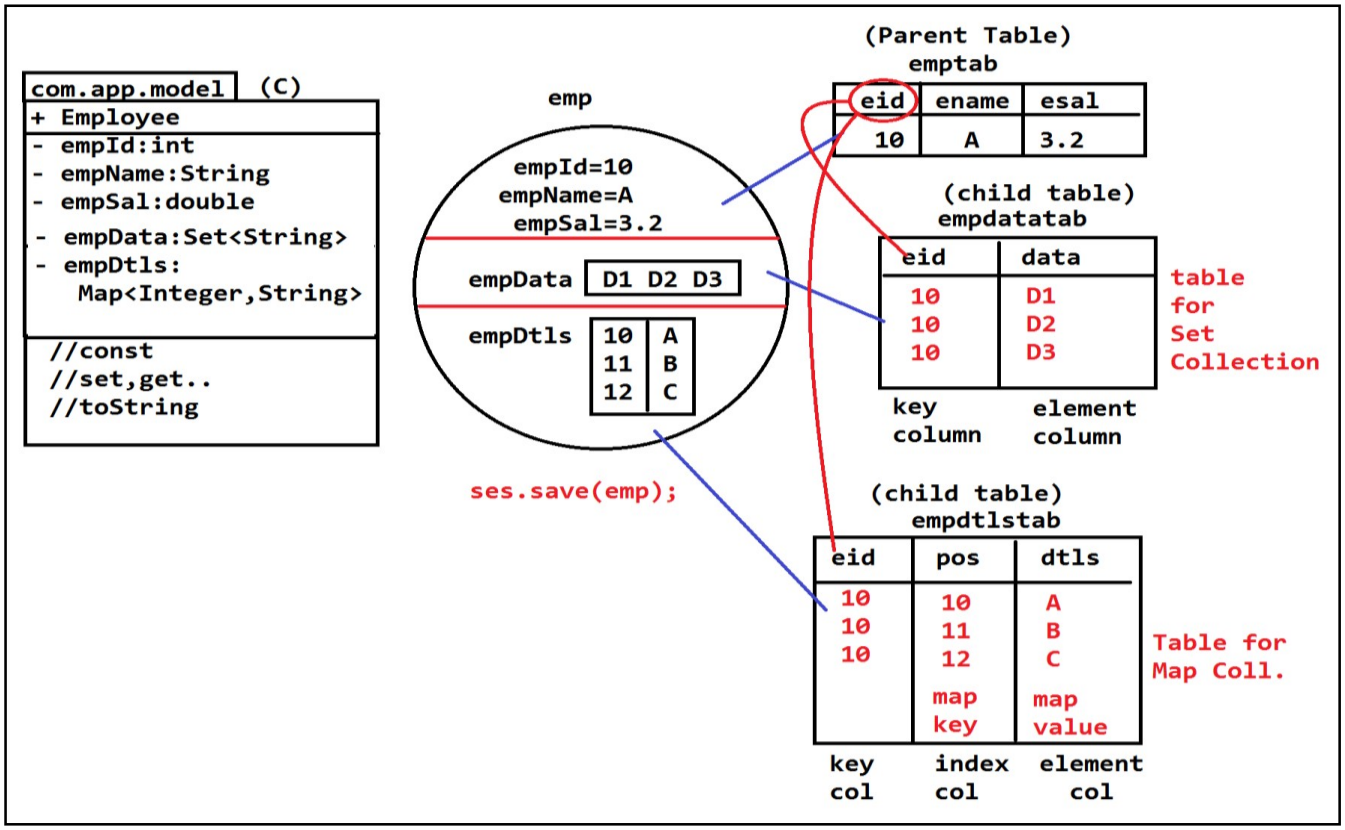
```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.app.Student;
public class Test {
        public static void main(String[] args) {
                Configuration cfg=new Configuration();
                cfg.configure();
                SessionFactory sf=cfg.buildSessionFactory();
                Session ses=sf.openSession();
                Transaction tx=ses.beginTransaction();

                Student st=new Student();
                st.setStdId(1001);
                st.setStdName("Ram");
                st.getStdData().add("D1");
                st.getStdData().add("D2");
                ses.save(st);
                tx.commit();
```

```
                ses.close();
        }
}
```

Output:-

```
INFO: HHH000476: Executing import
Hibernate:
    insert
    into
        StudTab
        (sname, sid)
    values
        (?, ?)
Hibernate:
    insert
    into
        stdproj
        (sid, pos, Data)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        stdproj
```

Example for Set and Map Collection:-



SET AND MAP

==========Code==========

1. **Model class**

```
package com.app;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;
import java.util.TreeMap;
import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.MapKeyColumn;
import javax.persistence.Table;
@Entity
@Table(name="custTab")
public class Customer {
	@Id
	@Column(name="cid")
	private int custId;
	@Column(name="cname")
	private String custName;
	@ElementCollection
	@CollectionTable(name="custData",
				joinColumns=@JoinColumn(name="cid")
					)
	@Column(name="data")
	private Set<String>custData=new HashSet<String>();
	@ElementCollection
	@CollectionTable(name="mapTab",
				joinColumns=@JoinColumn(name="cid")
					)
	@MapKeyColumn(name="pos")
	@Column(name="data")
	private Map<Integer,String>Data=new TreeMap<Integer,String>();
	public Customer() {
		super();
	}
	public int getCustId() {
		return custId;
	}
```

```
	public void setCustId(int custId) {
		this.custId = custId;
	}
	public String getCustName() {
		return custName;
	}
	public void setCustName(String custName) {
		this.custName = custName;
	}
	public Set<String> getCustData() {
		return custData;
	}
	public void setCustData(Set<String> custData) {
		this.custData = custData;
	}
	public Map<Integer, String> getData() {
		return Data;
	}
	public void setData(Map<Integer, String> data) {
		Data = data;
	}
	@Override
	public String toString() {
		return "Customer [custId=" + custId + ", custName=" + custName + ", custData=" +
custData + ", Data=" + Data
				+ "]";
	}
}
```

1. **Configuration file (hibernate.cfg.xml)**

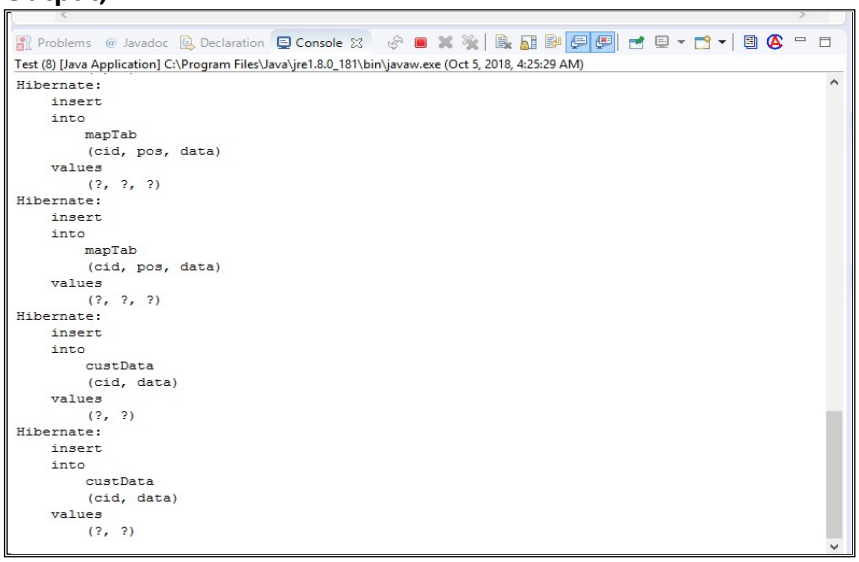
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
```

```
<mapping class="com.app.Customer"/>
</session-factory>
</hibernate-configuration>
```

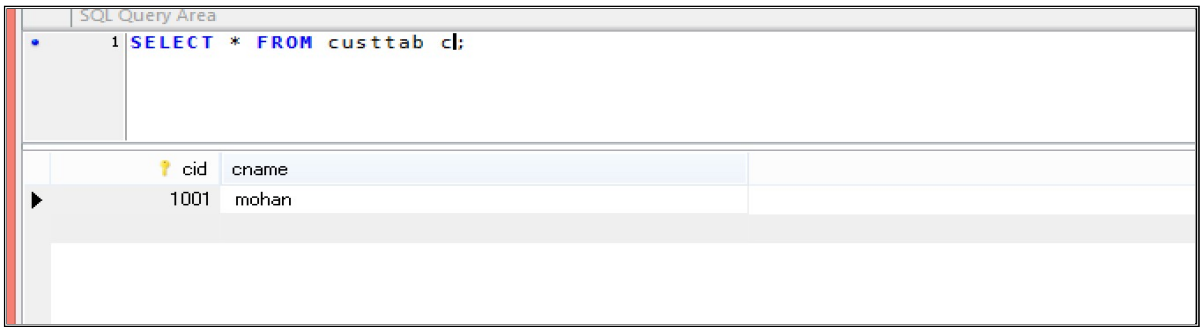
Test class:-

```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import com.app.Customer;
public class Test {
public static void main(String[] args) {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session ses=sf.openSession();
        Transaction tx=ses.beginTransaction();
        Customer c=new Customer();
        c.setCustId(1001);
        c.setCustName("mohan");
        c.getCustData().add("D1");
        c.getCustData().add("D2");
        c.getData().put(10, "M1");
        c.getData().put(11, "M2");
        ses.save(c);
        tx.commit();
        ses.close();
}
}
```

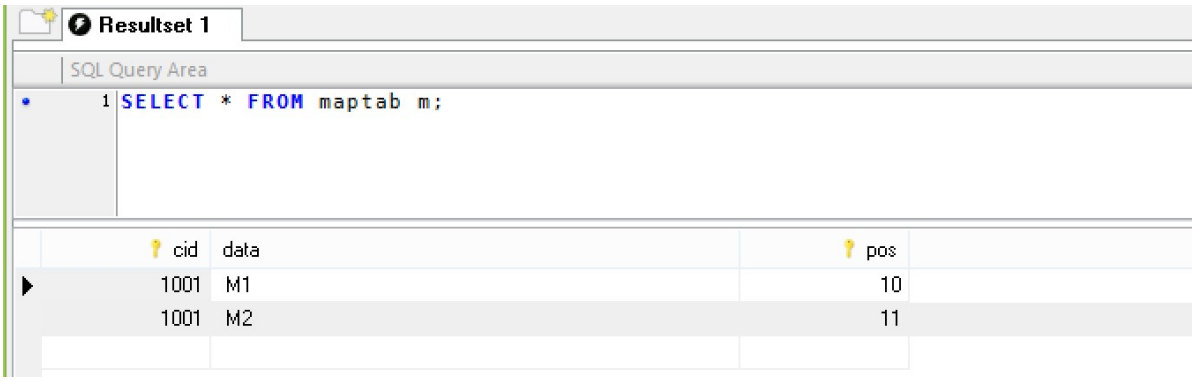
Output;-



Parent table



Child table



❖ MultiTable-multiClass concepts:-

1. Secondary Table
2. Component mapping
3. Inheritance
4. Association

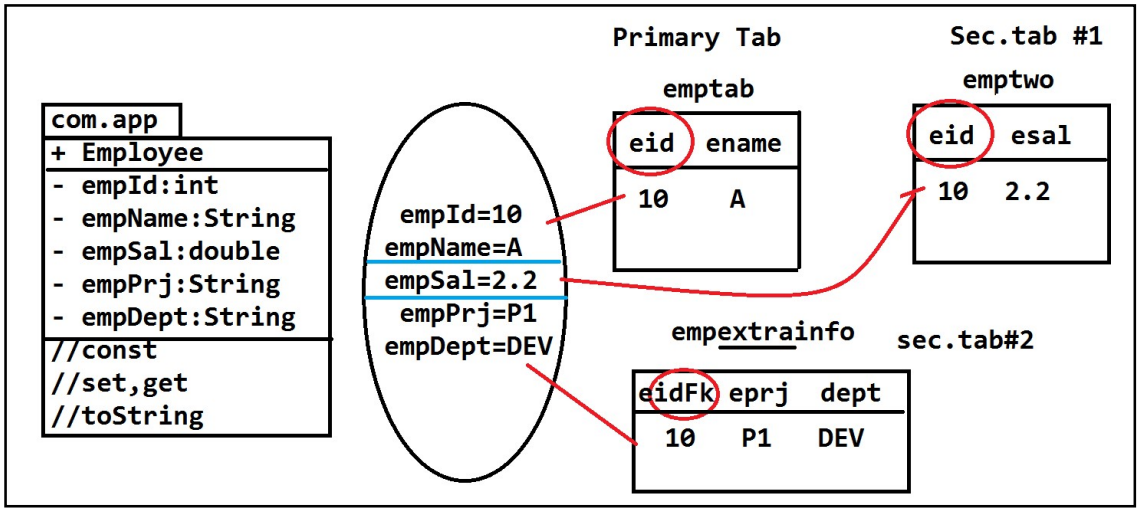
1. **SECONDARY TABLE:-** one class normally connected to one table (called as primary table). Some time we can converts and **store data in multiple tables of one class.** For this along with Primary table another table(s) created which are known as Secondary table(s).

Notes:-

1. Every Secondary table gets Primary key column automatically
2. We can create multiple Secondary Table for one class.
3. @Table indicates Primary Table.
   @SeconsaryTable indicates Sec. Tab.
4. For more than one sec. tab(s) format is
   @SeconadaryTables({
   @SecondaryTable(name=””),
   @SecondaryTable(name=””),…….
   })
5. At column level provide sec. table name using @Column(name=””,table=””)

- If table name is not provided for column, then it will be stored in PrimaryTab (that is @Table(name=””))

6. To change Primary key column name in Sec. table use code:
   pkJoinColumns=@Primarykey JoinColumn(name="colName")
   *** it is optional.



==========code==========

Model class

```
 package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.PrimaryKeyJoinColumn;
import  javax.persistence.SecondaryTable;
import javax.persistence.SecondaryTables;
import javax.persistence.Table;
@Entity
@Table(name="emptab")
@SecondaryTables({
        @SecondaryTable(name="emptwo"),
        @SecondaryTable(name="empextrainfo",pkJoinColumns=@PrimaryKeyJoinColumn(nam
e="eidfk"))
})
publicclass Employee {
        @Id
        @Column(name="eid")
        privateintempId;
        @Column(name="ename")
        private String empName;
        @Column(name="esal",table="emptwo")
        privatedoubleempSal;
        @Column(name="proj",table="empextrainfo")
```

```
	private String prjs;
	@Column(name="dept",table="empextrainfo")
	private String detp;
	public Employee() {
		super();
	}
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	public String getPrjs() {
		returnprjs;
	}
	publicvoid setPrjs(String prjs) {
		this.prjs = prjs;
	}
	public String getDetp() {
		returndetp;
	}
	publicvoid setDetp(String detp) {
		this.detp = detp;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", prjs=" + prjs + ", detp="
				+ detp + "]";
	}
```

```
}
```

Configuration File

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">create</property>
<mappingclass="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

Test class:-

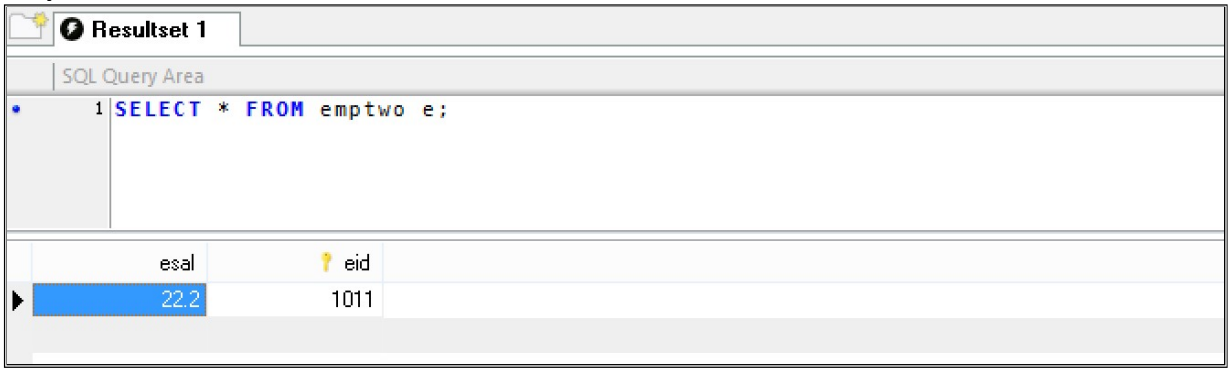
```
package com.app.model;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
public class Test {
public static void main(String[] args) {
        Configuration cfg=new Configuration();
        cfg.configure();
        SessionFactory sf=cfg.buildSessionFactory();
        Session ses=sf.openSession();
        Transaction tx=ses.beginTransaction();
        Employee emp=new Employee();
        emp.setEmpId(1011);
        emp.setEmpName("Mohan");
        emp.setEmpSal(22.2);
        emp.setPrjs("p1");
        emp.setDetp("computer");
        ses.save(emp);
        tx.commit();
        ses.close();
}
```

}

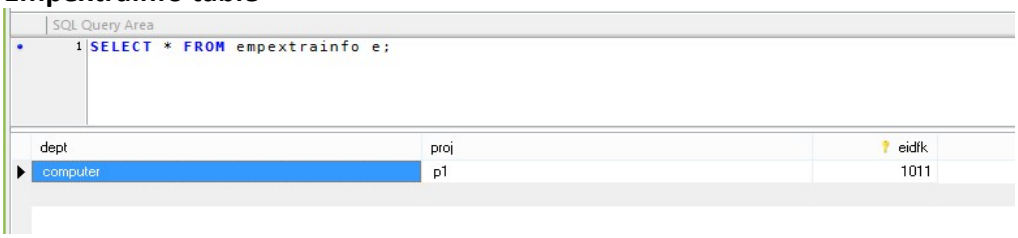
Output:-

Parent table

Emptwo table



Empextrainfo table



- **HAS-A Relation:-**

Using child type (class/interface) as a DataType in Parent (class) and creating variable is known as HAS-A (Associatation) Relation.

- Child can be class or interface
- But parent must be class only.
- Use child inside parent.
  Syntax:

Code:-
Class Address{}
Class employee
{
Address addr;//Has-A
}

- Coding order child first write then parent class

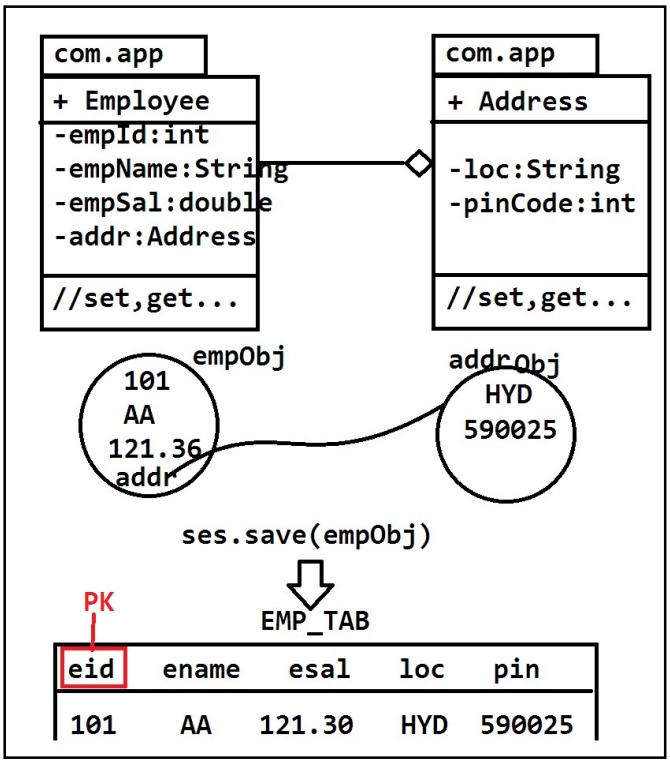
❖ **Component Mapping:-**

Two or more classes will be connected to one table. In this case Parent class makes HAS-A Relation with child as fully dependent.

That is if parent data not exist then child data has no meaning that should also be removed. It is called as fully dependent

Ex:- Employee with it's Address
Project with it's Modules
Student with it's Results

- **Consider below example (UML-ER)**



==========Coding :==========

Model class

```
package com.app.model;

import javax.persistence.Column;
import javax.persistence.Embeddable;

@Embeddable
publicclass Address {
	@Column(name="hno")
private String hno;
	@Column(name="loc")
private String loc;
	public Address() {
		super();
	}
	public String getHno() {
		returnhno;
	}
	publicvoid setHno(String hno) {
		this.hno = hno;
	}
	public String getLoc() {
		returnloc;
	}
	publicvoid setLoc(String loc) {
		this.loc = loc;
	}
	@Override
	public String toString() {
		return"Address [hno=" + hno + ", loc=" + loc + "]";
	}
}
////////////////////////////////////////////////////////////
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="emptabcom")
publicclass Employee {
	@Id
	@Column(name="eid")
	privateintempId;
```

```
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	privatedoubleempSal;
	@Embedded
	private Address addr;
	public Employee() {
		super();
	}
	publicint getEmpId() {
		returnempId;
	}
	publicvoid setEmpId(intempId) {
		this.empId = empId;
	}
	public String getEmpName() {
		returnempName;
	}
	publicvoid setEmpName(String empName) {
		this.empName = empName;
	}
	publicdouble getEmpSal() {
		returnempSal;
	}
	publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
	}
	public Address getAddr() {
		returnaddr;
	}
	publicvoid setAddr(Address addr) {
		this.addr = addr;
	}
	@Override
	public String toString() {
		return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", addr=" + addr + "]";
	}
}
```

Configuration file

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/test</pro
perty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
<mappingclass="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

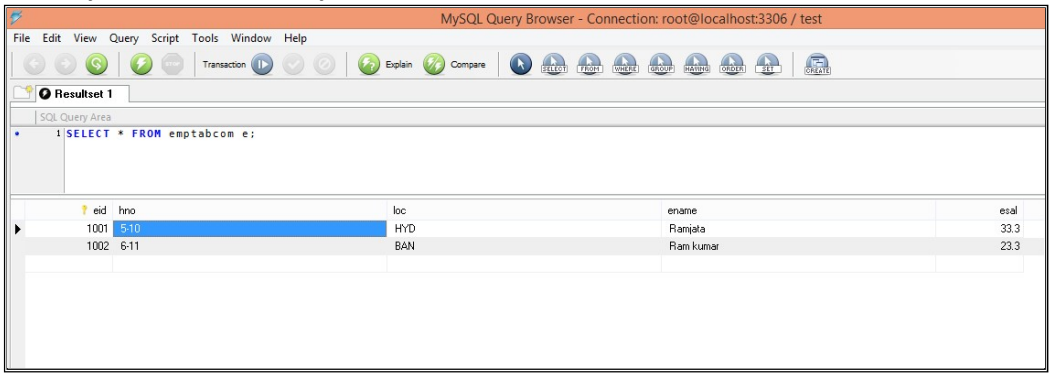
Test class

```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import  com.app.model.Address;
import com.app.model.Employee;
public class Test {
        public static void main(String[] args) {
                Configuration cfg=new Configuration();
                cfg.configure();
                SessionFactory sf=cfg.buildSessionFactory();
                Session ses=sf.openSession();
                Transaction tx=ses.beginTransaction();
                Address addr=new Address();
                addr.setHno("6-11");
                addr.setLoc("BAN");
                Employee emp=new Employee();
                emp.setEmpId(1002);
                emp.setEmpName("Ram kumar");
                emp.setEmpSal(23.3);
                emp.setAddr(addr);
                ses.save(emp);
                tx.commit();
                ses.close();
                }
}
```

Output:-

```
Test (10) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 5, 2018, 11:07:49 AM)
Oct 05, 2018 11:07:50 AM org.hibernate.engine.jdbc.connections.internal.DriverManag
INFO: HHH10001003: Autocommit mode: false
Oct 05, 2018 11:07:50 AM org.hibernate.engine.jdbc.connections.internal.PooledConne
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
Oct 05, 2018 11:07:50 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
Oct 05, 2018 11:07:50 AM org.hibernate.resource.transaction.backend.jdbc.internal.I
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.eng
Hibernate:
    insert
    into
        emptabcom
        (hno, loc, ename, esal, eid)
    values
        (?, ?, ?, ?, ?)
```

Table (Database Record)



@Embeddable:- This annotation must be applied on child class, to indicate it is eligible to connect with other (parent) class. That is Selecte for linking

@Embedded:- This annotation must be applied on HAS-A relation level.

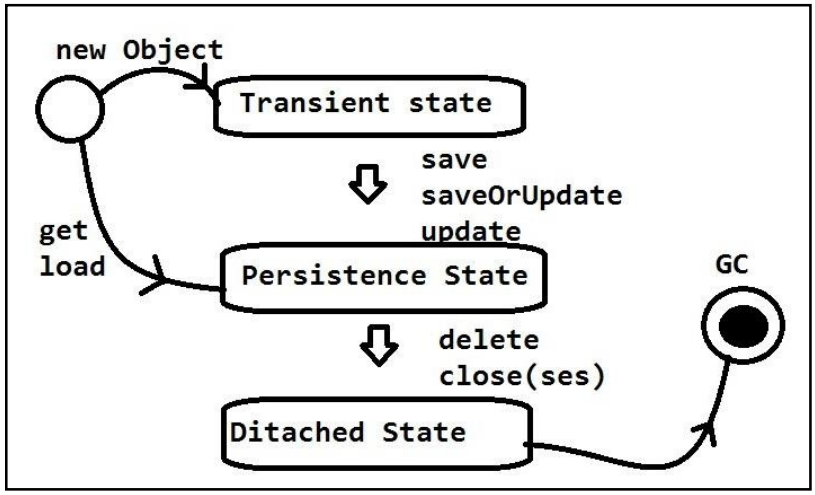
It indicates linked with parent.

@AttributeOverride:- It is optional (less used in Real-time). It is used to modify column name, without effecting child class code (Column name).

HIBERNATE OBJECT STATE:-

Every model class object must be having identifications steps based on operation perform by programmer, possible state are three

1. Transient state
2. persistence state
3. Detached state



Transient state

A model class object exist in heap area which not connected in data base (which not in Cache), a newly created object comes under transient state

Ex:-Employee emp=new employee();

Persistence State:-

A model class object which existedin Cache memory (**Session Cache/Session Factory Cache**) is known as Persistence State. If we perform session operation like. **Save(),update(),saveOrUpdate()** then object move to Persistence state from transient state. If we perform get() or load() operations object will be created in Cache that is directly Persistence State.

Detached state:-

An object (model class object) removed from Cache(Session Cache /Session Factory Cache)is known as Detached state. This object is eligible for garvage collection by performing operation like.

Ses.close() or ses.delete(obj);

HibernateUtil class:-

This class is used to create one object to SessionFactory(I) using singleton design pattern with eager loading (Static block).

** Sessionfactory is a heavy weight (consumes more memory) object.

Sf, loads driver class, creates connection, handles statement and also hibernate properties management (dialect, show_sql……)

==========code for HibernateUtic class==========

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
publicclass HibernateUtil {
        privatestatic SessionFactory sf=null;
        static
        {
                sf=new Configuration().configure().buildSessionFactory();
```

```
}
        publicstatic SessionFactory getSf() {
                returnsf;
        }
}
```

** Here static block will be executed only one time at the time of class loading.

- in test class use try-with-resource concept (jdk1.7) to use concept auto close() of Session object done bby JVM(JRE).
- A class or interface which extends/implements java.lang.AutoCloseaaable(I) having close() method can be used in truy-with-resource, not all.

==========Test class code==========

```
package com.app.test;
//ctrl+shift+o(imports)
public class Test
{
public static void  main(String arg[])
{
//jkd1.7 && Hibernate 5.x
Transaction tx=null;
try(Session ses=HibernateUtil.getSf().openSession() ) {
tx=ses.beginTransaction();
Employee e=new Employee();
e.setEmpId(10);
e.setEmpName("AA");
e.setEmpSal(33.3);
ses.save(e);
tx.commit();
}catch(Exception e)
{
tx.rollback();
e.printStackTrace();
}
}
}
```

- **Here Session(I) extends java.lang.AutoCloseable(I).** So, in test class we are not required to call

ses.close() method externally

****** To known impl class details of any class/interface reference variables used code

Obj.getClass().getName();

Ex:-

Session ses=……

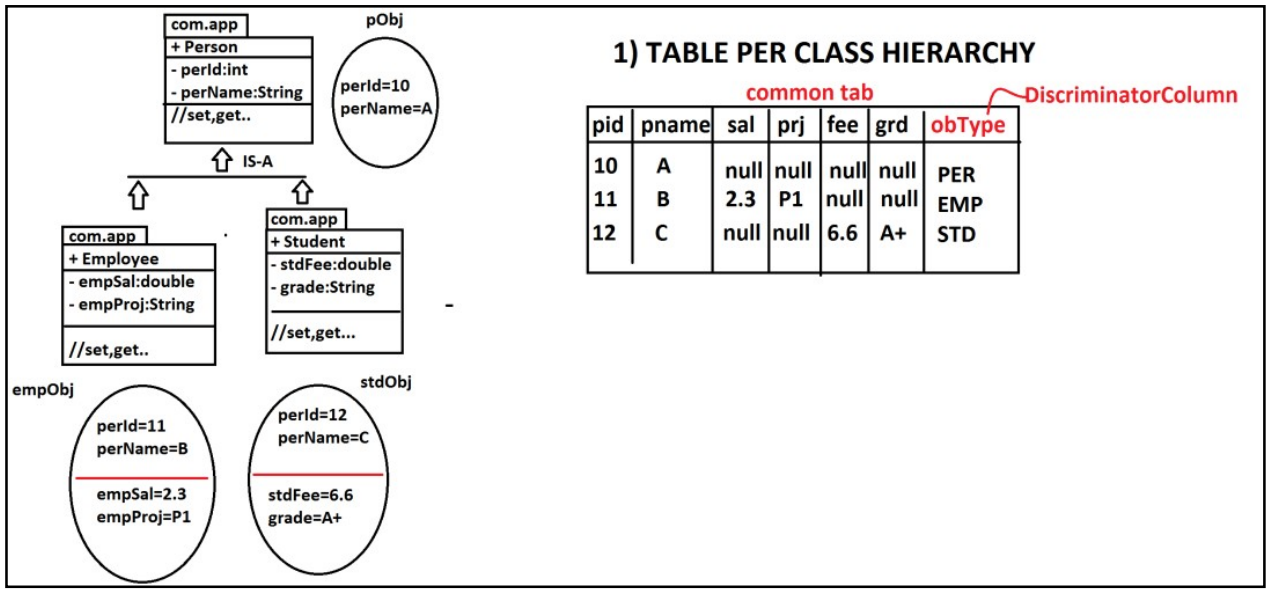
```
sysout(ses.getClass().getName());
Transaction tx=ses.beginTransaction();
sysout(tx.getClass().getName());
```

Inheritance Mapping [IS – A Relation]

1. **Table Per Class Hierarchy**
2. **Table Per Sub Class**
3. **Table Per Concrete Class**

1. **TABLE PER CLASS HIERARCHY:**
   This design provides **single table for allmodel classes**. It will considers all classes variables as column names and takes **one extra column** "**Discriminator**" which provides information like "Row related to which model class".

- Discriminator column can be int type or String/char type

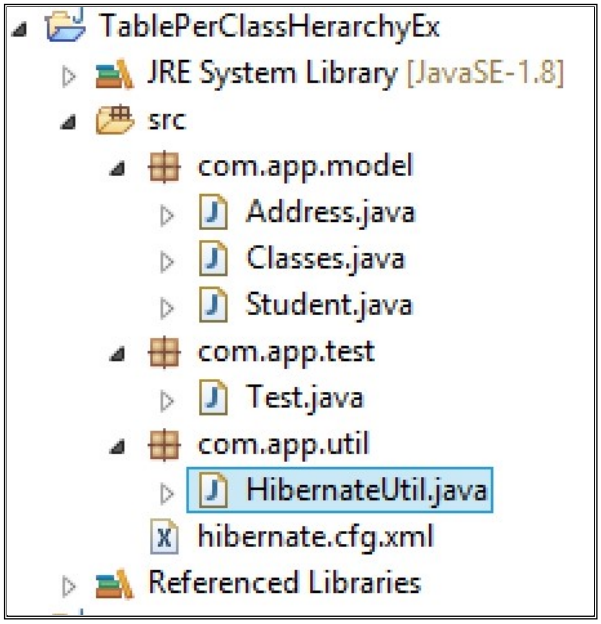


For IS-A Relation Mappoing enum and Annotation are given as:
enum: InheritanceType
Annotation: @Inheritance
For TABLE PER CLASS HIERATCHY DESIGN CODE:
@Inheritance(strategy= InheritanceType.SINGLE_TYPE)
For single table design, we should also provide extra column 'name,type and value' using code:
Annota.: **@DiscriminatorColumn**
Enum : **DiscriminatorTupe**
Code look like:
@DiscriminatorColumn(name="obType",discriminatorType= DiscriminatorType.STRING)
For object value code is:
@DiscriminatorValue(" --------- ")

==========Example==========code==========

TABLE PER CLASS HERARCHY :-

1. **Folder Structure**



2. **Model classes**

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.DiscriminatorColumn;
import javax.persistence.DiscriminatorType;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;
@Entity
@Table(name="commonTab")
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="obType",discriminatorType=DiscriminatorType.STRING)
@DiscriminatorValue(value="STD")
public class Student {
	@Id
	@Column(name="sid")
	private int stdId;
	@Column(name="sname")
	private String stdName;
```

```
public Student() {
        super();
}
public int getStdId() {
```

```
		return stdId;
	}
	public void setStdId(int stdId) {
		this.stdId = stdId;
	}
	public String getStdName() {
		return stdName;
	}
	public void setStdName(String stdName) {
		this.stdName = stdName;
	}
	@Override
	public String toString() {
		return "Student [stdId=" + stdId + ", stdName=" + stdName + "]";
	}
}
//////////////////////////////////////////
package com.app.model;
import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
@Entity
@DiscriminatorValue("ADD")
public class Address extends Student{
	@Column(name="vill")
	private String village;
	@Column(name="hno")
	private int Hoding_no;
	@Column(name="city")
	private String city;
	public Address() {
		super();
	}
	public String getVillage() {
		return village;
	}
	public void setVillage(String village) {
		this.village = village;
	}
	public int getHoding_no() {
		return Hoding_no;
	}
	public void setHoding_no(int hoding_no) {
		Hoding_no = hoding_no;
```

```
	}
	public String getCity() {
		return city;
	}
	public void setCity(String city) {
		this.city = city;
	}
	@Override
	public String toString() {
		return "Address [village=" + village + ", Hoding_no=" + Hoding_no + ", city=" + city
+ "]";
	}
}
package com.app.model;
import javax.persistence.Column;
import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
@Entity
@DiscriminatorValue("CLS")
public class Classes extends Student {
	@Column(name="cname")
	private String Class_Name;
	public Classes() {
		super();
	}
	public String getClass_Name() {
		return Class_Name;
	}
	public void setClass_Name(String class_Name) {
		Class_Name = class_Name;
	}
	@Override
	public String toString() {
		return "Classes [Class_Name=" + Class_Name + "]";
	}
}
```

3. Configuration file(hibernate.cfg.xml)

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

```
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect"> org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.model.Student"/>
<mapping class="com.app.model.Address"/>
<mapping class="com.app.model.Classes"/>
</session-factory>
</hibernate-configuration>
```

3. HibernateUtil file:-

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
private static SessionFactory sf=null;
static
{
        sf=new Configuration().configure().buildSessionFactory();
}
public static SessionFactory getSf()
{
        return sf;
}
}
```

4. Test class:-

```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Address;
import com.app.model.Classes;
import com.app.model.Student;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        try(Session ses=HibernateUtil.getSf().openSession()){
                Transaction tx1=ses.beginTransaction();
                Student st=new Student();
                st.setStdId(1001);
```

```
                st.setStdName("Mohan");
                Address add=new Address();
                add.setStdId(101);
                add.setStdName("Sohan");
                add.setVillage("GH");
                add.setHoding_no(011);
                add.setCity("hyd");
                Classes cs=new Classes();
                cs.setStdId(102);
                cs.setStdName("Anil");
                cs.setClass_Name("MCA");
                ses.save(st);
                ses.save(add);
                ses.save(cs);
                tx1.commit();
                ses.close();
        }catch(Exception ex)
        {
                ex.printStackTrace();

        }
}
}
```

** for DiscriminatorColumn default column name is =”Dtype” and default datatype is =”String”. Prossible datatype are String, char and Int.

Output:-

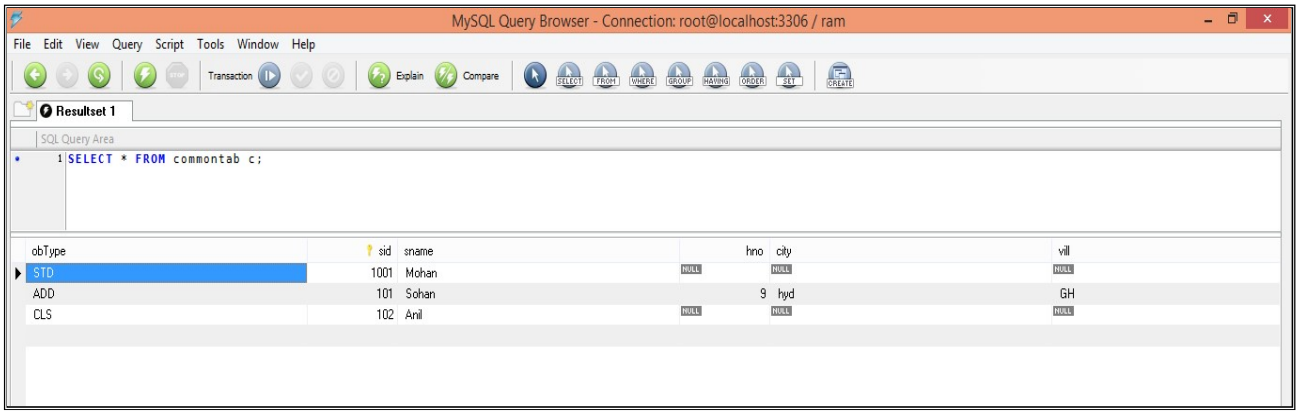
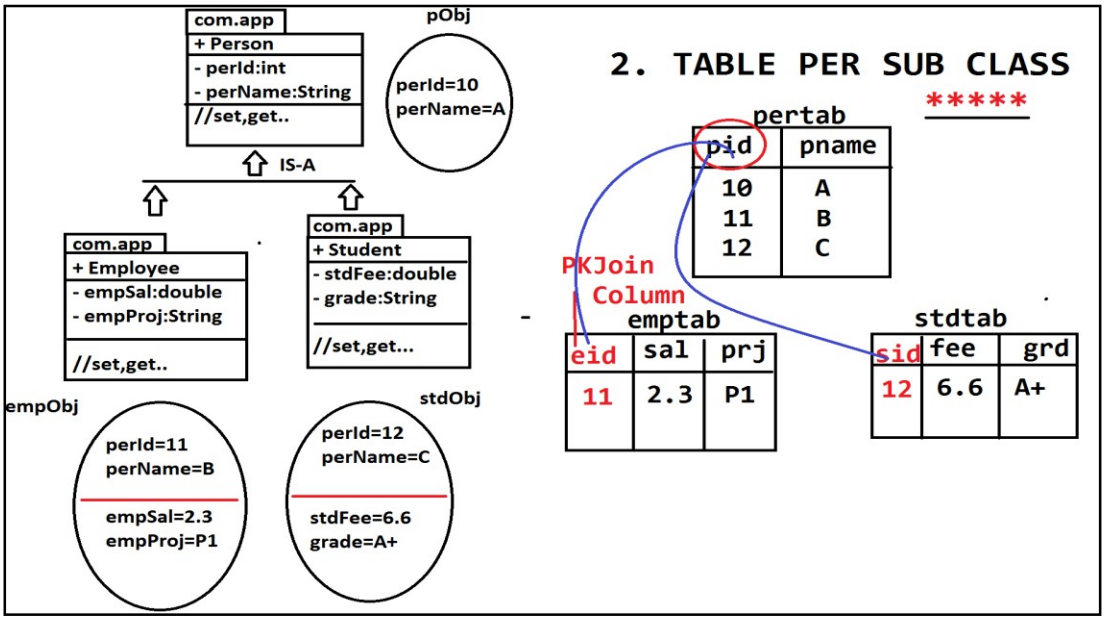


TABLE PER SUB CLASS:-

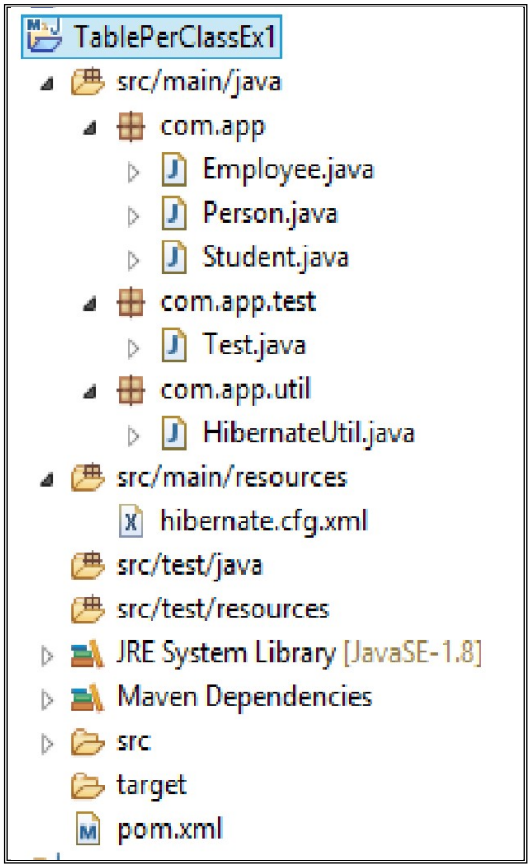
In this case hibernate creates table for every child case along with parent class.

- On saving data, parent data stored at parent table and child data stored at child table.
- **Parent** table and child table is connected using PK-FK column. FK column also called as key column.



==========code==========

Folder Structure



Model class

```
package com.app;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;
@Entity
@Table(name="pertab")
@Inheritance(strategy=InheritanceType.JOINED)
publicclass Person{
@Id
@Column(name="pid")
privateintperId;
@Column(name="pname")
private String perName;
public Person() {
        super();
}
publicint getPerId() {
```

```
		returnperId;
}
publicvoid setPerId(intperId) {
		this.perId = perId;
}
public String getPerName() {
		returnperName;
}
publicvoid setPerName(String perName) {
		this.perName = perName;
}
@Override
public String toString() {
		return"Person [perId=" + perId + ", perName=" + perName + "]";
}

}
////////////////////////////////////////////////////////////////////////////////////////////////
package com.app;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
@Entity
@Table(name="empTab")
@PrimaryKeyJoinColumn(name="eid")
publicclass Employee extends Person {
@Column(name="sal")
privatedoubleempSal;
public Employee() {
		super();
}
publicdouble getEmpSal() {
		returnempSal;
}
publicvoid setEmpSal(doubleempSal) {
		this.empSal = empSal;
}
@Override
public String toString() {
		return"Employee [empSal=" + empSal + "]";
}
}
////////////////////////////////////////////////////////////////////////////////////////////////
```

```
package com.app;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.PrimaryKeyJoinColumn;
import javax.persistence.Table;
@Entity
@Table(name="stdTab")
@PrimaryKeyJoinColumn(name="sid")
publicclass Student extends Person {
@Column(name="fee")
privatedoublestdFee;
@Column(name="grd")
private  String grade;
public Student() {
        super();
}
publicdouble getStdFee() {
        returnstdFee;
}
publicvoid setStdFee(doublestdFee) {
        this.stdFee = stdFee;
}
public String getGrade() {
        returngrade;
}
publicvoid setGrade(String grade) {
        this.grade = grade;
}
@Override
public String toString() {
        return"Student [stdFee=" + stdFee + ", grade=" + grade + "]";
}
}
```

- **Here PKJoinColumn means "a column which behaves as PK column of child table and FK column to link with parent table".**

HibernateUtil .java

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
publicclass HibernateUtil  {
privatestatic SessionFactory sf=null;
static
```

```
{
        sf=new Configuration().configure().buildSessionFactory();
}
publicstatic SessionFactory getSf()
{
        returnsf;
}
}
```

Hibernate.cfg.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</pr
operty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">create</property>
            <mappingclass="com.app.Person"/>
            <mappingclass="com.app.Employee"/>
            <mappingclass="com.app.Student"/>
</session-factory>
</hibernate-configuration>
```

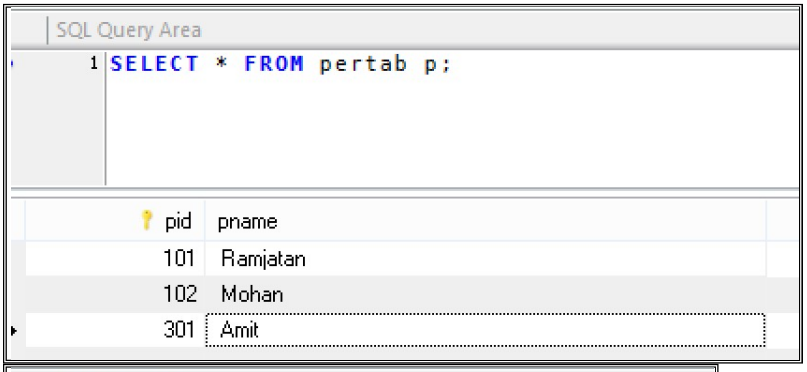
Test class

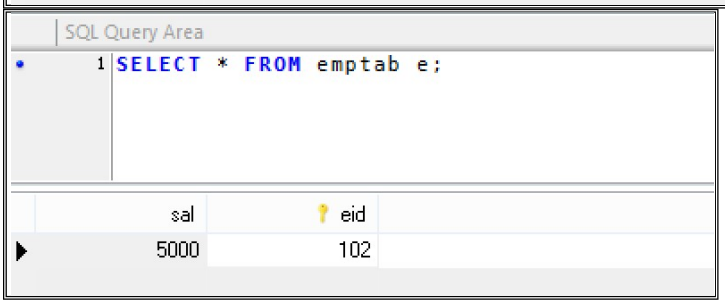
```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.Employee;
import com.app.Person;
import com.app.Student;
import com.app.util.HibernateUtil;
public class Test {
        public static void main(String[] args) {
                try(Session ses=HibernateUtil.getSf().openSession()){
                        Transaction tx=ses.beginTransaction();
                Person p=new Person();
                p.setPerId(101);
                p.setPerName("Ramjatan");
                Employee emp=new Employee();
                emp.setPerId(102);
```

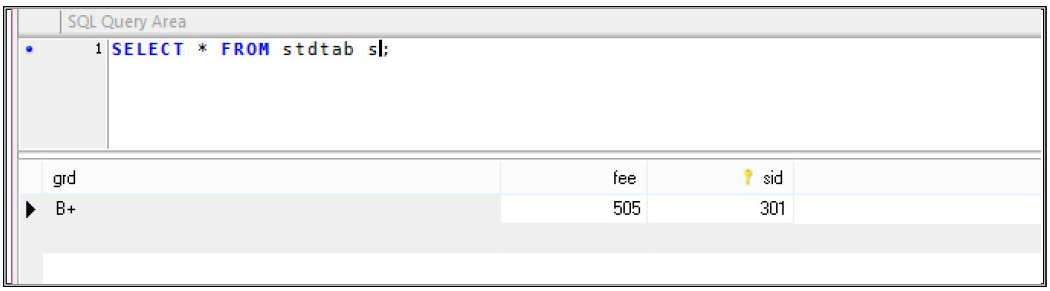
```
                emp.setPerName("Mohan");
                emp.setEmpSal(5000);
                Student st=new Student();
                st.setPerId(301);
                st.setPerName("Amit");
                st.setStdFee(505);
                st.setGrade("B+");
                ses.save(p);
                ses.save(emp);
                ses.save(st);
                        tx.commit();
                        ses.close();
                }catch(Exception ex)
                {
                        ex.printStackTrace();
                }
        }
}
```

Output:-

Data Base Record

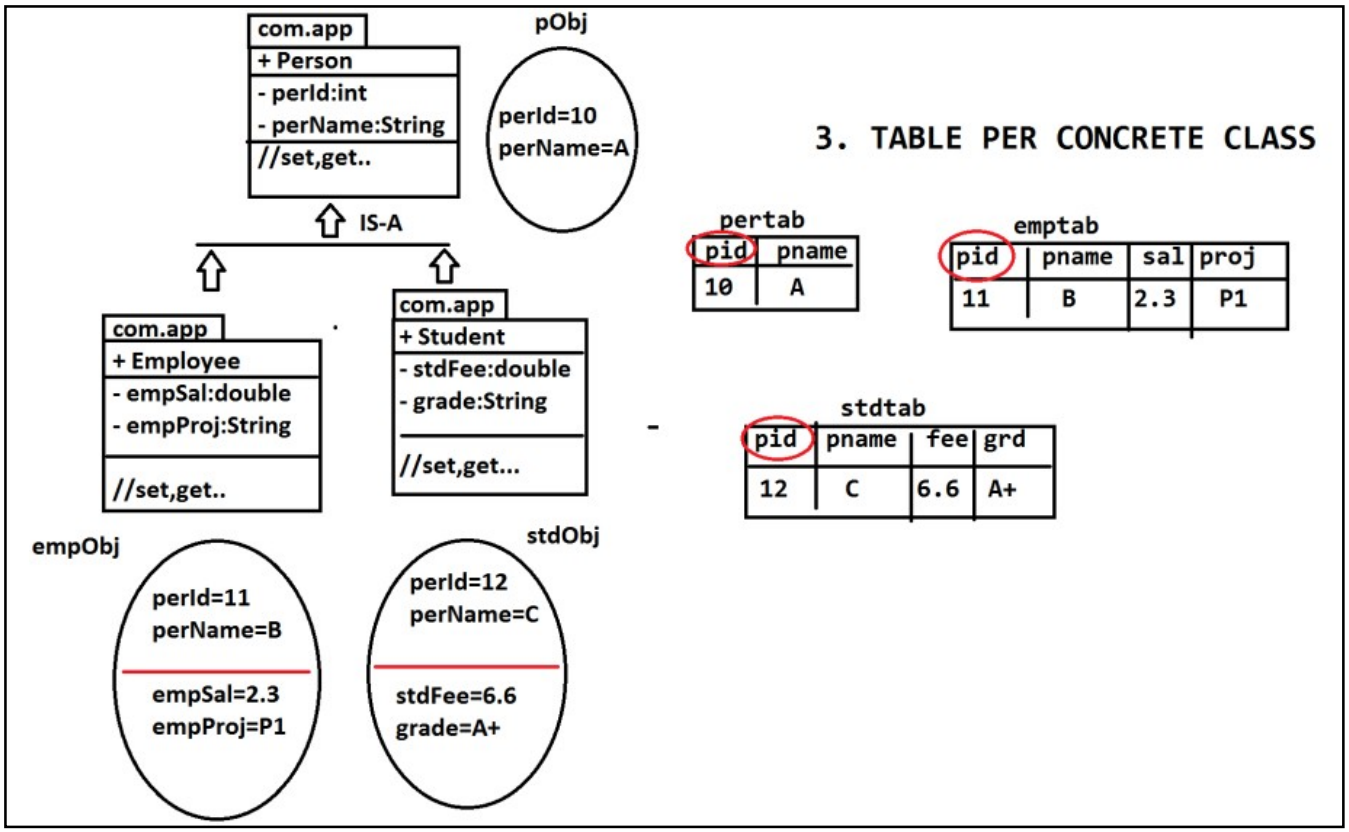




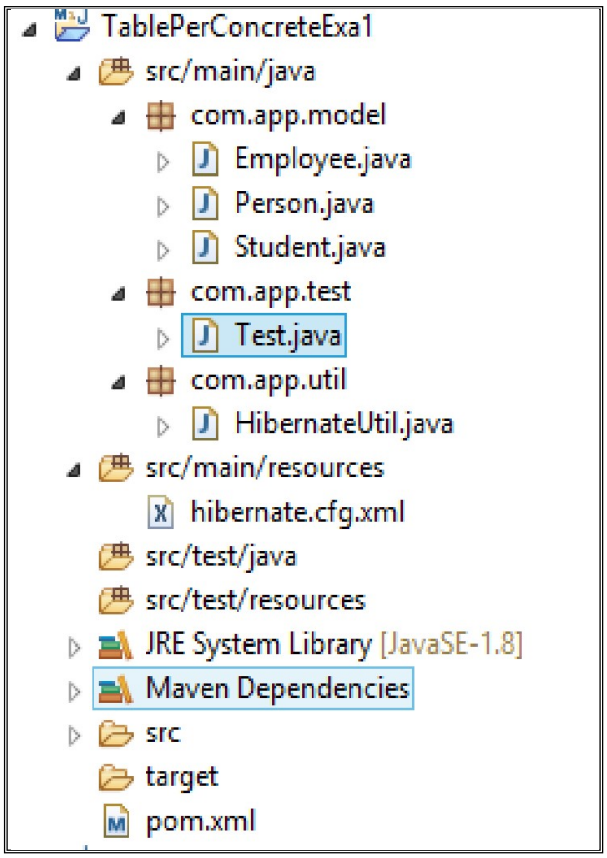


❖ **TABLE PER CONCRETE CLASS**

This design creates independent tables for every class in IS-A relation including parent class variable.



==========CODE==========



Model class:-

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Inheritance;
import javax.persistence.InheritanceType;
import javax.persistence.Table;
@Entity
@Table(name="perTab1")
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
publicclass Person {
@Id
@Column(name="pid")
privateintperId;
@Column(name="pname")
private String perName;
public Person() {
        super();
}
publicint getPerId() {
```

```
        returnperId;
}
publicvoid setPerId(intperId) {
        this.perId = perId;
}
public String getPerName() {
        returnperName;
}
publicvoid setPerName(String perName) {
        this.perName = perName;
}
@Override
public String toString() {
        return"Person [perId=" + perId + ", perName=" + perName + "]";
}
}
////////////////////////////////////////////////////////////////////////////////////////////////
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;
@Entity
@Table(name="emptab1")
publicclass Employee extends Person {
@Column(name="sal")
privatedoubleempSal;
@Column(name="prj")
private  String empProj;
public Employee() {
        super();
}
publicdouble getEmpSal() {
        returnempSal;
}
publicvoid setEmpSal(doubleempSal) {
        this.empSal = empSal;
}
public String getEmpProj() {
        returnempProj;
}
publicvoid setEmpProj(String empProj) {
        this.empProj = empProj;
}
@Override
```

```
public String toString() {
        return"Employee [empSal=" + empSal + ", empProj=" + empProj + "]";
}
}
    ////////////////////////////////////////////////////////////////////////////////////////////
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Table;
@Entity
@Table(name="stdTab1")
publicclass Student extends Person {
@Column(name="fee")
privatedoublestdFee;
@Column(name="grd")
private  String grade;
public Student() {
        super();
}
publicdouble getStdFee() {
        returnstdFee;
}
publicvoid setStdFee(doublestdFee) {
        this.stdFee = stdFee;
}
public String getGrade() {
        returngrade;
}
publicvoid setGrade(String grade) {
        this.grade = grade;
}
@Override
public String toString() {
        return"Student [stdFee=" + stdFee + ", grade=" + grade + "]";
}


}
```

HibernateUtil class

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
publicclass HibernateUtil  {
privatestatic SessionFactory sf=null;
```

```
static
{
        sf=new Configuration().configure().buildSessionFactory();
}
publicstatic SessionFactory getSf()
{
        returnsf;
}
}
```

Hibernate.cfg.xml file

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</property>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect"> org.hibernate.dialect.MySQL5Dialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">create</property>
                <mappingclass="com.app.model.Person"/>
                <mappingclass="com.app.model.Employee"/>
                <mappingclass="com.app.model.Student"/>
</session-factory>
</hibernate-configuration>
```

Test class

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.model.Person;
import com.app.model.Student;
import com.app.util.HibernateUtil;
publicclass Test {
publicstaticvoid main(String[] args) {
        try(Session ses=HibernateUtil.getSf().openSession()){
                Transaction  tx=ses.beginTransaction();
                Person p=new Person();
                p.setPerId(101);
```

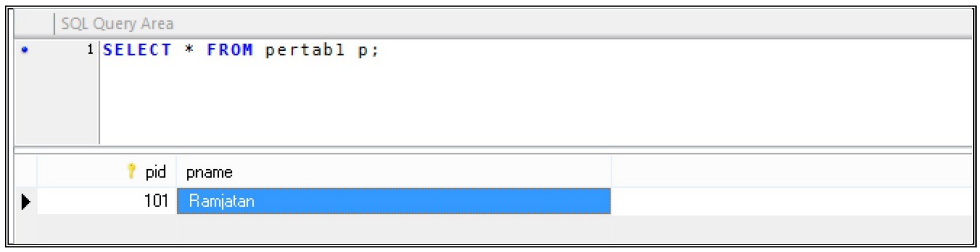
```
                p.setPerName("Ramjatan");
                Employee emp=new Employee();
                emp.setPerId(201);
                emp.setPerName("Mohan");
                emp.setEmpSal(5000);
                emp.setEmpProj("P1");
                Student st=new Student();
                st.setPerId(301);
                st.setPerName("Sudhir kumar");
                st.setStdFee(6000);
                st.setGrade("A+");
                ses.save(p);
                ses.save(emp);
                ses.save(st);
                tx.commit();
                //ses.close();
        }
}
}
```

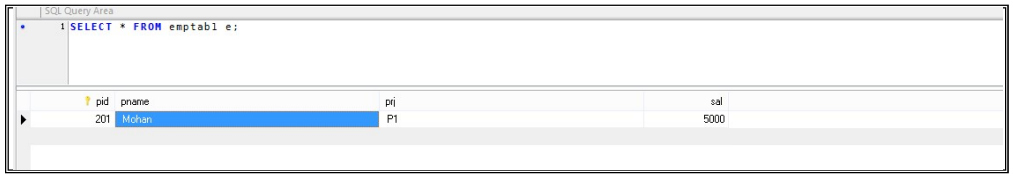
** **Table per concrete class** compared with all other designs, it is very faster in operations (insert,update,delete) etc….. used in regular operations lik:**insert/fetch/modify.**

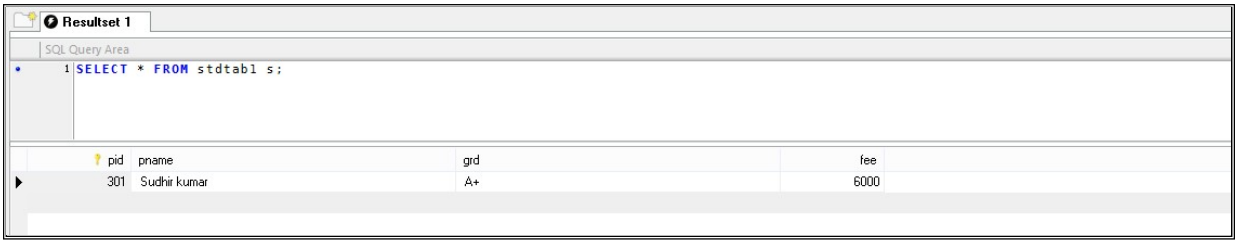
** **Table per sub class** is used in report generation tools concept. Like:-**iText,Jasper,POI etc..**

Output:-

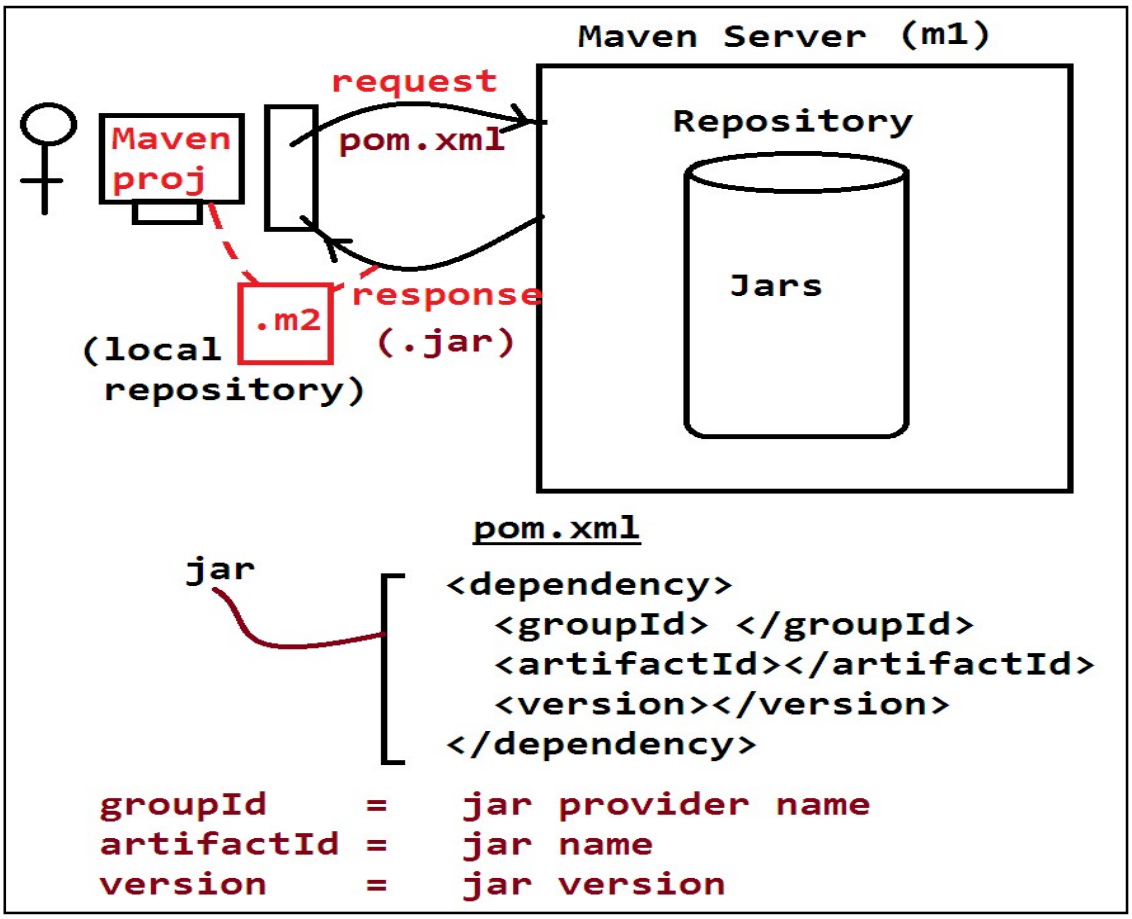
Database Table Record







Maven Working Flow



If multiple model classes are connected using IS-A Relation in Application (extends keyword) then Hibernate can creates tables based on design selected by Programmers.

Types of Maven Project :-

Simple Maven Project (stand alone)

1. **ArchType maven Project(Web/Webservieces/enterprise apps)**
   Complie
   .java->.class
   Build
   .class-> .jar/.war/.ear

Deploy:

Place .war.ear in server and start

***) Server terms:

UAT= user Acdeptance test server Production=Realtime server

(Used by end users)

QA/DEV serves=Testing/ Development Team servers

- **HIBERNATE MAVEN APPLICATION :-**

Steps:- 1.

Create maven Projcet

>file > new > other > Search with Maven > choose "Maven Project" next > choose checkbox [v] create simple Project (skip arch….)"

> next > Enter detail like:

Group id : **org.nareshittech**

artifactId : **HibernateAppMaven**

version : **1.0**

- Click on Finish

Steps: 2.

Open pom.xml in XMl mode and add dependenies and build-pulgins(copy from document)

Steps: 3.

Update Maven Project *******

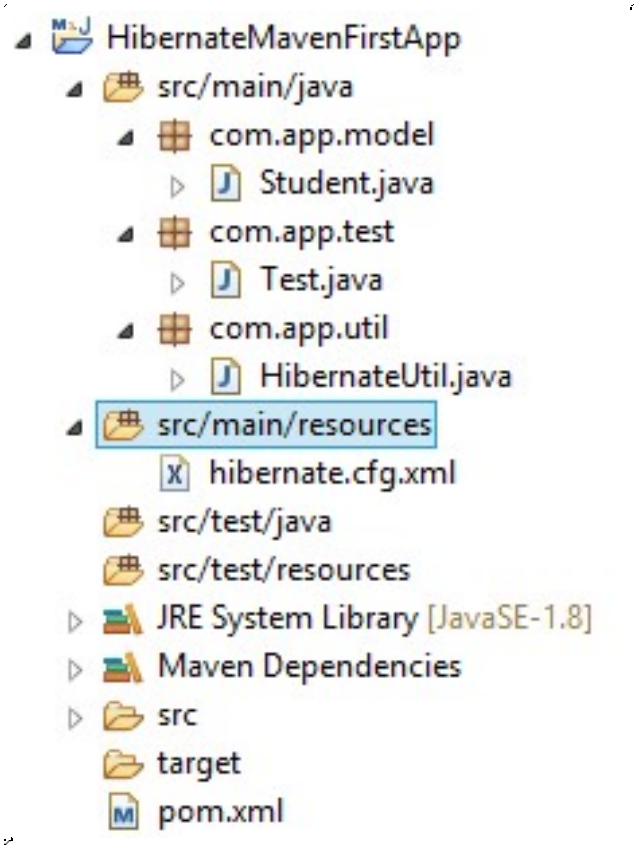
- Right click on Project > Maven > update Project > next Finish

Steps : 4.

Define hibernate files (model,util,test,cfg,file);

****** Steps for Clean Setup:**

- Right click on Project > Run As > Maven install Then next
- Right click on project > maven > update project



Example 1
Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>HibernateMavenFirstApp</artifactId>
<version>1.0</version>
<dependencies>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.1.11.Final</version>
</dependency>
<dependency>
```

```
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.6</version>
</dependency>
</dependencies>
<build>
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.1</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

//////////////////////Model class//////////////////////////////

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="stdTab2")
publicclass Student {
	@Id
	@Column(name="sid")
	privateintstdId;
	@Column(name="sname")
	private String stdName;
	@Column(name="sfee")
	privatedoublestdFee;
	public Student() {
		super();
	}
	publicint getStdId() {
		returnstdId;
	}
	publicvoid setStdId(intstdId) {
```

```
		this.stdId = stdId;
	}
	public String getStdName() {
		returnstdName;
	}
	publicvoid setStdName(String stdName) {
		this.stdName = stdName;
	}
	publicdouble getStdFee() {
		returnstdFee;
	}
	publicvoid setStdFee(doublestdFee) {
		this.stdFee = stdFee;
	}
	@Override
	public String toString() {
		return"Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" + stdFee
+ "]";
	}
	}
```

/////////////////HibernateUtil file//////////////////////

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
publicclass HibernateUtil {
	privatestatic SessionFactory sf=null;
	static
	{
	sf=new Configuration().configure().buildSessionFactory();
	}
	publicstatic SessionFactory getSf()
	{
		returnsf;
	}
}
```

/////////////////////Hibernate.cfg.xml//////////////////////////////

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
```

```
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</pr
operty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
      <propertyname="hibernate.dialect">
      org.hibernate.dialect.MySQL5Dialect</property>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">update</property>
            <mappingclass="com.app.model.Student"/>

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

//////////////////////Test class:-////////////////////////////////

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Student;
import com.app.util.HibernateUtil;
publicclass Test {
      publicstaticvoid main(String[] args) {
            try(Session ses=HibernateUtil.getSf().openSession()){
                  Transaction tx1=ses.beginTransaction();
                  Student st=new Student();
                  st.setStdId(1002);
                  st.setStdName("Sumanth");
                  st.setStdFee(5000);
                  ses.save(st);
                  tx1.commit();
            }catch(Exception ex)
            {
                  ex.printStackTrace();
            }
      }
}
```

Output:-

```
Test (13) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 6, 2018, 10:56:10 AM)
Oct 06, 2018 10:56:11 AM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
Oct 06, 2018 10:56:12 AM org.hibernate.resource.transaction.backend.
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [or
Hibernate:
    insert
    into
        stdTab2
        (sfee, sname, sid)
    values
        (?, ?, ?)
```

SQL Query Area

```
1 SELECT * FROM stdtab2 s;
```

sid	sfee	sname
1001	222.2	Mohan
1002	5000	Sumanth

❖ Multiplicity

Mulitiplicity:- Rows in one table connected to rows in another table using Tables PK-FK Columns.

- One table PK column should be taken as another table FK column.
- These are 4 types in Database.
- There are
  a). one-to-one (1…1)
  b). one-to-many(1…*)
  c). many-to-one(*…1)
  d). many-to-many(*…*)

* In simple 1…* means " 1 row in one table is conneted to many(*) rows in another table"

* it speaks relations between rows. By creating link bettewen tables.

* Hint: many(*) side extra column(FK) is created. For *…* extra table with 2 FK column will be created.

** for 1…1 use *…1 and apply unique condition at many side.

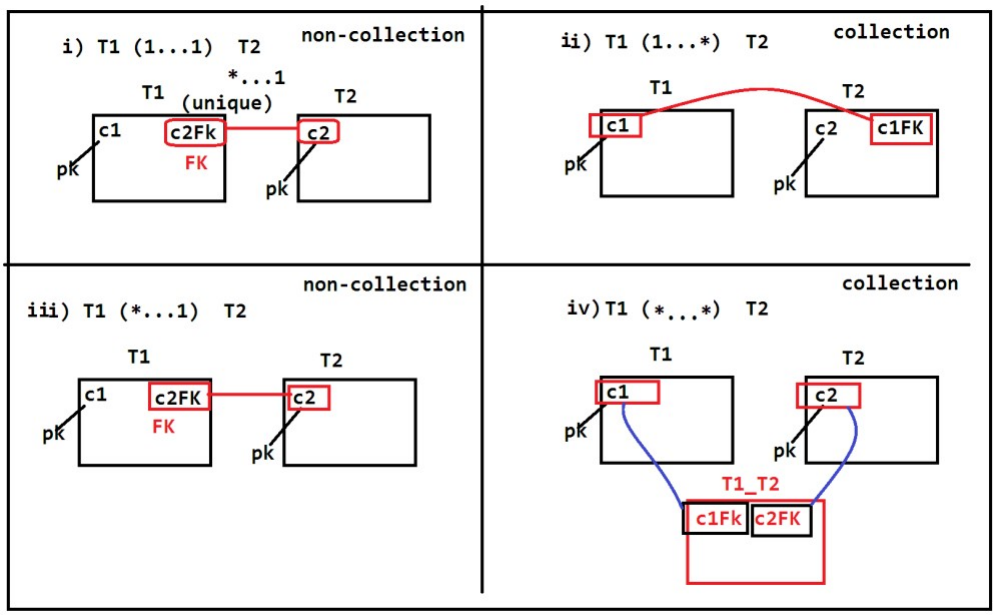
ASSOCIATION MAPPING (HAS-A):-

Hibernate supports multiplicity using HAS-A relation. It has made them into 2 types.

a. Non-Collection based
b. Collection based.

Non-collection based
1…1
*…1

Collection based
1…*
…



Steps to implement HAS-A concept:-

1. Wirte tow classes code any apply HAS-A relation between those
2. Write multiplicity Annotation over HAS-A Relation level
   1…* => @OneToMany
   … => @ManyToMany
   *…1 => @ManyToOne
   1…1 => @ManyToOne(Unique)
3. Convert HAS- to collection mode if multiplicity is collection type
4. Draw tables with PK-FK columns

Exm#1

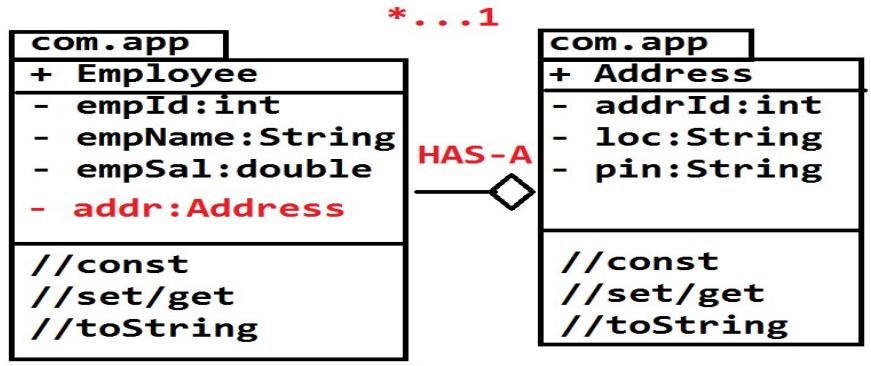
Employee —— **HAS-A** —— **Address**

*…1

```
Class Employee
{
 @ManyToOne
 Address addr;
}
Class Project {
@OneToMany
Set<Module> mobs;
}
```

```
classAddress
{
 }
class Module {
}
```

Example for :Many-to-One(*…1)



Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>HibernateMavenFirstApp</artifactId>
<version>1.0</version>
<dependencies>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.1.11.Final</version>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.6</version>
</dependency>
</dependencies>
<build>
<plugins>
```

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.1</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

Model class:-

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="addrtab")
publicclass Address {
@Id
@Column(name="aid")
privateintaddrId;
@Column(name="loc")
private String loc;
@Column(name="pin")
private String pin;
public Address() {
        super();
}
publicint getAddrId() {
        returnaddrId;
}
publicvoid setAddrId(intaddrId) {
        this.addrId = addrId;
}
public String getLoc() {
        returnloc;
}
publicvoid setLoc(String loc) {
        this.loc = loc;
```

```
}
public String getPin() {
        returnpin;
}
publicvoid setPin(String pin) {
        this.pin = pin;
}
@Override
public String toString() {
        return"Address [addrId=" + addrId + ", loc=" + loc + ", pin=" + pin + "]";
}
}
//////////////////////////////////////////////////////////////////////
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
@Entity
@Table(name="empTab")
publicclass Employee {
@Id
@Column(name="eid")
privateintempId;
@Column(name="ename")
private String empName;
@Column(name="esal")
privatedoubleempSal;
@ManyToOne
@JoinColumn(name="aidFk")//fk column
private Address addr;
public Employee() {
        super();
}
publicint getEmpId() {
        returnempId;
}
publicvoid setEmpId(intempId) {
        this.empId = empId;
}
public String getEmpName() {
        returnempName;
```

```
}
publicvoid setEmpName(String empName) {
	this.empName = empName;
}
publicdouble getEmpSal() {
	returnempSal;
}
publicvoid setEmpSal(doubleempSal) {
	this.empSal = empSal;
}
public Address getAddr() {
	returnaddr;
}
publicvoid setAddr(Address addr) {
	this.addr = addr;
}
@Override
public String toString() {
	return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", addr=" + addr + "]";
}
}
```

HibernateUtil

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf=null;
	static
	{
		sf=new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf()
	{
		return sf;
	}
}
```

Config file (.cfg.xml)

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
```

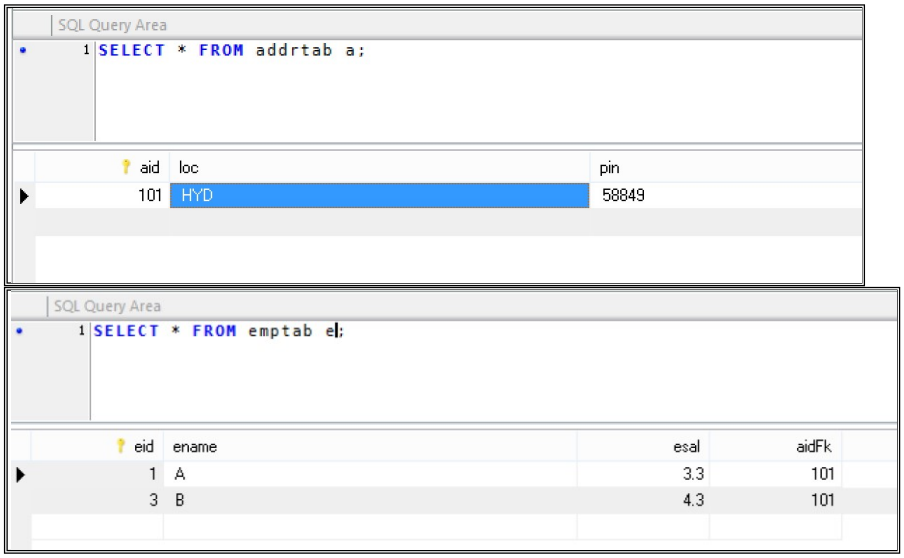
```
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</pr
operty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hbm2ddl.auto">create</property>
<mappingclass="com.app.model.Employee"/>
<mappingclass="com.app.model.Address"/>
</session-factory>
   </hibernate-configuration>
```

Test class:

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Address;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
publicclass Test {
publicstaticvoid main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession())
        {
                tx=ses.beginTransaction();
                Address addr=new Address();
                addr.setAddrId(101);
                addr.setLoc("HYD");
                addr.setPin("58849");
                Employee e1=new Employee();
                e1.setEmpId(1);
                e1.setEmpName("A");
                e1.setEmpSal(3.3);
                e1.setAddr(addr);
                //alt+shift+R -> to rename local variable
                Employee e3=new Employee();
                e3.setEmpId(3);
                e3.setEmpName("B");
                e3.setEmpSal(4.3);
                e3.setAddr(addr);
                ses.save(addr);
                ses.save(e1);
                //ses.save(e2);
```

```
                ses.save(e3);
                tx.commit();
        }
}
}
```

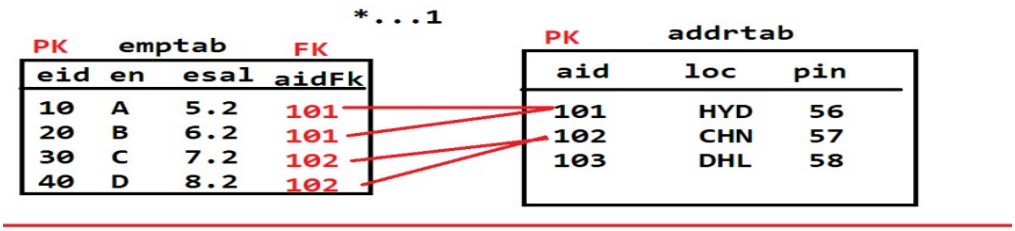
Output:-

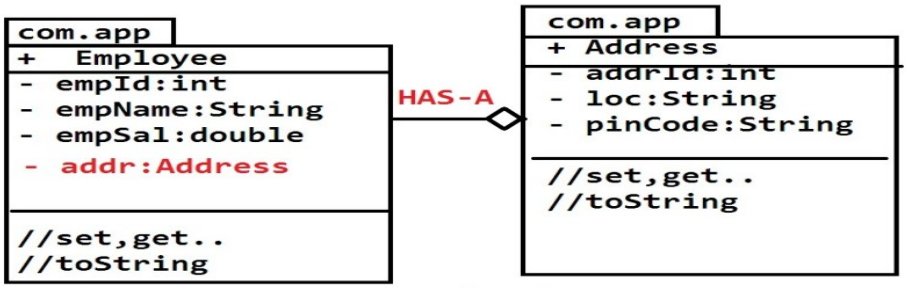


2. ONE-TO-ONE:-

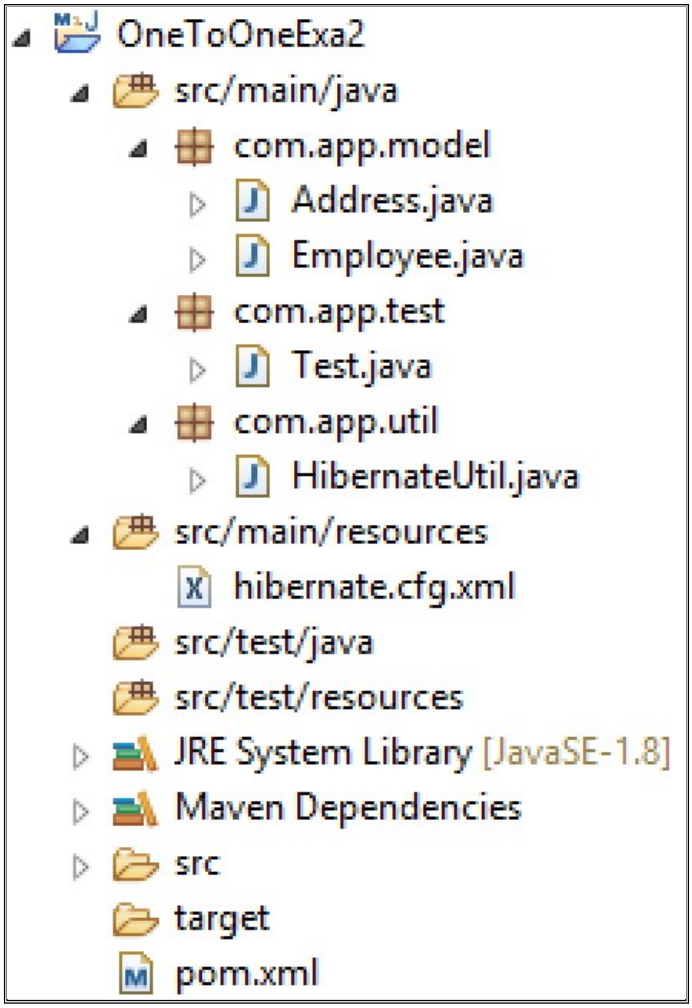
For this case use many-to-one with unique condition over FK column.

- Unique column will not allow duplicates. But null values (multiple) are accepted.
- Null indicates no-data and FK column null indicates no child row.
- To make FK column unique code is:
  @JoinColumn(name="",unique=true)





Folder structure



Pom.xml

```
<projectxmlns="http://maven.apache.org/POM/4.0.0"xmlns:xsi="http://www.w3.org/
2001/XMLSchema-instance"xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>HibernateMavenFirstApp</artifactId>
<version>1.0</version>
<dependencies>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-core</artifactId>
<version>5.1.11.Final</version>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.1.6</version>
</dependency>
</dependencies>
<build>
```

```
<plugins>
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-compiler-plugin</artifactId>
<version>3.1</version>
<configuration>
<source>1.8</source>
<target>1.8</target>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

Model class:-

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="addrtab1")
publicclass Address {
@Id
@Column(name="aid")
privateintaddrId;
@Column(name="loc")
private String loc;
@Column(name="pin")
private String pin;
public Address() {
	super();
}
publicint getAddrId() {
	returnaddrId;
}
publicvoid setAddrId(intaddrId) {
	this.addrId = addrId;
}
public String getLoc() {
	returnloc;
}
publicvoid setLoc(String loc) {
	this.loc = loc;
}
public String getPin() {
	returnpin;
```

```
}
publicvoid setPin(String pin) {
        this.pin = pin;
}
@Override
public String toString() {
        return"Address [addrId=" + addrId + ", loc=" + loc + ", pin=" + pin + "]";
}
}
    ////////////////////////////////////////////////////////////////////////////////
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Table;
@Entity
@Table(name="emptab2")
publicclass Employee {
@Id
@Column(name="eid")
privateintempId;
@Column(name="ename")
private String empName;
@Column(name="esal")
privatedoubleempSal;
@ManyToOne
@JoinColumn(name="aidFK",unique=true)
private  Address addr;//HAS-A
public Employee() {
        super();
}
publicint getEmpId() {
        returnempId;
}
publicvoid setEmpId(intempId) {
        this.empId = empId;
}
public String getEmpName() {
        returnempName;
}
publicvoid setEmpName(String empName) {
        this.empName = empName;
```

```
}
publicdouble getEmpSal() {
        returnempSal;
}
publicvoid setEmpSal(doubleempSal) {
        this.empSal = empSal;
}
public Address getAddr() {
        returnaddr;
}
publicvoid setAddr(Address addr) {
        this.addr = addr;
}
@Override
public String toString() {
        return"Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", addr=" + addr + "]";
}
}
```

//////////////HibernateUtil .java//////////////////////////

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
publicclass HibernateUtil {
        privatestatic SessionFactory sf=null;
        static
        {
                sf=new Configuration().configure().buildSessionFactory();
        }
        publicstatic SessionFactory getSf()
        {
                returnsf;
        }
}
```

//////////////////Configuration file////////////////////////////

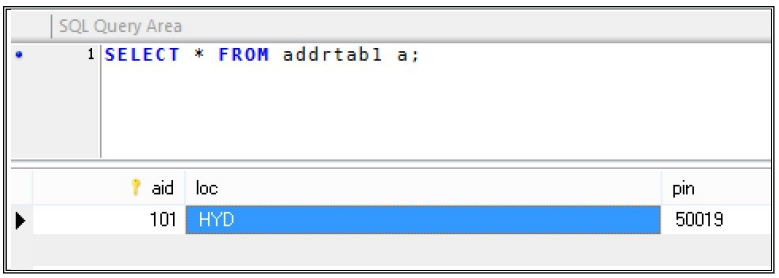
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</pr
operty>
```

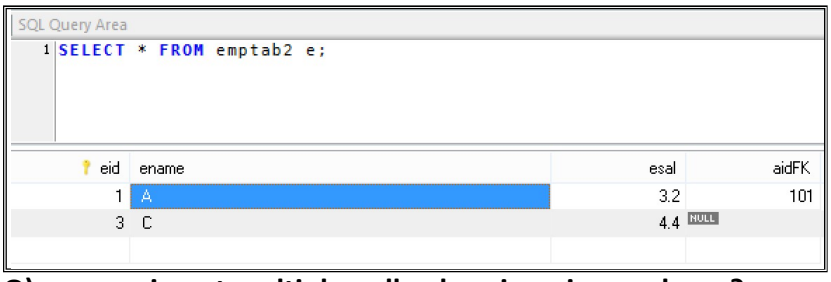
```
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hbm2ddl.auto">create</property>
<mappingclass="com.app.model.Address"/>
<mappingclass="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

//////////////////Test class://////////////////////////

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Address;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
                Transaction tx=null;
                try(Session ses=HibernateUtil.getSf().openSession()){
                        tx=ses.beginTransaction();
                        Address a1=new Address();
                        a1.setAddrId(101);
                        a1.setLoc("HYD");
                        a1.setPin("50019");
                        Employee e1=new Employee();
                        e1.setEmpId(1);
                        e1.setEmpName("A");
                        e1.setEmpSal(3.2);
                        e1.setAddr(a1);
                        Employee e2=new Employee();
                        e2.setEmpId(3);
                        e2.setEmpName("C");
                        e2.setEmpSal(4.4);
                        ses.save(a1);
                        ses.save(e1);
                        ses.save(e2);
                        tx.commit();
                }
        }
}
```

Output





Q). can we insert multiple null values in unique column?

Ans.-> Yes, Unique means no duplicate data but null means no data. So, multiple null values can be inserted to unique column.

Q). can we insert null to any column?

Ans.->Yes , every column (int /varchar/number/BLOB/binary /text/float all types), Supports null as default values in database.

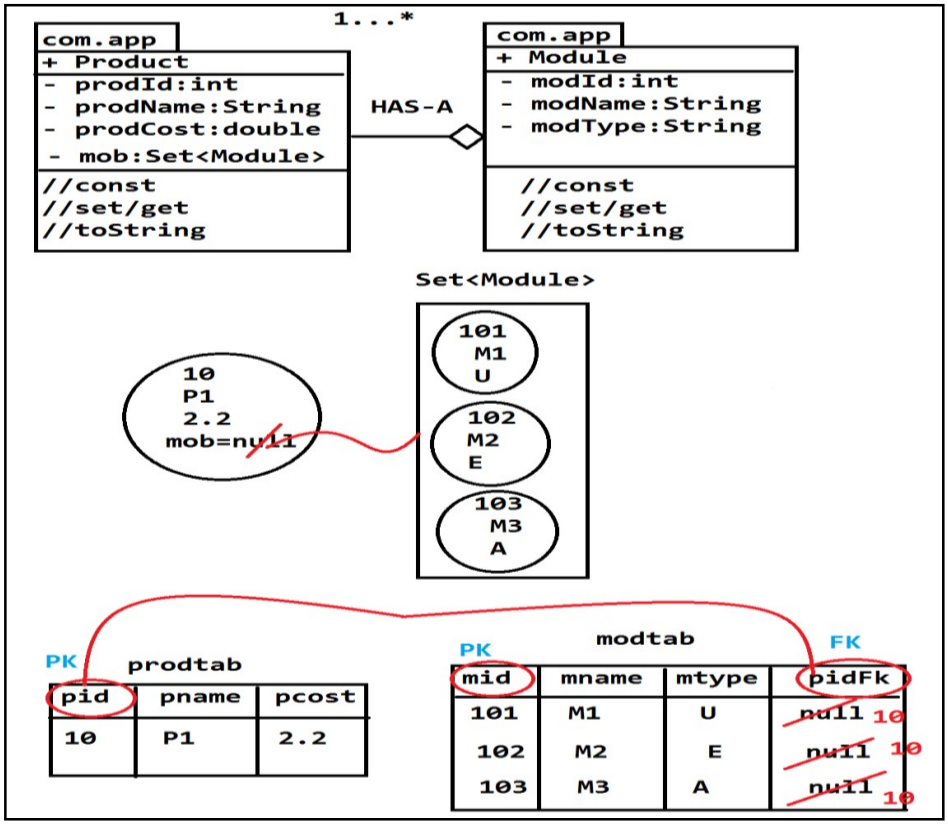
ONE-TO-MANY:

It is collection based multiplicity in Hibernate.

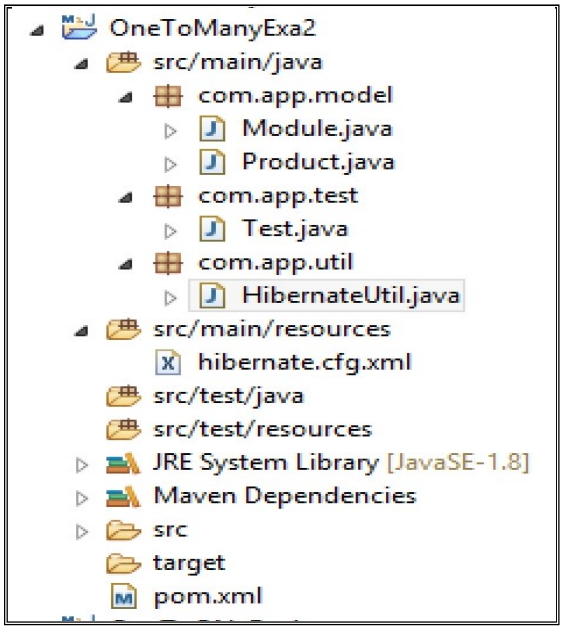
For Parent class create one object and child class create multiple objects and convert them into One collection Type(List,Set….)

- Here first child objects are inserted (insert query for child..) then parent objects are inserted (insert query for parent) them finally parent ID will be updaed with child table FK column (update query for child table).

Example:-



Folder structure



Pom.xml (same as before code)

Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="modtab")
publicclass Module {
@Id
@Column(name="mid")
privateintmodId;
@Column(name="mname")
private String modName;
@Column(name="mtype")
private String modType;
public Module() {
	super();
}
publicint getModId() {
	returnmodId;
}
publicvoid setModId(intmodId) {
	this.modId = modId;
}
public String getModName() {
	returnmodName;
}
publicvoid setModName(String modName) {
	this.modName = modName;
}
public String getModType() {
	returnmodType;
}
publicvoid setModType(String modType) {
	this.modType = modType;
}
@Override
public String toString() {
	return"Module [modId=" + modId + ", modName=" + modName + ", modType=" +
modType + "]";
}
}
```

```
////////////////////////////////////////////////////////////////////////////////////////////
package com.app.model;
import  java.util.HashSet;
import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;
@Entity
@Table(name="prodtab")
publicclass Product {
@Id
@Column(name="pid")
privateintprodId;
@Column(name="pname")
private String prodName;
@Column(name="pcost")
privatedoubleprodCost;
@OneToMany
@JoinColumn(name="pidFk")
private Set<Module>mob=new HashSet<Module>();
public Product() {
	super();
}
publicint getProdId() {
	returnprodId;
}
publicvoid setProdId(intprodId) {
	this.prodId = prodId;
}
public String getProdName() {
	returnprodName;
}
publicvoid setProdName(String prodName) {
	this.prodName = prodName;
}
publicdouble getProdCost() {
	returnprodCost;
}
publicvoid setProdCost(doubleprodCost) {
	this.prodCost = prodCost;
}
```

```
public Set<Module> getMob() {
        returnmob;
}
publicvoid setMob(Set<Module>mob) {
        this.mob = mob;
}
@Override
public String toString() {
        return"Product [prodId=" + prodId + ", prodName=" + prodName + ", prodCost=" +
prodCost + ", mob=" + mob + "]";
}
}
```

////////////////HibernateUtil.java//////////////////////////

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
publicclass HibernateUtil {
        privatestatic SessionFactory sf=null;
        static
        {
                sf=new Configuration().configure().buildSessionFactory();
        }
        publicstatic SessionFactory getSf()
        {
                returnsf;
        }
}
```

////////////configuration file//

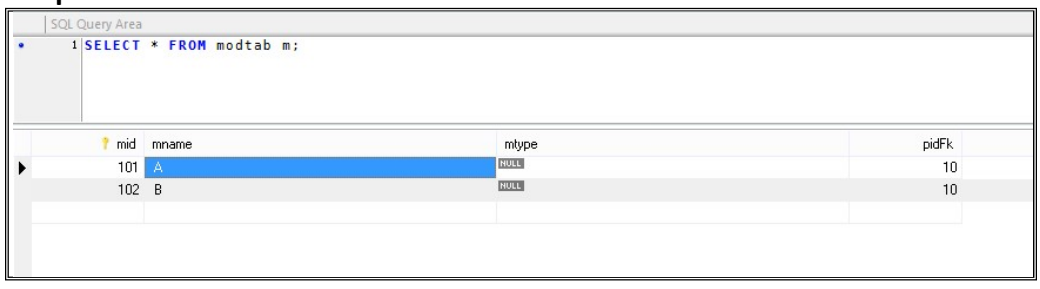
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</pr
operty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hbm2ddl.auto">create</property>
<mappingclass="com.app.model.Module"/>
<mappingclass="com.app.model.Product"/>
</session-factory>
```

```
</hibernate-configuration>
```

////////////////////////////Test class://////////////////////////////

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Module;
import com.app.model.Product;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
                Transaction tx=null;
                try(Session ses=HibernateUtil.getSf().openSession()){
                        tx=ses.beginTransaction();
                        Module m1=new Module();
                        m1.setModId(101);
                        m1.setModName("A");
                        //m1.setModCost(3.3);
                        Module m2=new Module();
                        m2.setModId(102);
                        m2.setModName("B");
                        //m2.setModCost(2.3);
                        Product p=new Product();
                        p.setProdId(10);
                        p.setProdName("A");
                        //p.setCost(2.2);
                        p.getMob().add(m1);
                        p.getMob().add(m2);
                //      p.getMob().add(m3);
                        ses.save(m1);
                        ses.save(m2);
                        //ses.save(m3);
                        ses.save(p);
                        tx.commit();
                }
        }
}
```

Output:-



```
SQL Query Area
1  SELECT * FROM prodtab p;
```

pid	pcost	pname
10	2.2	A

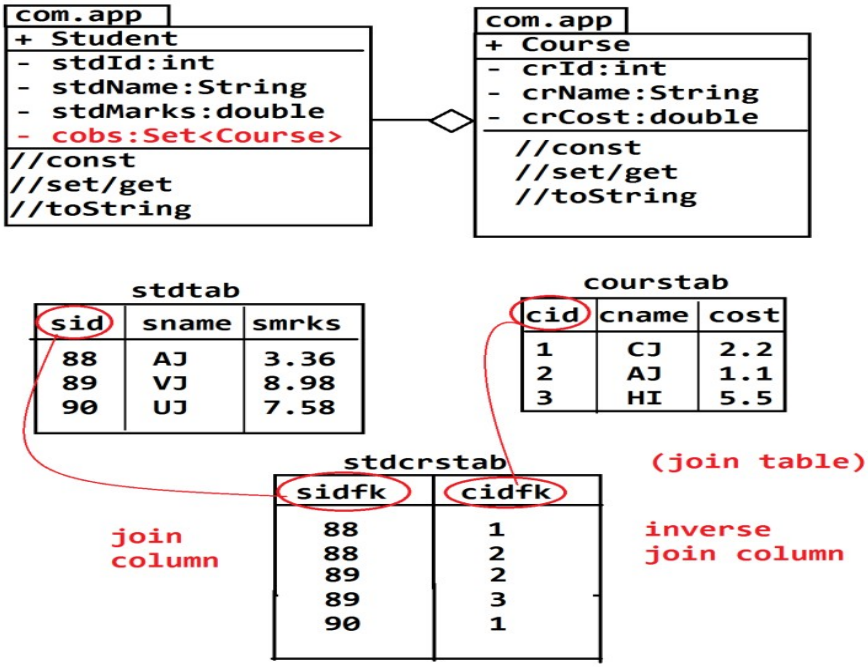
- **Many-To-Many:-**

In two model classes are connected using HAS-A relation and both sides executing one-to-many relation, then it is consider as many to many.
Ex:- Ones student can learn mulitiple courses (1 student =* course) in same way one course can be taken by multiple students(1 c
That is both sides many to many can shows an

...

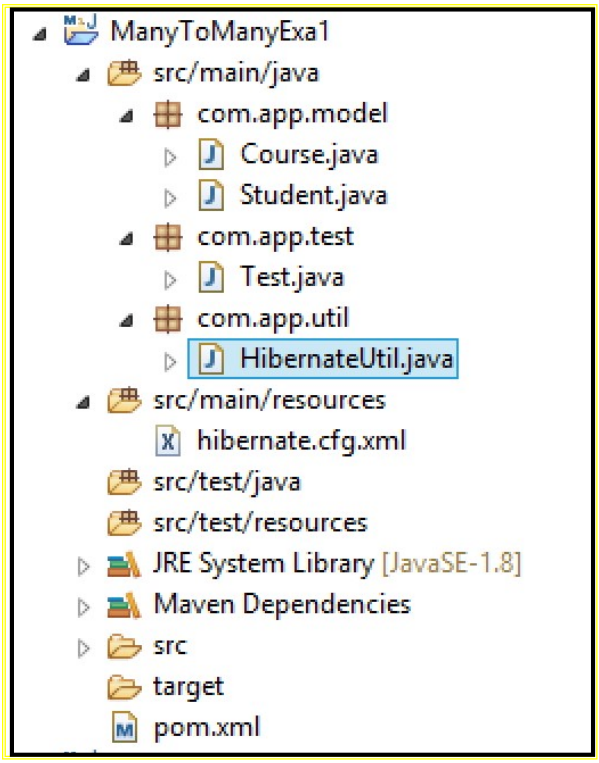
Student ---------- <>Course

HAS-A



==========code==========

Folder structure



Model classes

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="courstab")
publicclass Course {
@Id
@Column(name="cid")
privateintcrId;
@Column(name="cname")
private String crName;
@Column(name="ccost")
privatedoublecrCost;
public Course() {
	super();
}
publicint getCrId() {
	returncrId;
}
publicvoid setCrId(intcrId) {
```

```
		this.crId = crId;
}
public String getCrName() {
		returncrName;
}
publicvoid setCrName(String crName) {
		this.crName = crName;
}
publicdouble getCrCost() {
		returncrCost;
}
publicvoid setCrCost(doublecrCost) {
		this.crCost = crCost;
}
@Override
public String toString() {
		return"Course [crId=" + crId + ", crName=" + crName + ", crCost=" + crCost + "]";
}

}
		////////////////////////////////////////////////////////////////////////////////
```

```
package com.app.model;

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

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.Table;

@Entity
@Table(name="stdtab2")
publicclass Student {
@Id
@Column(name="sid")
privateintstdId;
@Column(name="sname")
private String stdName;
@Column(name="smarks")
privatedoublestdMarsk;
```

```
@ManyToMany
@JoinTable(name="stdcrstab",joinColumns=@JoinColumn(name="sidFk"),
inverseJoinColumns=@JoinColumn(name="cidFk")
)
private Set<Course>cobs=new HashSet<Course>(0);///HAS-A
public Student() {
	super();
}
publicint getStdId() {
	returnstdId;
}
publicvoid setStdId(intstdId) {
	this.stdId = stdId;
}
public String getStdName() {
	returnstdName;
}
publicvoid setStdName(String stdName) {
	this.stdName = stdName;
}
publicdouble getStdMarsk() {
	returnstdMarsk;
}
publicvoid setStdMarsk(doublestdMarsk) {
	this.stdMarsk = stdMarsk;
}
public Set<Course> getCobs() {
	returncobs;
}
publicvoid setCobs(Set<Course>cobs) {
	this.cobs = cobs;
}
@Override
public String toString() {
	return"Student [stdId=" + stdId + ", stdName=" + stdName + ", stdMarsk=" + stdMarsk +
", cobs=" + cobs + "]";
}
}
```

////////////////HibernateUtil.java//////////////////////

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf=null;
```

```
		static
		{
			sf=new Configuration().configure().buildSessionFactory();

	}
		public static SessionFactory getSf() {
			return sf;
		}
	}
```

////////////////Configuratin file//////////////////

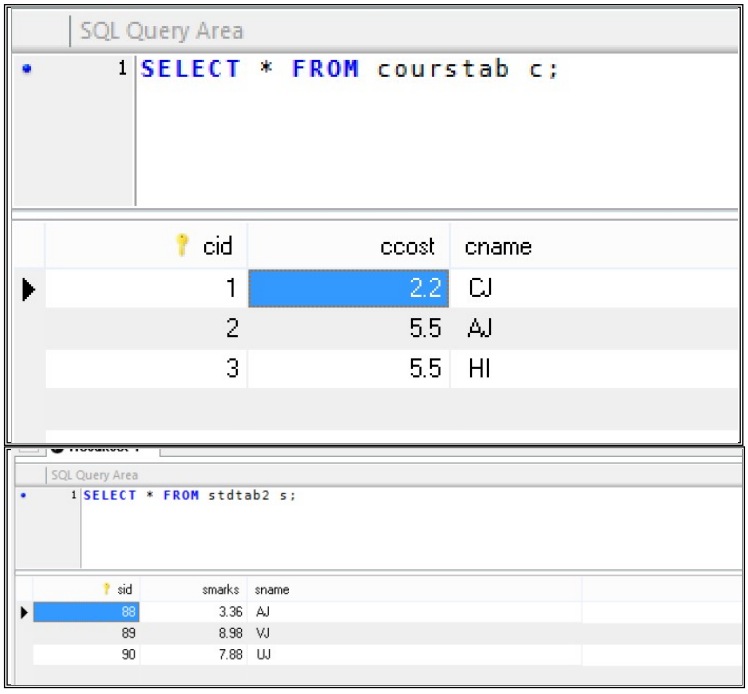
```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
<propertyname="hibernate.connection.url">jdbc:mysql://localhost:3306/ramdb</pr
operty>
<propertyname="hibernate.connection.username">root</property>
<propertyname="hibernate.connection.password">system</property>
<propertyname="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</propert
y>
<propertyname="hibernate.show_sql">true</property>
<propertyname="hibernate.format_sql">true</property>
<propertyname="hibernate.hbm2ddl.auto">create</property>
<mappingclass="com.app.model.Course"/>
<mappingclass="com.app.model.Student"/>
</session-factory>
	</hibernate-configuration>
```

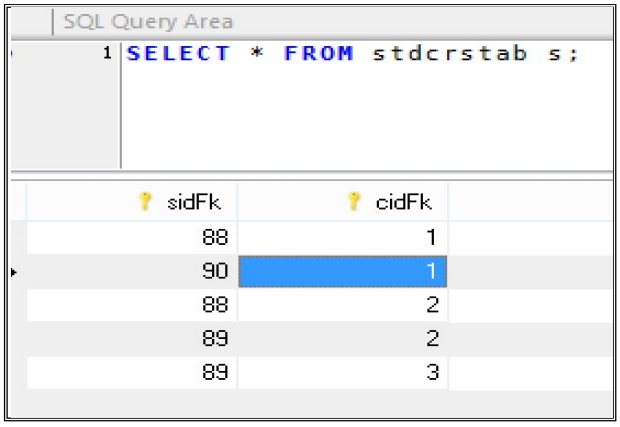
////////////////Test class:////////////////////////

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.app.model.Course;
import com.app.model.Student;
import com.app.util.HibernateUtil;
publicclass Test {
publicstaticvoid main(String[] args) {
	Transaction tx=null;
	try(Session ses=HibernateUtil.getSf().openSession())
	{
		tx=ses.beginTransaction();
```

```
                Course c1=new Course();
                c1.setCrId(1);
                c1.setCrName("CJ");
                c1.setCrCost(2.2);
                Course c2=new Course();
                c2.setCrId(2);
                c2.setCrName("AJ");
                c2.setCrCost(5.5);
                Course c3=new Course();
                c3.setCrId(3);
                c3.setCrName("HI");
                c3.setCrCost(5.5);
                Student s1=new Student();
                s1.setStdId(88);
                s1.setStdName("AJ");
                s1.setStdMarsk(3.36);
                s1.getCobs().add(c1);
                s1.getCobs().add(c2);
                Student s2=new Student();
                s2.setStdId(89);
                s2.setStdName("VJ");
                s2.setStdMarsk(8.98);
                s2.getCobs().add(c2);
                s2.getCobs().add(c3);
                Student s3=new Student();
                s3.setStdId(90);
                s3.setStdName("UJ");
                s3.setStdMarsk(7.88);
                s3.getCobs().add(c1);
                ses.save(c1);
                ses.save(c2);
                ses.save(c3);
                ses.save(s1);
                ses.save(s2);
                ses.save(s3);
                tx.commit();
                //ses.close();
        }
}
}
```

Output:-


SQL Query Area
1 SELECT * FROM courstab c;
cid ccost cname
1 2.2 CJ
2 5.5 AJ
3 5.5 HI
SQL Query Area
1 SELECT * FROM stdtab2 s;
sid smarks sname
88 3.36 AJ
89 8.98 VJ
90 7.88 UJ

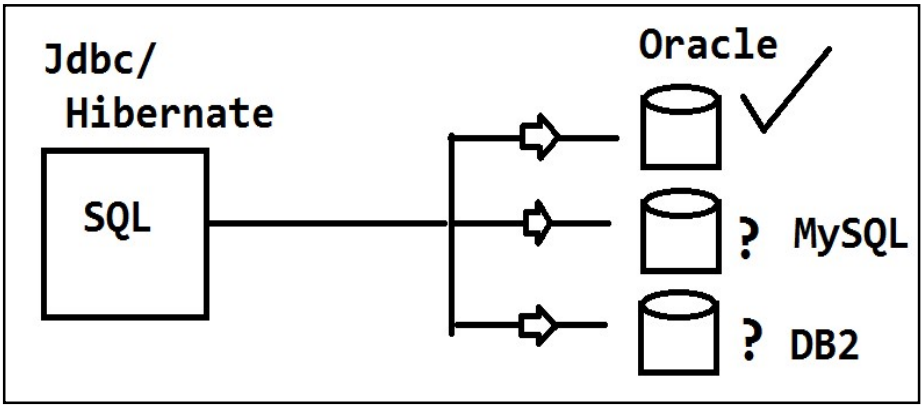

SQL Query Area
1 SELECT * FROM stdcrstab s;
sidFk cidFk
88 1
90 1
88 2
89 2
89 3

HQL(HIBERNATE QUERY LANGUAGE):-

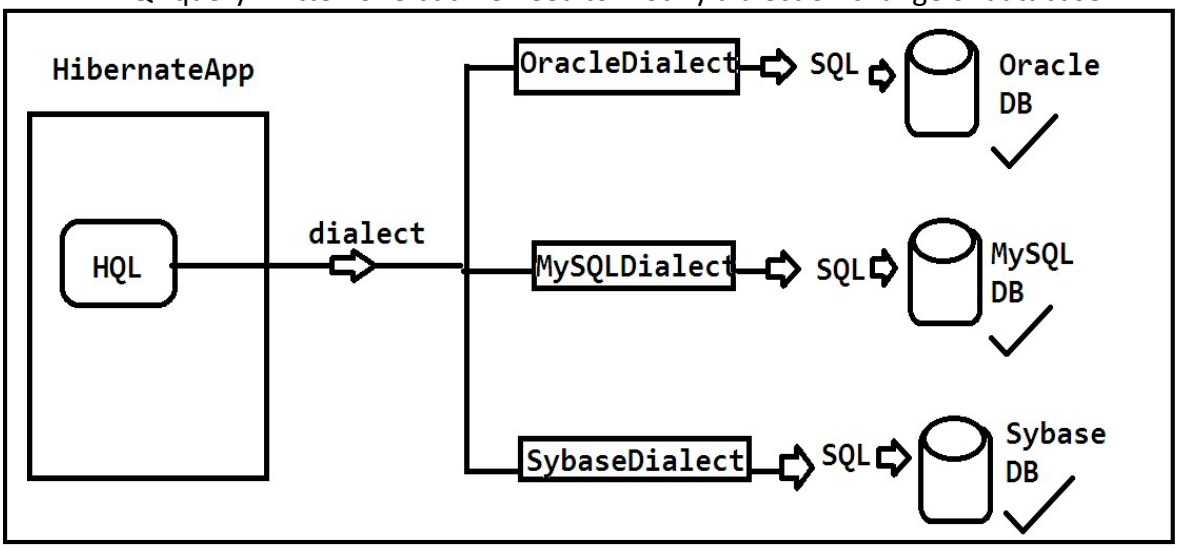
By session operation like save(),update(),delete(),get(),load(),saveOrUpdate()..

We can perform single row operation.

- To perform multi row operation hibernate has provided HQL Concept. It will support both select and Non-Select operation(update,delete,and copy)
- *** HQL will not support multi-row insert.
- SQL is data base dependent it means query written to one database may not work with another database.



- HQL query written one but we need to modify dialect on change of database.



- SQL query incasesensetive that is upper case letter or lower case letter are allowed in writing query

Ex: select eid,ename from emptab

SELECT eid,Ename from empTab

Select Eid, ENAME from EMPTAB

All are valid

- HQL is partially case sensitive that is java related word case sensitive and SQL related words (**select ,update, delete, from, set,group by having…..)** case sensitive
- To convert HQL to SQL replace with column name with variable name table name with class name.

Ex:- SQL:-

Select eid,ename from emptab

HQL:

Select empIde, empName from com.app.model.Employee

Ex#2:-

SQL:- update emptab set ename=?,esal=? Where eid=?

HQL:-

UPDATE com.app.model.Employee set empName=?,empSal=? Where empId=?

EX#3:-***

SQL:-

select *from emptabl

HQL:-

from com.app.model.Employee

EX#4:

SQL:-

delete from emptabl where eid=?

HQL:-

delete from com.app.model.Employee where empId=?

Ex#5

SQL:-

select count(eid) from emptab group by eid;

HQL:-

Select count (empId) from com.app.Employee group by empId;

HQL SELECT: we can use select queries to fetch data DB tables (Multiple rows). Final output is given by Hibernate is java.util.List(no.of rows=no.of objects>> stored in List Collection only)

FULL LOADING :-select all columns using Query(HQL/SQL) is known as Full loading (One Full row = one complete Model class Object). So final output will be List<T>, T=Type/Model class Name

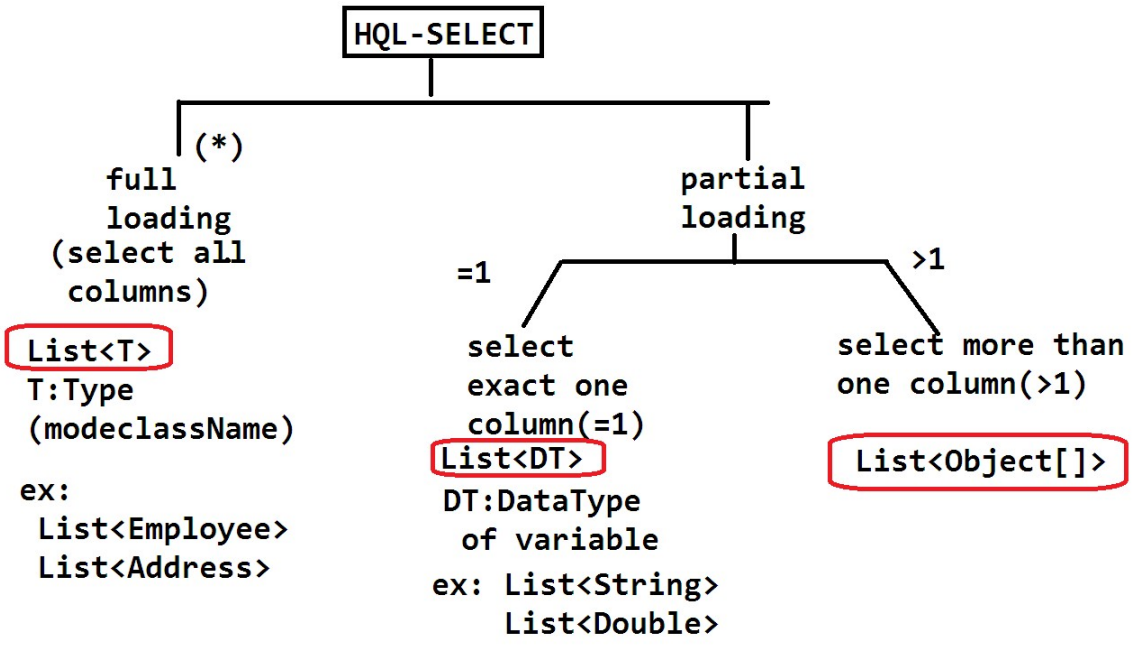
PARTIAL LOADING : selecting one column (=1 column) or more then one columns (>1 column) is known as Partial loading.

Final outoput is given as:

= 1 column: List<DT>

DT=DataType of varivales/column

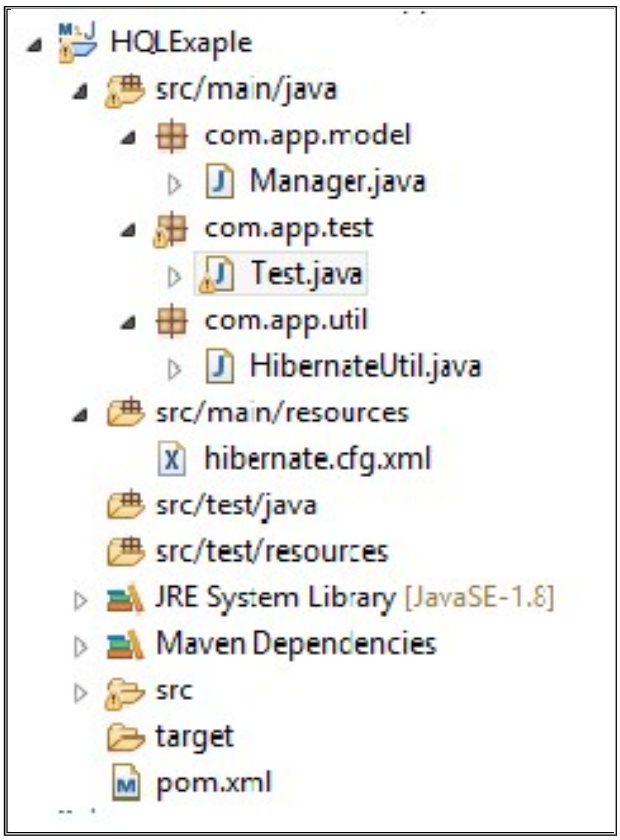
>1 column :List<Object[]>



HQL CODING:- in Test class after creating session object , follow below steps:

1. Define one HQL String
   Ex: String hql="….";
2. Create Query object using session object
   Query q=ses.createQuery(hql);
3. Execute query object using list() method
   List<_> data=q.list();
   Exa: data select using HQL

Folder Structure



1. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>org.nareshittech</groupId>
	<artifactId>HQLExaple</artifactId>
	<version>1.0</version>
	<dependencies>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.0.5</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
	</dependencies>
	<build>
```

```
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.0</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
                </plugin>
            </plugins>
        </build>
</project>
```

2. Configuration file(cfg.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Manager"/>
</session-factory>
</hibernate-configuration>
```

3. Model Class:-

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="managertab")
public class Manager {
        @Id
        @Column(name="mid")
        private int mngId;
```

```
	@Column(name="mname")
	private String mngName;
	@Column(name="mfee")
	private double mngFee;
	public Manager() {
		super();
	}
	public int getMngId() {
		return mngId;
	}
	public void setMngId(int mngId) {
		this.mngId = mngId;
	}
	public String getMngName() {
		return mngName;
	}
	public void setMngName(String mngName) {
		this.mngName = mngName;
	}
	public double getMngFee() {
		return mngFee;
	}
	public void setMngFee(double mngFee) {
		this.mngFee = mngFee;
	}
	@Override
	public String toString() {
		return "Manager [mngId=" + mngId + ", mngName=" + mngName + ", mngFee=" +
mngFee + "]";
	}
}
```

4. **HibernateUtil(.java)**

```
	package com.app.util;
	import org.hibernate.SessionFactory;
	import org.hibernate.cfg.Configuration;
	public class HibernateUtil {
	private static SessionFactory sf=null;
	  static
		{
			sf=new Configuration().configure().buildSessionFactory();
		}
	public static SessionFactory getSf()
		{
					return sf;
```

```
            }
        }
```

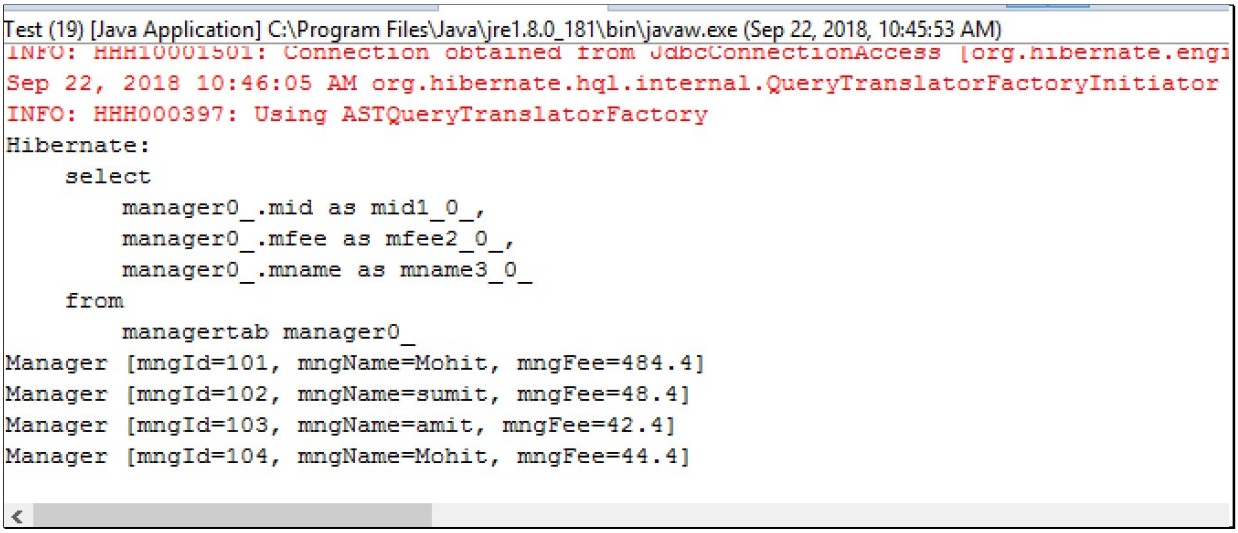
5. Test class:

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
//Insert Data Code
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
        tx=ses.beginTransaction();
        Manager m1=new Manager();
        m1.setMngId(101);
        m1.setMngName("Mohit");
        m1.setMngFee(484.4);
        Manager m2=new Manager();
        m2.setMngId(102);
        m2.setMngName("sumit");
        m2.setMngFee(48.4);
        Manager m3=new Manager();
        m3.setMngId(103);
        m3.setMngName("amit");
        m3.setMngFee(42.4);
        Manager m4=new Manager();
        m4.setMngId(104);
        m4.setMngName("Mohit");
        m4.setMngFee(44.4);
        ses.save(m1);
        ses.save(m2);
        ses.save(m3);
        ses.save(m4);
        tx.commit();
        ses.close();
        }catch(Exception ex)
        {

        }
}
}
```

//Example Display All Record

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
        =======//Display all record=======
                String hql="from com.app.model.Manager";
                Query q=ses.createQuery(hql);
                List<Manager>man=q.list();
                for(Manager m:man)
                {
                        System.out.println(m);
                }

        }catch(Exception ex)
        {

        }
}
}
```

OutPut:-(Display All Record)



Example Code:-// Select one Record Only

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
        String hql="select mngName from com.app.model.Manager ";
                Query q=ses.createQuery(hql);
                List<String>list=q.list();
                for(String s:list)
                {
                        System.out.println(s);
                }
        }catch(Exception ex)
        {

        }
}
}
```

OutPut:-(Select one Record Only)

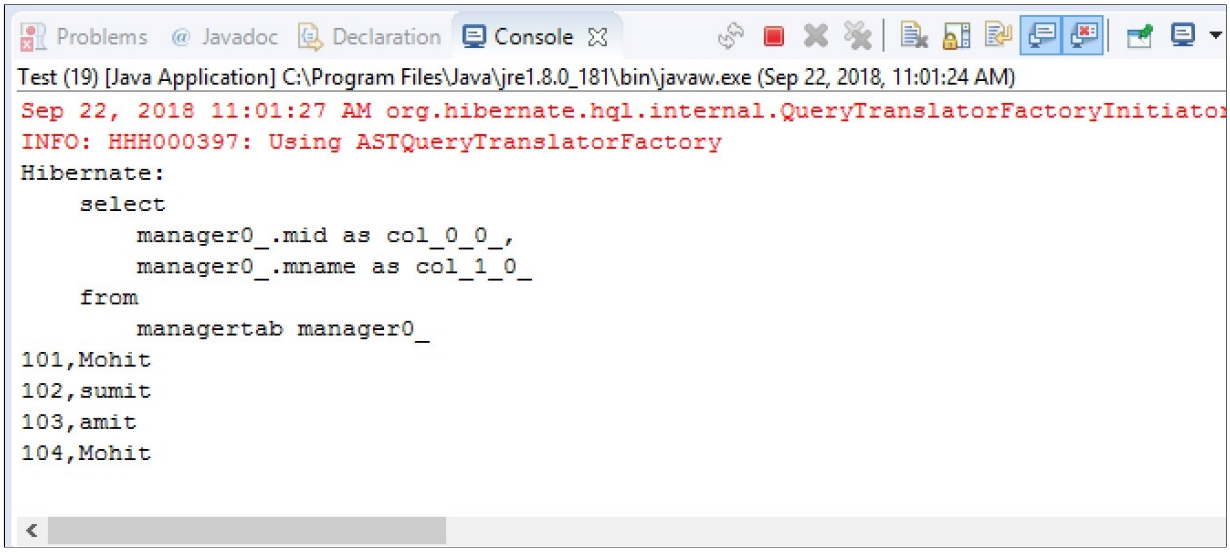
```
Sep 22, 2018 10:52:50 AM org.hibernate.resource.transaction.backend.jdbc.internal.
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.en
Sep 22, 2018 10:52:51 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiato
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        manager0_.mname as col_0_0_
    from
        managertab manager0_
Mohit
sumit
amit
Mohit
```

Example Code:- //Select Multiple Record

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
```

```
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()) {
        String hql="select mngId,mngName from com.app.model.Manager";
                Query q=ses.createQuery(hql);
                List<Object[]>list=q.list();
                for(Object[]ob:list)
                {
                System.out.println(ob[0]+","+ob[1]);
                }

        }catch(Exception ex)
        {

        }
}
}
```

Output:- Select Multiple Record



Example Code: Full Loading

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()) {
                String hql="from com.app.model.Manager";
```

```
                Query q=ses.createQuery(hql);
                List<Manager>m=q.list();
                m.forEach(System.out::println);
                }catch(Exception ex)
        {

        }
}
}
```

OutPut:- Full Loading

```
Test (19) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 22, 2018, 11:07:40 AM)
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        manager0_.mid as mid1_0_,
        manager0_.mfee as mfee2_0_,
        manager0_.mname as mname3_0_
    from
        managertab manager0_
Manager [mngId=101, mngName=Mohit, mngFee=484.4]
Manager [mngId=102, mngName=sumit, mngFee=48.4]
Manager [mngId=103, mngName=amit, mngFee=42.4]
Manager [mngId=104, mngName=Mohit, mngFee=44.4]
```

PARAMETER IN HQL:- it is used to pass data to query at runtime.
A parameter in simple called as input to HQL/SQL query at runtime.
Hibernate supports two types of parameters concepts.

1. **Positional Parameters(?)**
2. **Named Parameters (:name)**

1. **Positional Parameters:-** It indicates position numbers of ? Symbols. States from Zero (in hibernate).
    - Starts from 1 in JDBC.
    - One HQL Query can have multiple "?" symbols which indicates data comes at runtime.
    - Index number starts from zero, to set data based on position used method "setParameter(**index,data)"**

    Example:-
    Select eid from emptab
    Where eid> ?------ 0
    Or ename= ? -------1
    And esal < ? -------- 2
    Select eid from emptab
    Where eid< ?----------0

And eid > ?----------0---0
Or ename =? ---------- 1

- **Position number may get changed if query gets changes**

====Example==========

1. **Model class, cfg,util are same as before**
2. **Test class:// cfg,sf,ses**

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()) {
String hql="from com.app.model.Manager where mngId=? or mngName=?";
        Query q=ses.createQuery(hql);
        q.setParameter(0, 102);
        q.setParameter(1, "AA");
        List<Manager>m=q.list();
                m.forEach(System.out::println);
        }catch(Exception ex)
        {

        }
}
}
```

OutPut:- (Positional Paramerter)

```
Test (19) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 22, 2018, 11:17:20 AM)
WARN: [DEPRECATION] Encountered positional parameter near line 1, column 53 in HQ
Hibernate:
    select
        manager0_.mid as mid1_0_,
        manager0_.mfee as mfee2_0_,
        manager0_.mname as mname3_0_
    from
        managertab manager0_
    where
        manager0_.mid=?
        or manager0_.mname=?
Manager [mngId=102, mngName=sumit, mngFee=48.4]
```

2) Named Parameters:- it is used to provide a name in place of ? symbole, to indicate data comes at runtime.

*** This is new concept in Hibernate, not exist in JDBC**

*** Name should be unique in HQL (duplicates not allowed)**

****** We can use variableName as parameter name also.**

- Name never gets changed if query is changed.
- To pass data at runtime code is : **setParameter(name,data)**
- **Syntax is : name (colon name)**

Ex#1:

==========code==========

Test class:// cfg,sf,ses

```
package com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()) {
String hql="from com.app.model.Manager where mngId=:a or mngName=:b";
        Query q=ses.createQuery(hql);
        q.setParameter("a", 101);
        q.setParameter("b", "aa");
        List<Manager>m=q.list();
        m.forEach(System.out::println);
        }catch(Exception ex)
        {
        }
}
}
```

OutPut:- (Named Parameter)

```
Test (19) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 22, 2018, 11:26:23 AM)
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        manager0_.mid as mid1_0_,
        manager0_.mfee as mfee2_0_,
        manager0_.mname as mname3_0_
    from
        managertab manager0_
    where
        manager0_.mid=?
        or manager0_.mname=?
Manager [mngId=101, mngName=Mohit, mngFee=484.4]
```

Q. Can we use positional and named parameters in one query ?

Ans.:- YES, But condition is once named parameters started then ? symbols not allowed. If written then, hibernate thows **QuerySysntexException: cannot define positional parameter after any named parameters have been defined.**

Ex :

Orders :

? ? ? ? ----- valid

? ? :b :a----- valid

:a :b ? ? ----invalid

? ? :a ? -----invalid

In clause:- To work with random rows in DB table use in clause.

Syntax:

Select ….from …

Where column in (values);

- To handle this in Hibernate
- Used named parameters
- Create values collection
- Call setParameterList method

========code=====

Test class:// cfg,sf,ses

String hql="from com.app.model.Employee where empId in (:a)";

Query q=ses.createQuery(hql);

List<Integer> al=Arrays.asList(10,12,14,8);

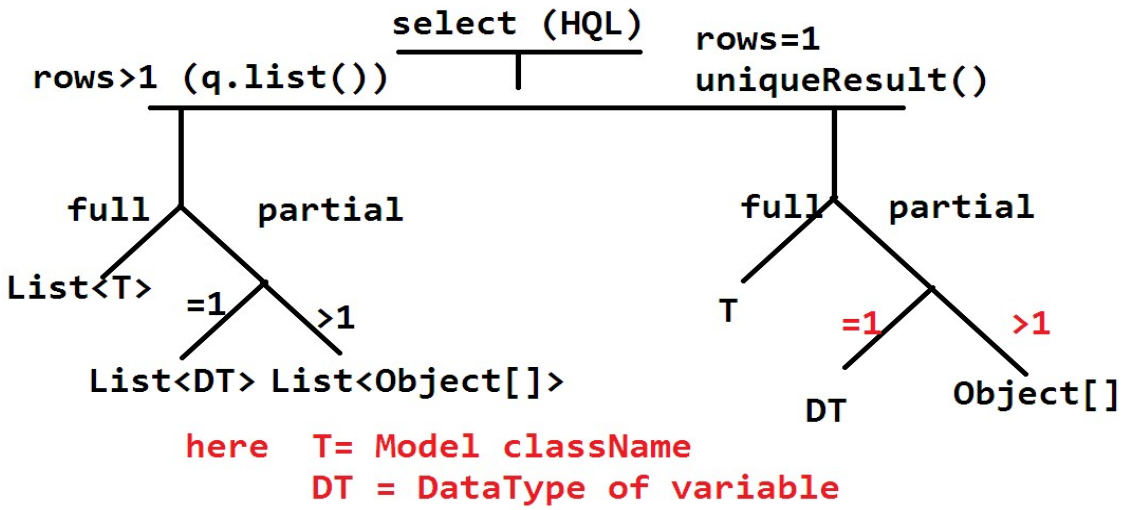
q.setParameteList("a",al);

List<Employee> e=q.list();

e.forEach(System.out::println);

uniqueResult():- this method is used for select HQL operation. If query returns one row data then choose this method instead of query. List() method.

- if will save memory, by avoiding list object for one row data.



Ex:-

Test class:/// cfg,sf,ses

Case#1 one=row with all columns

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session   ses=HibernateUtil.getSf().openSession())  {
String hql="from com.app.model.Manager where mngId=104";
        Query q=ses.createQuery(hql);
        Manager m=(Manager)q.uniqueResult();
        System.out.println(m);
                }catch(Exception ex)
        {
        }
}
}
```

OutPut:-

```
Test (19) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 22, 2018, 11:34:22 AM)
Sep 22, 2018 11:34:25 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiato
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        manager0_.mid as mid1_0_,
        manager0_.mfee as mfee2_0_,
        manager0_.mname as mname3_0_
    from
        managertab manager0_
    where
        manager0_.mid=104
Manager [mngId=104, mngName=Mohit, mngFee=44.4]
```

======case#2===== one row one column=======

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()) {
String hql="select mngName from com.app.model.Manager where mngId=102";
        Query q=ses.createQuery(hql);
        String mname= (String)q.uniqueResult();
        System.out.println(mname);
        }catch(Exception ex)
        {
        }
}
}
```

OutPut:-

```
Test (19) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 22, 2018, 11:39:32 AM)
Sep 22, 2018 11:39:35 AM org.hibernate.resource.transaction.backend.jdbc.internal
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.e
Sep 22, 2018 11:39:35 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiat
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        manager0_.mname as col_0_0_
    from
        managertab manager0_
    where
        manager0_.mid=102
sumit
```

=======case#3====== one row= multiple columns=====

```
package  com.app.test;
import org.hibernate.Query;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()) {
String hql="select mngName,mngFee from com.app.model.Manager where mngId=101";
        Query q=ses.createQuery(hql);
        Object[]m=(Object[])q.uniqueResult();
        System.out.println(m[0]+","+m[1]);
        }catch(Exception ex)
        {
        }
}
}
```

Output:-

```
Test (19) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 22, 2018, 11:41:51 AM)
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.e
Sep 22, 2018 11:41:54 AM org.hibernate.hql.internal.QueryTranslatorFactoryInitiat
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        manager0_.mname as col_0_0_,
        manager0_.mfee as col_1_0_
    from
        managertab manager0_
    where
        manager0_.mid=101
Mohit,484.4
```

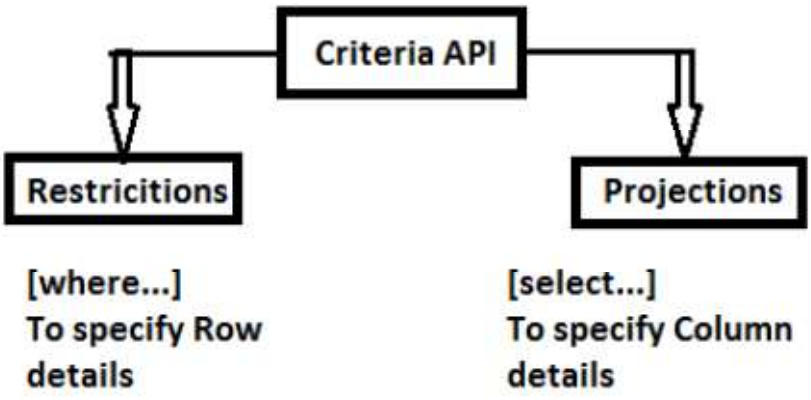
- if query return multiple rows data and still used uniqueResult() method, then hibernate throws Exception like **NonUniqueResultException** : query did not return a unique result: 3

Criteria API: -

This concept is given by Hibernate to perform “Multi Row-Select” Operations.
It will not select Non-Select Operations.
No external query should be written by Programmer.
It is advanced concept of HQL (Query API).
It supports Data loading from table to Application as List.
By default it will select “All Rows and All Columns”.
It supports two child API’s: -

1] Restrictions (To specify where clause)

2] Projections (To specify select clause).



#1.Create Criteria Object using Session.

```
Criteria c = ses.createCriteria(T.class);
```

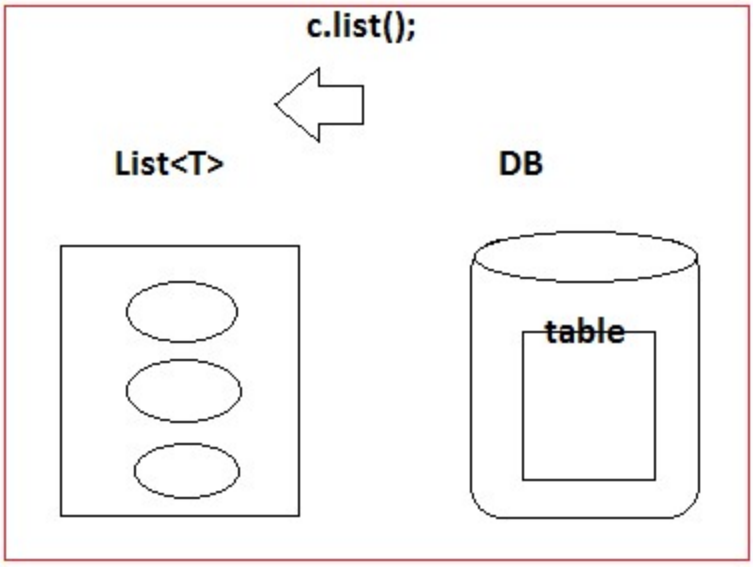
#2. Call list() /uniqueResult for execution

```
List<T> data = c.list();
```

#3. Print data

```
Sysout(data);
```

By default "Criteria API" will execute query like "select * from <table>".
It means,selecting all columns (full loading) and all rows.
Complete table rows are loaded into Session cache as a List<T>.



Syntax To Create Criteria Object: -

By using createCriteria() method which is in Session we can create Criteria object by passing

#1] .class Parameter

#2] Fully Qualified Class Name

Syntax#1

```
Criteria c = ses.createCriteria(T.class);
```

List <T> data = c.list();

Syntax#2

Criteria c = ses.createCriteria("com.app.Employee");

List<T> data = c.list();

Example: Test.java

```
//Create Criteria Object with Session
        Criteria c = ses.createCriteria(Employee.class);
/* OR
* Criteria c = ses.createCriteria("com.sathyatech.model.Employee");
*/
//Call list()/uniqueResult() method
List<Employee> emps = c.list();
```

Criteria API With Projections: -

To select required columns from DB table use Projections API with Criteria Object.

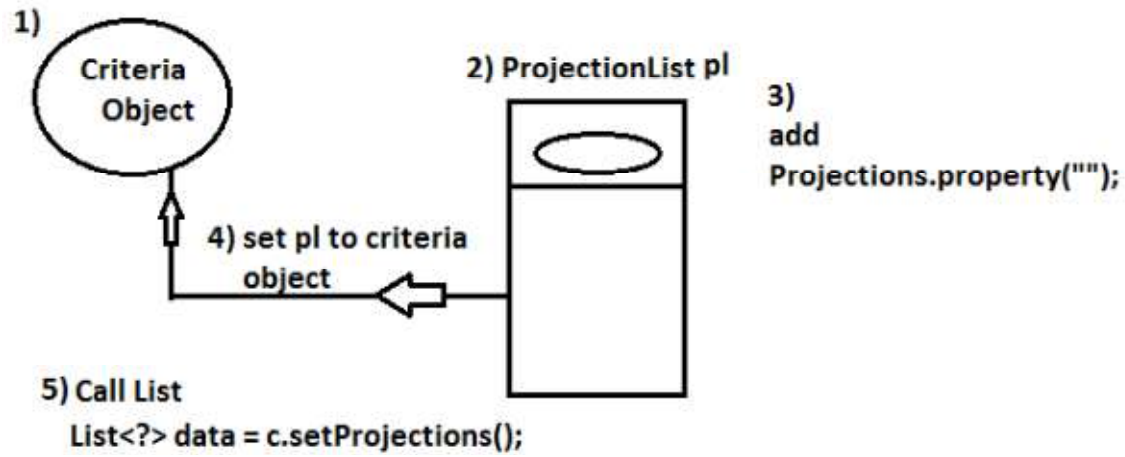
Projections API is a child API of Criteria API. That means without Criteria, Projections will not work.

Using only Criteria (no projections) indicates full loading.

Criteria with Projections indicates partial loading.

--Steps to implement Projections—

- #1 Create Criteria Object using session
- #2 Create empty "ProjectionsList"
- #3 Add one by one "Projections Property to Projections List"
- #4 Set PL to Criteria Object
- #5 call list() method



Example: Criteria API Projection – One Or More Column ->

```
//Create Criteria Object with Session
Criteria c = ses.createCriteria(Employee.class);
/* OR
* Criteria c = ses.createCriteria("com.sathyatech.model.Employee");
*/
//Create empty ProjectionList obj
ProjectionList pl = Projections.projectionList();
```

```
//add Property to ProjectionList
pl.add(Projections.property("empName"));
pl.add(Projections.property("empSal"));
//Set ProjectionList to criteria Obj
c.setProjection(pl);
//Call list()/uniqueResult() method
List<Object[]> emps = c.list();
//Print Data
for(Object[] ob:emps) {
System.out.println(ob[0]);
System.out.println(ob[1]);
}
```

Restrictions: -

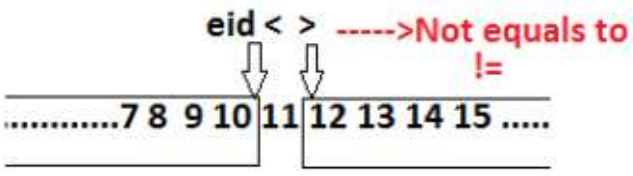
A Restrictions is a child API of Criteria. It is used to specify "where clause", to select rows (not all).
To create where clause conditions, Restrictions class has provided methods for every symbol to indicate one condition.
EX:

SYMBOL	METHOD
>	gt
>=	ge
<	lt
<=	le
<> / !=	ne
=	eg
And	and
Or	or
Like	(like/ilike)
in	in
not in	not in

Steps To implement Restrictions: -
#1 Create Criteria Object
#2 Create Restrictions and add to Criteria Object ***
#3 Call list() / uniqueResult() method
1) Requirement: -

select * from emptab where **empId>11**

```
//select * from emptab
Criteria c = ses.createCriteria("Employee.class");
//where empId>=11
c.add(Restrictions .le("empId",11));
```



Restrictions API Examples: -

```
1)//eid>10
Restrictions.gt(“empId”,10);
2)//eid>=10
Restrictions.ge(“empId”,10);
3)//eid<10
Restrictions.lt(“empId”,10);
4)//eid<=10
Restrictions.le(“empId”,10);
5)//eid=10
Restrictions.eq(“empId”,10);
6)//eid<>10 / eid!=10
Restrictions.ne(“empId”,10);
7)//eid>=10 and eid<=20
Restrictions.between(“empId”,10,20);
8)//ename is null
Restrictions.isNull(“empName”);
9)//ename is not null
Restrictions.isNotNull(“empName”);
10)//ename exactly 3 chars
//ename like ‘_ _ _’
Restrictions.like(“empName”,”___”);
11)//ename like ‘_a%’ or ename like ‘_A%’
//ignore case(upper & lower accepted)
Restrictions.ilike(“empName”,”_A%”);
12)//eid in (10,36,49)
Restrictions.in(“empId”,new Object[]{10,36,49});
13)//eid>10 or esal<=500
Restrictions.or(Restrictions.gt(“empId”,10),Restrictions.le(“empSal”,500.0));
14)//eid>10 and esal<=500
Restrictions.and(Restrictions.gt(“empId”,10),Restrictions.le(“empSal”,500.0));
```

Example: CriteriaAPI Restriction One Or More Row:

```
//Create Criteria Object with Session
Criteria c = ses.createCriteria(Employee.class);
/* OR
* Criteria c = ses.createCriteria("com.sathyatech.model.Employee");
                        */
//Create Restrictions and add to Ctriteria Obj
c.add(Restrictions.le("empId", 11));
//Call list()/uniqueResult() method
List<Employee> emps = c.list();
//Print Data
for(Employee e:emps) {
System.out.println(e);
}
```

Q) Write Criteria API code for below query?

SQL:- select ename,esal from emptab where (eid>10 or eid!=11) and (ename not null or esal >= 52.36)

```
Criteria c = ses.createCriteria(Employee.class);
c.add(Restrictions.and(Restrictions.or(Restrictions.gt("empId",10),Restrictions.ne("empId",11)),
Restrictions.or(Restrictions.NotNull("empName"),Restrictions.ge("empSal",52.36)))
ProjectionList pl = Projections.projectionList();
pl.add(Projections.property("empName"));
pl.add(Projections.property("empSal"));
c.setProjection(pl);
List<Object[]> emps = c.list();
for(Object[] ob:emps){
Sysout(ob[0]);
Sysout(ob[1]);
        }
```

PAGINATION IN HIBERNATE

It is a process of display few (page size rows in UI/Application, instead of loading all rows from DB table.

EX:- Google Search, Gmail Inbox/sent email , Bank account Transaction.

- **Page:**- Data shown with fixed size (no of rows to be shown in one access) as output .
- **Page Size:**- No.of rows in a page.

===Basic Formula=======

Total rows=tr=no.of rows in table page size =ps=no.of rows per page no.of pages =np= tr/ps+(tr%ps>0?1:0)
Tr/ps =provides no.of full pages
Tr%ps>0?1:0 = provides no.of partial pages

- Full page =no.of rows in page is equals ot page size(ps)
- Partial page =no.of rows in page is less then to page size(ps)
- At max 1(one) and at min zero(0) partial pages may occure in one application.
  *** We can never get tow partial pages

----Example-------

Ex#1:

Tr=18 ps=5
Np= 18/5 + (18%5>0?1:0)
= 3 + 3 >0?1:0
= 3 + 1
= 4

Page number	**1**	**2**	**3**	**4**
Start row	**0**	**5**	**10**	**15**
			.	**16**
			.	**17**
End row	**4**	**9**	**15**	

** For every row, hibernate provides row number which starts from zero(0)

Example **Emptab**

Row id	**eid**	**ename**	**esal**
0	**58**	**A**	**3.6**
1	**12**	**B**	**9.2**
2	**74**	**C**	**8.6**

……

Example 2

tr=126 **ps=40**
Np=3+1=4

Pages	**1**	**2**	**3**	**4**
Start	**0**	**40**	**80**	**120**
				…125
End	39	79	119	

Hibernate Pagination API Method

Query(I) [org.hibernate.query] has provided pagination methods to get required rows from Database Table.

Those are:-

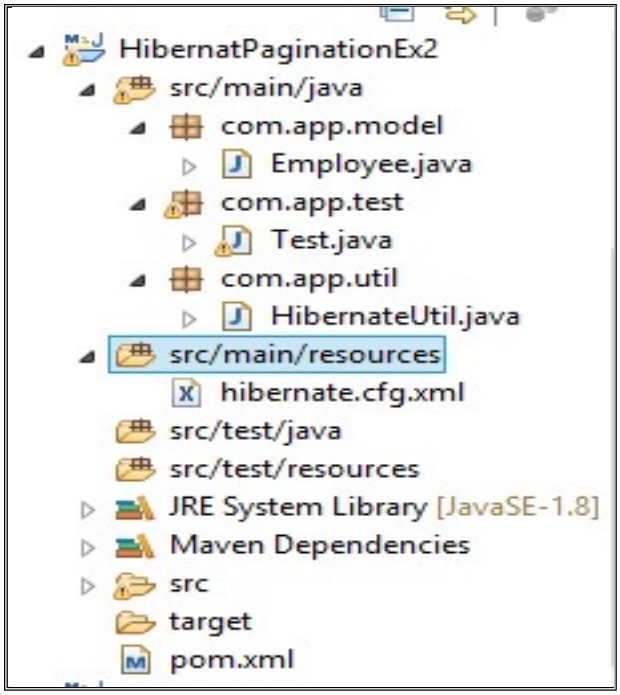
setFirstResult(int index)

setMaxResults(int size)

- Using those methods internaly generates limit clause for SQL query.
- Default value ofr method setFirestResult is zero(0).

- If no valaue is provided for setMaxResult then it will load all rows from current position.
- Ex:- consider below code with briefly explain

1. Folder structure



2. POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>org.nareshittech</groupId>
	<artifactId>HQLJOIN</artifactId>
	<version>1.0</version>
	<dependencies>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.6</version>
		</dependency>
	</dependencies>
```

```
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name = "emptab")
public class Employee {
	@Id
	@Column(name = "eid")
	private int empId;
	@Column(name = "ename")
	private String empName;
	@Column(name = "esal")
	private double empSal;
	public Employee() {
		super();
	}
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
```

```
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName + ", empSal="
+ empSal + "]";
	}
}
```

4. HibernateUtil class

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf = null;
	static {
		sf = new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf() {
		return sf;
	}
}
```

5. Configuration file (cfg.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Employee"/>
```

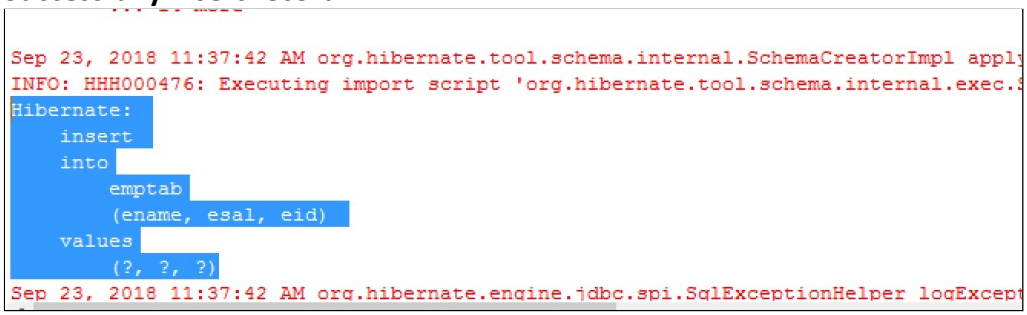
```
</session-factory>
</hibernate-configuration>
```

6. **Test class**

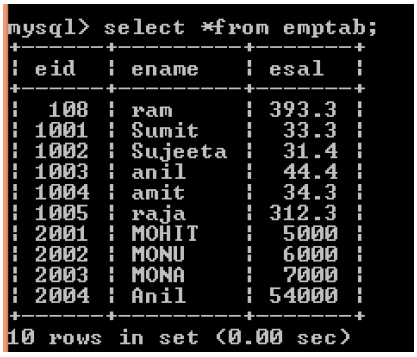
```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
		Transaction tx = null;
		try (Session ses = HibernateUtil.getSf().openSession()) {
			tx=ses.beginTransaction();
			Employee emp=new Employee();
			emp.setEmpId(1001);
			emp.setEmpName("Sumit");
			emp.setEmpSal(33.3);
			 Employee emp1=new Employee();
			emp1.setEmpId(1002);
			emp1.setEmpName("Sujeeta");
			  emp1.setEmpSal(31.4);
			Employee emp2=new Employee();
			emp2.setEmpId(1003);
			emp2.setEmpName("anil");
			emp2.setEmpSal(44.4);
			 Employee emp3=new Employee();
			emp3.setEmpId(1004);
			emp3.setEmpName("amit");
			emp3.setEmpSal(34.3);
			 Employee emp4=new Employee();
			emp4.setEmpId(1005);
			emp4.setEmpName("raja");
			emp4.setEmpSal(312.3);
			ses.save(emp);
			ses.save(emp1);
			ses.save(emp2);
			ses.save(emp3);
			ses.save(emp4);
			ses.save(emp4);
			tx.commit();
		} catch (Exception ex) {
			ex.printStackTrace();
```

```
            }
        } }
```

Successfully insert record



Consider below table with database



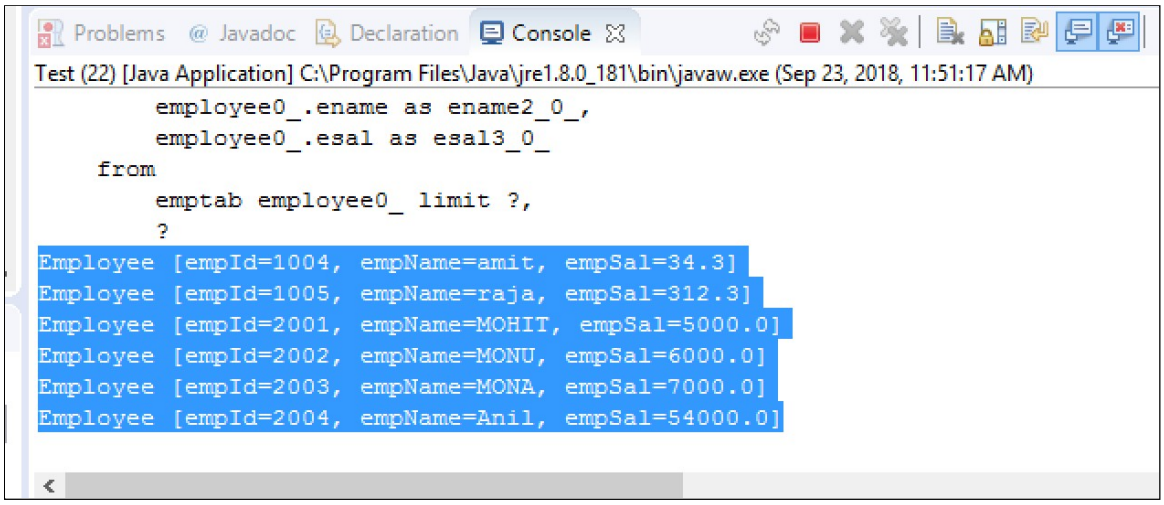
```
mysql> select *from emptab;
+------+---------+-------+
| eid  | ename   | esal  |
+------+---------+-------+
|  108 | ram     | 393.3 |
| 1001 | Sumit   |  33.3 |
| 1002 | Sujeeta |  31.4 |
| 1003 | anil    |  44.4 |
| 1004 | amit    |  34.3 |
| 1005 | raja    | 312.3 |
| 2001 | MOHIT   |  5000 |
| 2002 | MONU    |  6000 |
| 2003 | MONA    |  7000 |
| 2004 | Anil    | 54000 |
+------+---------+-------+
10 rows in set (0.00 sec)
```

Ex:- Test class:

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
        public static void main(String[] args) {
                Transaction tx = null;
                try (Session ses = HibernateUtil.getSf().openSession()) {
        String hql="from com.app.model.Employee";
        Query q=ses.createQuery(hql);
        q.setFirstResult(4);
        q.setMaxResults(6);
        List<Employee>empob=q.list();
        empob.forEach(System.out::println);
                } catch (Exception ex) {
                        ex.printStackTrace();
                }
        }
```

```
}
```

output selected rows Id are



Above code can also be written as:

```
List<Employee> eob= ses.createQuery("from " +
Employee.class.getName()).setFirstResult(4).setMaxResult(3).list();
```

Ex#2----------

Select exactly rowId#6 only.

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
		Transaction tx = null;
		try (Session ses = HibernateUtil.getSf().openSession()) {
	String hql="from com.app.model.Employee ";
	Query q=ses.createQuery(hql);
	q.setFirstResult(6);
	q.setMaxResults(1);
	List<Employee>emps=q.list();
	emps.forEach(System.out::println);
		} catch (Exception ex) {
			ex.printStackTrace();
		}
	}
}
```

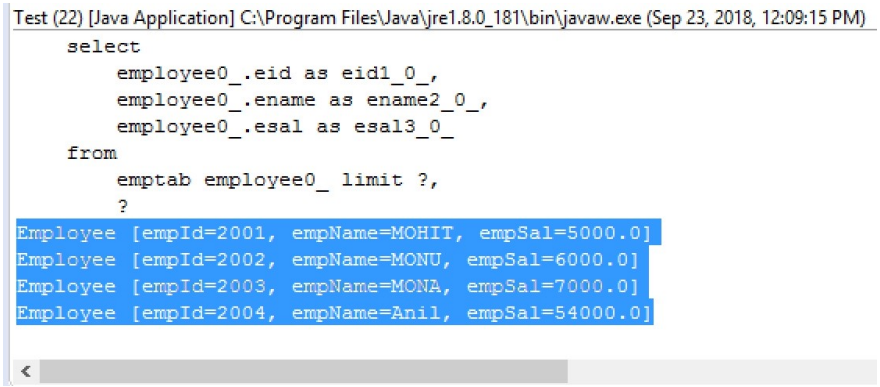
Output (Exactly 6 nos record show)

```
Sep 23, 2018 12:04:35 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiat
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        employee0_.eid as eid1_0_,
        employee0_.ename as ename2_0_,
        employee0_.esal as esal3_0_
    from
        emptab employee0_ limit ?,
        ?
Employee [empId=2001, empName=MOHIT, empSal=5000.0]
```

Ex#3 ==========

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
		Transaction tx = null;
		try (Session ses = HibernateUtil.getSf().openSession()) {
	String hql="from com.app.model.Employee ";
	Query q=ses.createQuery(hql);
	q.setFirstResult(6);
	q.setMaxResults(6);
	List<Employee>emps=q.list();
	emps.forEach(System.out::println);
		} catch (Exception ex) {
			ex.printStackTrace();
		}
	}
}
```

Output

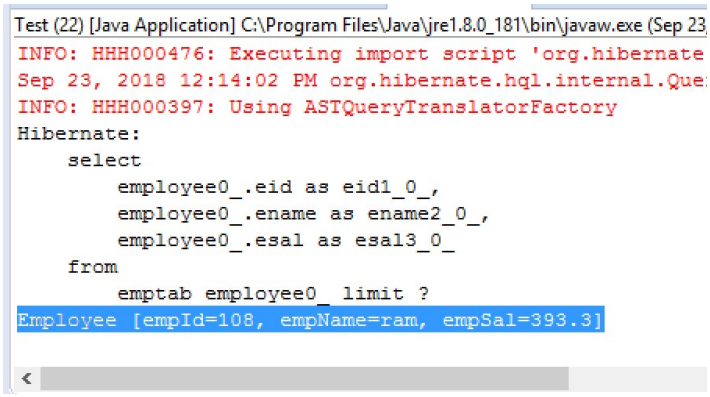


```
Test (22) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 23, 2018, 12:09:15 PM)
    select
        employee0_.eid as eid1_0_,
        employee0_.ename as ename2_0_,
        employee0_.esal as esal3_0_
    from
        emptab employee0_ limit ?,
        ?
Employee [empId=2001, empName=MOHIT, empSal=5000.0]
Employee [empId=2002, empName=MONU, empSal=6000.0]
Employee [empId=2003, empName=MONA, empSal=7000.0]
Employee [empId=2004, empName=Anil, empSal=54000.0]
```

Ex#4 ==My select query returns multiple rows, but choose always first row in result and print that.

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
		Transaction tx = null;
		try (Session ses = HibernateUtil.getSf().openSession()) {
	String hql="from com.app.model.Employee ";
	Query q=ses.createQuery(hql);
	q.setFirstResult(0);
	q.setMaxResults(1);
	List<Employee>emps=q.list();
	emps.forEach(System.out::println);
		} catch (Exception ex) {
			ex.printStackTrace();
		}
	}
}
```

OutPut:-



HQL NON=SELECT OPERATION:- HQL supports non-select operations like

a. Update multiple rows
b. Delete mulitiple rows copy rows from one table to another table (backup data)
   - Use method executeUpdate():int return no.of rows effected
   - It supports both positional and named parameters.

Ex#- update HQL=====

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
		Transaction tx = null;
		try (Session ses = HibernateUtil.getSf().openSession()) {

			tx=ses.beginTransaction(); Employee emp=new Employee();
String hql="update com.app.model.Employee set empName=:a,empSal=:b where empId=:c";
	Query q=ses.createQuery(hql);
	q.setParameter("a", "ramveer");
	q.setParameter("b", 345.3);
	q.setParameter("c", 1001);
		intcount=q.executeUpdate();
			tx.commit();
			System.out.println(count);
			} catch (Exception ex) {
			ex.printStackTrace();
		}
	}
}
```

OutPut:-

```
Test (22) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 23, 2018, 12:20:10 PM)
Sep 23, 2018 12:20:12 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    update
        emptab
    set
        ename=?,
        esal=?
    where
        eid=?
1
```

===Ex#2 delte HQL====

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
		Transaction tx = null;
		try (Session ses = HibernateUtil.getSf().openSession()) {

			tx=ses.beginTransaction(); Employee emp=new Employee();
	String hql="delete com.app.model.Employee where empId=:a";
	Query q=ses.createQuery(hql);
	q.setParameter("a", 1001);
	intcount= q.executeUpdate();
	tx.commit();
	System.out.println(count);
			} catch (Exception ex) {
			ex.printStackTrace();
		}
	}
}
```

SQL JOINS:-To get data from multiple table using single select query use Joins concept in Database.

Example:- Employee details are stored in different tables like:emptab,addrtabk,project and paymentstab etc..

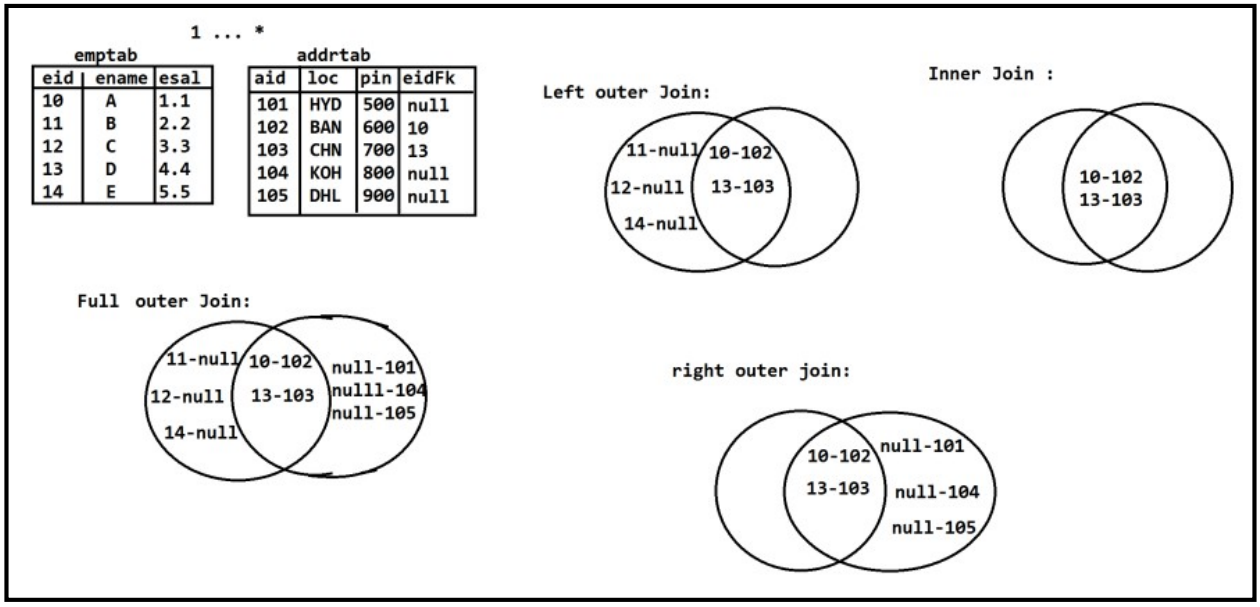
To generate one full and final report using all above table data use SQL Joins concept.

- Types of SQL Joins:-
  1. Equal join(=)
  2. Cross join(x)

3. Self join(o)
4. Inner join or join
5. Outer joins
   a. Left outer join|left join
   b. Right outer join|right join
   c. Full outer join| full join

** Few databases may not support full and right outer joins.
** here word 'outer' is optional.

Example: with tables and ven digram.



Syntax:

consider two tables Table1 and Table2 which are connected using PK-FK columns then Join query looks like:

Select t1.*,t2.*
From table1 t1
join type
Table2 t2
On t1. PKCol1=t2.FKCol
[where condition];

Here Join type=inner/outer joins and where condition is optional.

EX:- Inner Join Example and output:-

```
Select e.enae,a.loc
From emptab e
  Inner join
     Addrtab a
On e.eid=a.eidFk;
```

Output

ENAME LOC
A BAN

D CHN

Ex#2—Left example and output:

```
Select e.ename,a.pin
From emptab e
 Left outer join
Addrtab a
On e.eid=a.eidFk;
```

Output

ENAME	PIN
A	600
B	NULL
C	NULL
D	700
E	NULL

HQL JOINs:-hibernate supports JOINS concept using HQL programming.

- it supports mainly Inner Joins and Outer Joins.
- To write HQL joins two model classes
  a. Must be connected using HAS-A
  b. Must have any one multiplicity
- HQL Joins Sysntax:
  Select Pobj.varivales ,cobj.variables from **ParentClass** Pobj
  JOIN Type
  On **pobj.childHASAVariable** as **cobj** where **condition**;

Ex:- consider below two model classes (with any multiplicity)

```
class Employee {                    class Address {
int empId;                          int addrId;
String empName                      String  loc;
double empSal;                      long pinCode;
Address addr;
}                                   }
```

```
Ex: HQL JOIN is:
select emp.empName,addr.loc
from com.app.Employee emp
            INNER JOIN
ON emp.addr as addr;
```

Steps 1:- create two model classes with HAS-A relation and any one mulitiplicity

Ex:- Employee --------- <> Address

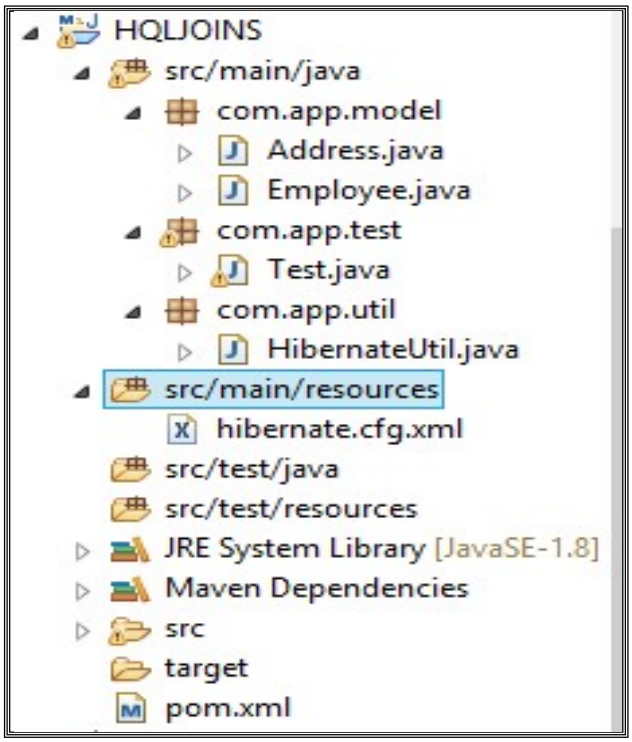
***…**1

Steps 2:- create objects and save into DB suing test class.

Steps 3:- write JOIN HQL and execute

Example:-

1. Folder Structure



2. POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>org.nareshittech</groupId>
	<artifactId>HQLJOINS</artifactId>
	<version>1.0</version>
	<dependencies>
		<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.6</version>
		</dependency>
	</dependencies>
	<build>
```

```
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. Model class

a. Address class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name="addrTab")
public class Address {
	@Id
	@Column(name="aid")
	private int addrId;
	@Column(name="loc")
	private String loc;
	@Column(name="pin")
	private long pinCode;
	public Address() {
		super();
	}
	public int getAddrId() {
		return addrId;
	}
	public void setAddrId(int addrId) {
		this.addrId = addrId;
	}
	public String getLoc() {
		return loc;
	}
	public void setLoc(String loc) {
```

```
		this.loc = loc;
	}
	public long getPinCode() {
		return pinCode;
	}
	public void setPinCode(long pinCode) {
		this.pinCode = pinCode;
	}
	@Override
	public String toString() {
		return "Address [addrId=" + addrId + ", loc=" + loc + ", pinCode=" + pinCode + "]";
	}
}
```

Employee class

```
package com.app.model;
import  java.util.HashSet;
import java.util.Set;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToMany;
import javax.persistence.Table;
@Entity
@Table(name = "emptab")
public class Employee {
	@Id
	@Column(name = "eid")
	private int empId;
	@Column(name = "ename")
	private String empName;
	@Column(name = "esal")
	private double empSal;
	@OneToMany
	@JoinColumn(name = "eidfk")
	private Set<Address> addr = new HashSet<Address>(0);
	public Employee() {
		super();
	}
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
```

```
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	public Set<Address> getAddr() {
		return addr;
	}
	public void setAddr(Set<Address> addr) {
		this.addr = addr;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName + ", empSal="
+ empSal + ", addr=" + addr + "]";
	}
}
```

b. HibernateUtil file

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf = null;
	static {
		sf = new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf() {
		return sf;
	}
}
```

c. Configuration file

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

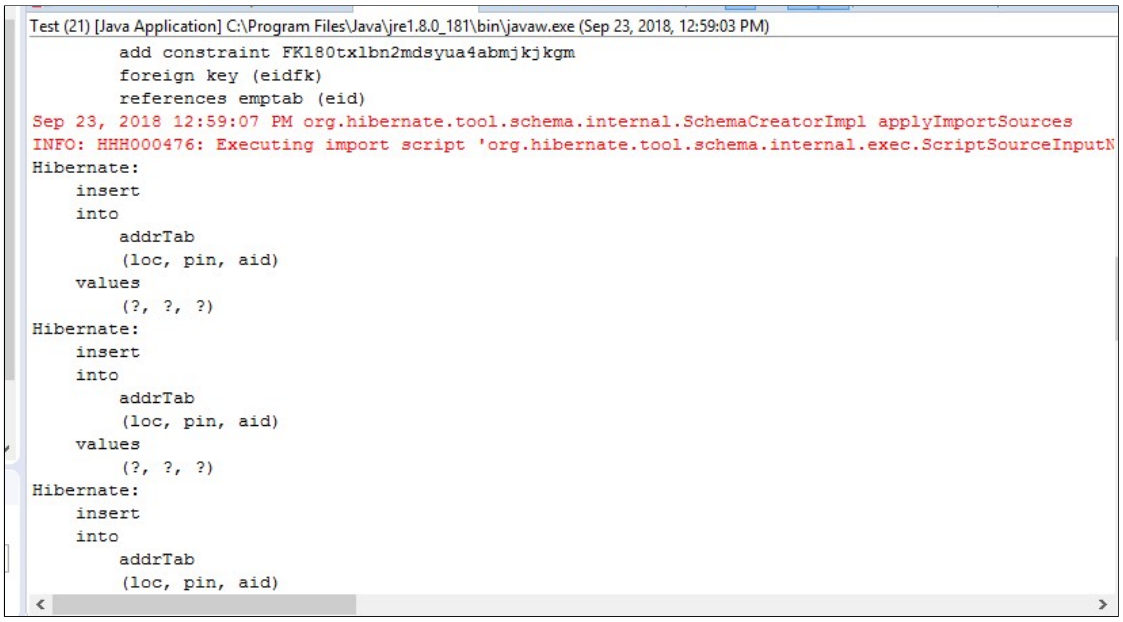
```
<hibernate-configuration>
        <session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
                <property name="hibernate.connection.username">root</property>
                <property name="hibernate.connection.password">system</property>
        <property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
                <property name="hibernate.show_sql">true</property>
                <property name="hibernate.format_sql">true</property>
                <property name="hibernate.hbm2ddl.auto">update</property>
                <mapping class="com.app.model.Address" />
                <mapping class="com.app.model.Employee" />
        </session-factory>
</hibernate-configuration>
```

d. Test class

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
        tx=ses.beginTransaction();
        Address a1=new Address();
                a1.setAddrId(1001);
                a1.setLoc("HYD");
                a1.setPinCode(50001);
                Address a6=new Address();
                a6.setAddrId(1006);
                a6.setLoc("MUM");
                a6.setPinCode(450001);
                Address a2=new Address();
                a2.setAddrId(1002);
                a2.setLoc("BAN");
                a2.setPinCode(40001);
                Address a3=new Address();
                a3.setAddrId(1003);
                a3.setLoc("DLH");
                a3.setPinCode(70001);
                Address a4=new Address();
                a4.setAddrId(1004);
```

```
                a4.setLoc("CHI");
                a4.setPinCode(350001);
                Address a5=new Address();
                a5.setAddrId(1005);
                a5.setLoc("BR");
                a5.setPinCode(40001);
                Employee e1=new Employee();
                e1.setEmpId(2001);
                e1.setEmpName("MOHIT");
                e1.setEmpSal(5000);
                e1.getAddr().add(a1);
                e1.getAddr().add(a3);
                Employee e2=new Employee();
                e2.setEmpId(2002);
                e2.setEmpName("MONU");
                e2.setEmpSal(6000);
                Employee e3=new Employee();
                e3.setEmpId(2003);
                e3.setEmpName("MONA");
                e3.setEmpSal(7000);
                Employee e4=new Employee();
                e4.setEmpId(2004);
                e4.setEmpName("Anil");
                e4.setEmpSal(54000);
                ses.save(a1);
                ses.save(a2);
                ses.save(a3);
                ses.save(a4);
                ses.save(a5);
                ses.save(a6);
                ses.save(e1);
                ses.save(e2);
                ses.save(e3);
                ses.save(e4);
                tx.commit();
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

OutPut:



Code:- Test class:

```
package com.app.test;
import java.util.List;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.util.HibernateUtil;
public class Test {
public static void main(String[] args) {
        try(Session ses=HibernateUtil.getSf().openSession()){
String hql=" select emp.empName, a.loc " + "from com.app.model.Employee emp" + " left join "
+ " emp.addr as a " ;
                Query q=ses.createQuery(hql);
                //q.setParameter(1," A");
                List<Object []>obs=q.list();
                for(Object [] ob:obs)
                {
                        System.out.println(ob[0]+","+ob[1]);
                }
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

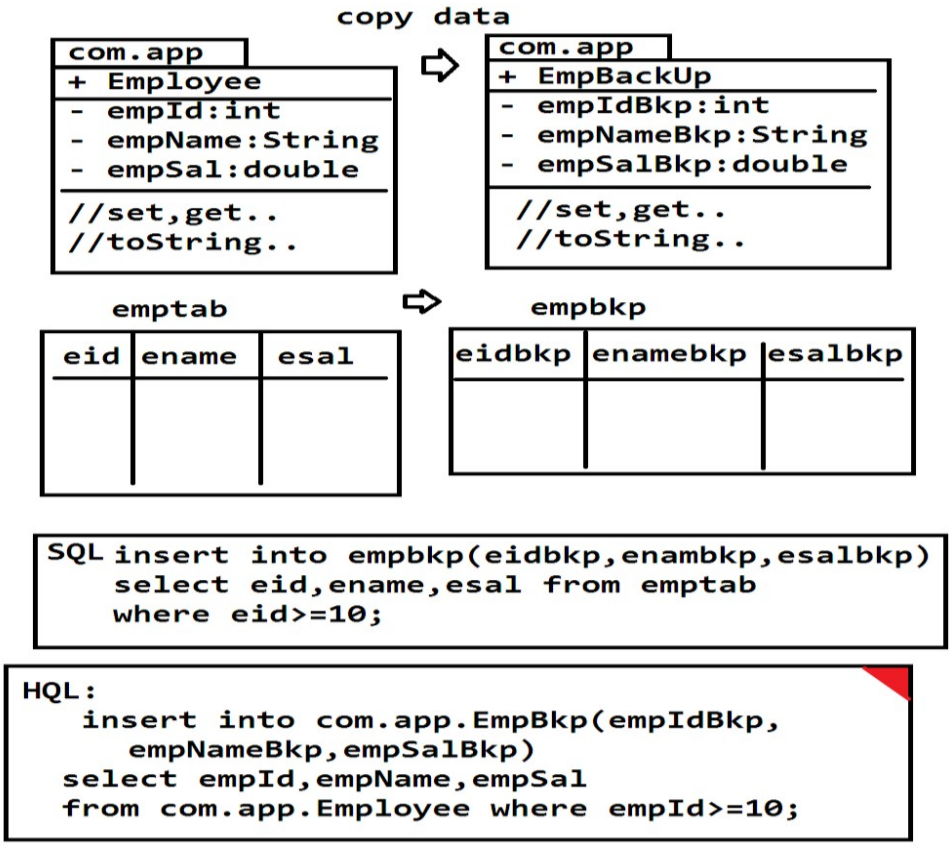
Output

```
Test (21) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 23, 2018, 1:03:51 PM)
Sep 23, 2018 1:03:54 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate:
    select
        employee0_.ename as col_0_0_,
        addr1_.loc as col_1_0_
    from
        emptab employee0_
    left outer join
        addrTab addr1_
            on employee0_.eid=addr1_.eidfk
```

- Use Oracle8iDialect or Oracle10gDialect when you are working on JOINS.

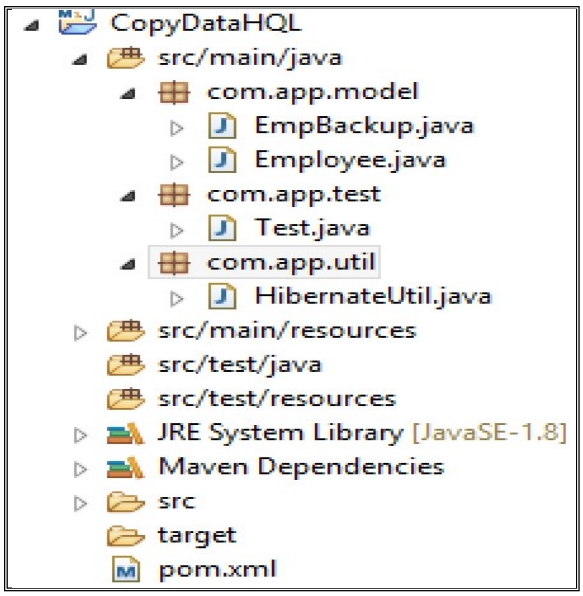
COPY DATA USING HQL:- this concept is used to copy data from source (main) table to backup (support) table (either all rows or few rows).

Consider below example:-



Example:

1. **Folder structure**



2. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>CopyDataHQL</artifactId>
<version>1.0</version>
<dependencies>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.0.5</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
```

```
                        <configuration>
                            <source>1.8</source>
                            <target>1.8</target>
                        </configuration>
                    </plugin>
                </plugins>
            </build>
        </project>
```

3. Model classes(source model class)

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="emptab")
public class Employee {
    @Id
    @Column(name="eid")
    private int empId;
    @Column(name="ename")
    private String empName;
    @Column(name="esal")
    private double empSal;
    public Employee() {
        super();
    }
    public int getEmpId() {
        return empId;
    }
    public void setEmpId(int empId) {
        this.empId = empId;
    }
    public String getEmpName() {
        return empName;
    }
    public void setEmpName(String empName) {
        this.empName = empName;
    }
    public double getEmpSal() {
        return empSal;
    }
    public void setEmpSal(double empSal) {
        this.empSal = empSal;
```

```
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName + ", empSal="
+ empSal + "]";
	}
	}
```

(Destination/backup model class)

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="empbkp")
public class EmpBackup {
	@Id
	@Column(name="eidbkp")
	private int empIdBkp;
	@Column(name="enamebkp")
	private String empNameBkp;
	@Column(name="esalbkp")
	private double empSalBkp;
	public EmpBackup() {
		super();
	}
	public int getEmpIdBkp() {
		return empIdBkp;
	}
	public void setEmpIdBkp(int empIdBkp) {
		this.empIdBkp = empIdBkp;
	}
	public String getEmpNameBkp() {
		return empNameBkp;
	}
	public void setEmpNameBkp(String empNameBkp) {
		this.empNameBkp = empNameBkp;
	}
	public double getEmpSalBkp() {
		return empSalBkp;
	}
	public void setEmpSalBkp(double empSalBkp) {
```

```
		this.empSalBkp = empSalBkp;
	}
	@Override
	public String toString() {
		return "EmpBackup [empIdBkp=" + empIdBkp + ", empNameBkp=" +
  empNameBkp + ", empSalBkp=" + empSalBkp + "]";
	}
}
```

4. HibernateUtil

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
  private static SessionFactory sf=null;
  static
  {
	sf=new Configuration().configure().buildSessionFactory();
  }
  public static SessionFactory getSf()
  {
	return sf;
  }
}
```

5. Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Employee"/>
<mapping class="com.app.model.EmpBackup"/>
</session-factory>
</hibernate-configuration>
```

6. **Test class :**

Steps 1. Insert few records into emptab

```
package com.app.test;

import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
                tx=ses.beginTransaction();
                Employee emp1=new Employee();
                emp1.setEmpId(102);
                emp1.setEmpName("Mohan");
                emp1.setEmpSal(6000);
                Employee emp2=new Employee();
                emp2.setEmpId(103);
                emp2.setEmpName("Ramesh");
                emp2.setEmpSal(7000);
                ses.save(emp1);
                ses.save(emp2);
                tx.commit();
                ses.close();
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output:-

Data base

eid	ename	esal
101	Ramjatan	5000
102	Mohan	6000
103	Ramesh	7000

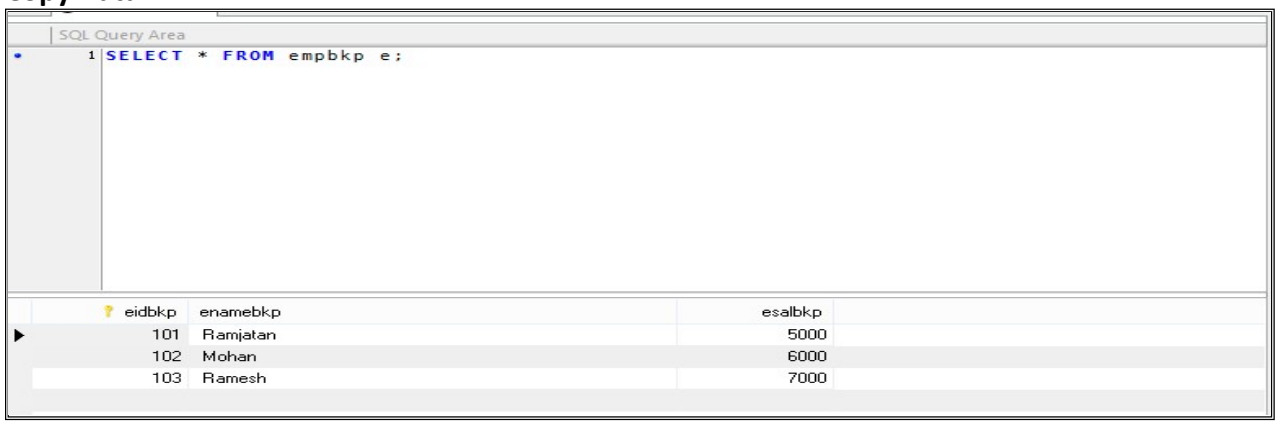
Steps 2. Write below code emptab to copy backuptab

```
package com.app.test;
```

```
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
String hql="insert into com.app.model.EmpBackup(empIdBkp,empNameBkp,empSalBkp)Select
empId,empName,empSal from com.app.model.Employee";
                                Query q=ses.createQuery(hql);
                                intcount =q.executeUpdate();
                                tx.commit();
                                System.out.println("copied:"+count);
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output:-

Copy Data



Sub concepts in Mulitplicity(HAS-A)

Mulitiplicity supported 3 child concepts (all are optional and provided Default typoes also).

1). Direction
2). Fetch Type
3). Casecading

1. Direction:

a. **Uni-Direction:-** if we know parent details, we can get child details using get()/load() methods [select operation]. But using child details we cannot get parent details.

b. **bi-direction:-** Using parent details we can get child details even using child details we can get parent details.

Table for reverse Multiplicity

Direction	**Bi-Direction**
OneToMany	**OneToOne**
ManyToMany	**ManyToMany**
ManyToOne	**OneToMany**
ManyToOne(unique)	**OneToOne**

Ex:- Employee ----------<>Address
1..*

Uni-Directional code:

Class Employee {
@OneToMany
Set<Address>addr;
}

class Address {

Bi-Directional code:

Class Emplolyee {
@OneToMany
}

class Address {
@OneToOne(mappedBy=”addr”)
Employee emp;
}

- Here mappedBy=” ” indicates “bi-direction” concept to Hibernate. It says “it not multiplicity, it is just reverse link”

Ex#2: many-to-one

Product ---------- <>Licence
***…1**

Class Product {
@ManyToOne
Licence lob;
}

class Licence {
@OneToMany(
mappedBy=”lob”)
Set<Product>probs;
}

Ex#3 :- many-to-many ***…***

Student --------- <> Course

Class Student {
@ManytoMany
}

class Course {
@ManyToMany(mappedBuy=”cobs”)
List<Student>sobs;
}

Ex#4:-

Person -------- <>IdCard
1…1

Class Person {
@ManyToOne

```
@JoinColumn(unique=true)
IdCard cob;
}
Class IdCard {
@OneToOne(mappedBy="cob")
Person pob;
}
```

FetchType(Loading type):- FetchType is a enum in hibernate which has possible values

a). EAGER

b). LAZY

EAGER OR EAGER LOADING:- In multiplicity on loading parent data also load child data at same time, this is classed as EAGER loading.

LAZY OR LAZY LOADING:- In multiplicity on loading parent data do not load child data until child object related methods are called.

Multiplicity	**DefaultType**
OneToOne	EAGER
ManyToOne	EAGER
OneToMany	LAZY
ManyToMany	LAZY

- If child in only one Object then default is choosen as EAGER and if child is multiple objects then default is choosen as LAZY by Hibernate.
- To Specify our-own FetchType, code is ….(Fetch=FetchType.<VALUE>)

Ex:- **Employee ------- <> Address(1…*)**

Req: Select all address objects on loading employee object at same time

Code:-

```
Class Address{ }
Class Employee {
        @OneToMany(Fetch=FetchType.EAGER)
        @JoinColumn(name="eidFk")
Set<Address> addr;
}
```

Ex#2:- **Product ----------<> Model (*…1)**

Req:- On selecting product , do not load model at same time.

Code:-

```
Class Model { }
Class product {
        @ManyToOne(fetch=FetchType.LAZY)
Model mob;
}
```

CASCADING:-

Aggrigation:- It creates a loose relation between Parent and child incase of HAS-A (Association).
It means:- On performing any (Non-select) operation over parent , it is not applicable for child. On child again we should write code to do operation.
** By default every HAS-A is Aggregation (I.e no cascading).
Composition:- It creates a strong relation between Parent and child class objects in case of HAS-A (Association).
It means:- On performing any (Non-select) operations same applicable over child objects also. Do not write extra code over child objects.
Note:-

- **Cascading used to convert HAS-A to Composition.**
- **Cascading applicable only for non-select operation [insert,update and delete].**
- **FetchType is applicable for only select (get/load) operation.**

Example code:-

Multiplicity with Directional + FetchType +Cascading :-
Employee---------- <> Address
1…*

----------Parent class code:----------

```
Package com.app.model;
@Entity
@Table(name="emptab")
public class Employee {
@Id
@Column(name="eid")
private int empId;
@Column(name="ename")
private  String empName;
//Employee 1…* Address
@OneToMany (fetch=FetchType.EAGER, cascade=CascadeType.ALL)
@JoinColumn(name="eidFk")
private Set<Address> addr=new HashSet<Address>(0);
//Const, set/get… toString….
}
```

------child class code----------

```
Package com.app.model;
//ctrl+shift+o(imports)
@Entity
@Table(name="addrtab")
public  class Address {
@Id
@Column(name="aid")
private  int addrId;
```

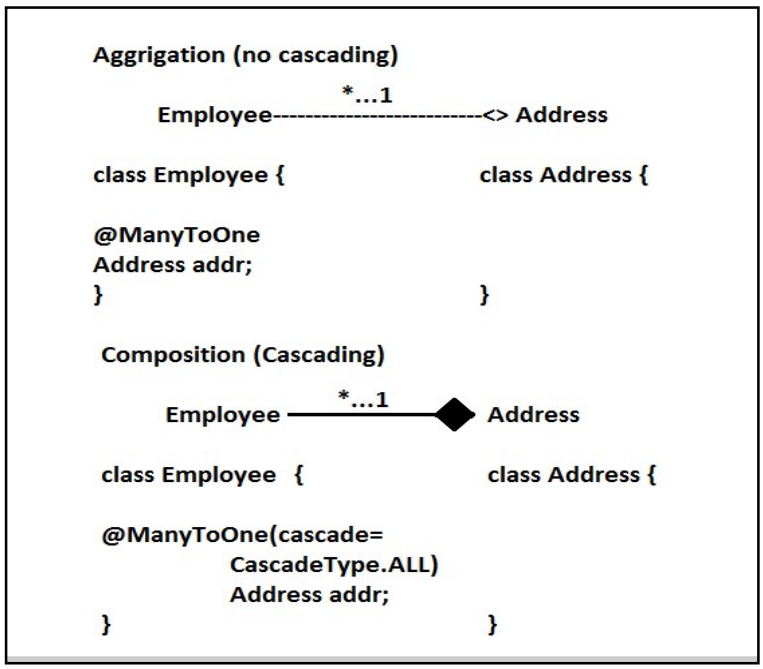
```
@Column (name="loc")
private  String loc;
//bi-directional code (1…1)
@OneToOne(mappedBy="addrs")
//const, set,get…toString…
}
Test class:/// cfg,sf,ses,tx
Address addr1=new Address();
Addr1.setAddrId(101);
Addr1.setLoc("HYD");
Address addr2=new Address();
addr2.setAddrId(102);
addr2.setLoc("DHL");
Address addr3=new Address();
addr3.setAddrId(103);
addr3.setLoc("JRK");
Employee emp=new Empllyee();
emp.setEmpId(10);
emp.setEmpName("SAM");
emp.getAddrs().add(addr1);
emp.getAddrs().add(addr2);
emp.getAddrs().add(addr3);
//cascading applied , so below child obje
///save code not required.
/* ses.save(addr1);
ses.save( addr2);
ses.save(addr3);
*/
ses.save(emp);
Test class #2:// cfg,sf,ses
Employee e=ses.get(Employee.class,10);
```

- **Console:** Query printed emptab
  Left outer join with addrtab (for EAGER FETCH TYPE).

Cascading :- By default every hibernate application performs non-select operation only on parent object, if same needs to be applied on child without writing extra code use "Cascading" concept.

- **If no cascading applied that is Aggrigation if it is applied then it is Composition concept.**
- Applicable only for non-select operation

Example:-

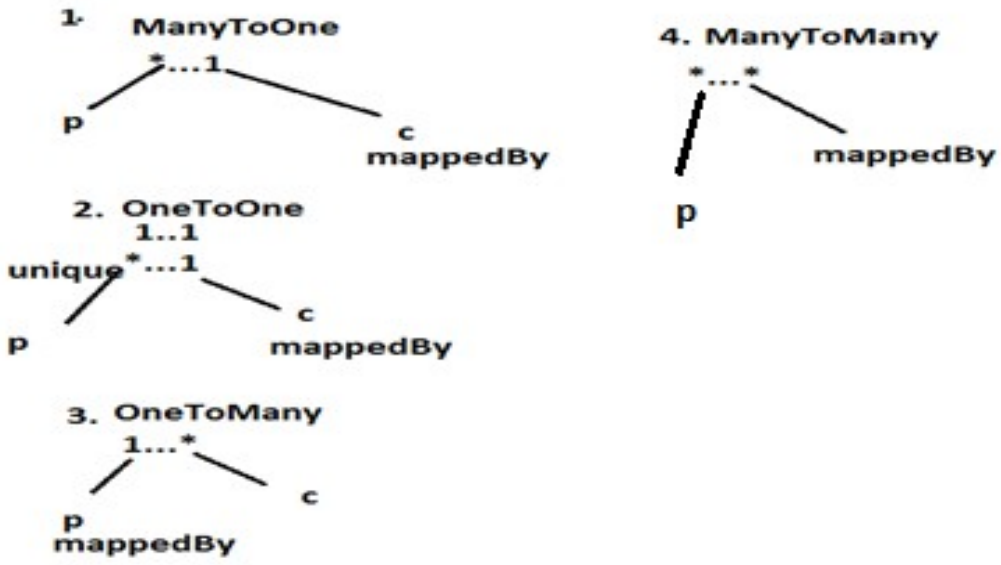


- **Here CascadeType is a enum.**
  Possible values:

ALL:- Apply Composition for all no-select operation.
That is(save, update,delete)
PERSIST:- Apply Composition only for SAVE operations
MERGE: Apply composition only for update operation
REMOVE: Apply composition only for delete operation.
REFRESH: Apply composition only for re-loading data from DB or Any Type Cache (Session/Factory)
DETACH: Apply Composition only for remove object from only cache [not from DB] memory.

ARENT CLASS (DIRECTIONCODE)	**Child class (Bi-Direction code)**
@ManyToOne(fetch=FetchType.EAGER,cascade= **CascadeType.ALL**) **@JoinColumn(name="**---"**,unique**=true)	@OneToOne(mappedBy=" ----- "
@ManyToOne(fetch=FetchType.EAGER,cascade= **CascadeType.ALL**)	@OneToOne(mappedBy=" ----- "
@ManyToMany(fetch=FetchType.EAGER,CASCADE= **CascadeType.ALL mappedBy=" ")**	@ManyToOne
@ManyToMany(fetch=FetchType.EAGER,casede= CascadeType.ALL)	@ManyToMany(mappedBy=" --- "

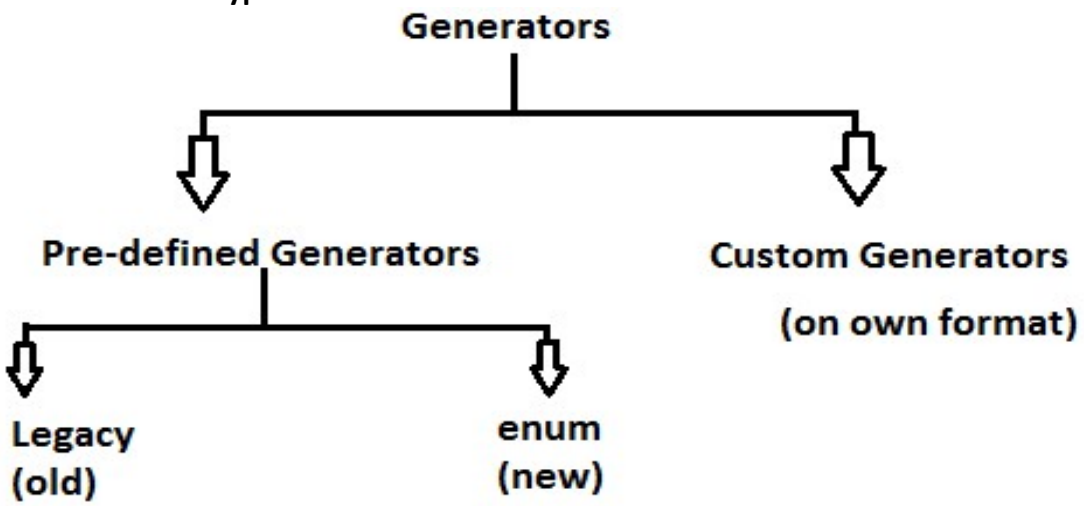
MappedBy:- it indicates multiplicity must be applied on bi-directional using no extra FK column created again one more time.It must be applied to non-owing side that is exact opposite to FKcolumn side. That is



Primary key Generators:-

A Generator is a class defined in Hibernate it is used to generate Primary key value on Performing save(Insert) opration.

Generators are two types in Hibernates:



1. **Pre-Defined Generators:-**

These are class already implementing in hibernate. Categorized into two types.

a. Legacy (also called as old generator)
b. Enum Type (also called as new generator)

2. **Custom Generators:-**

We can defined our own formate primary key generator class. It is known as Custom Generator. These are the mostly used generators in real time.

Ex:- University Registeration number

AAdhar card id

Pan card number

Bank A/C Id

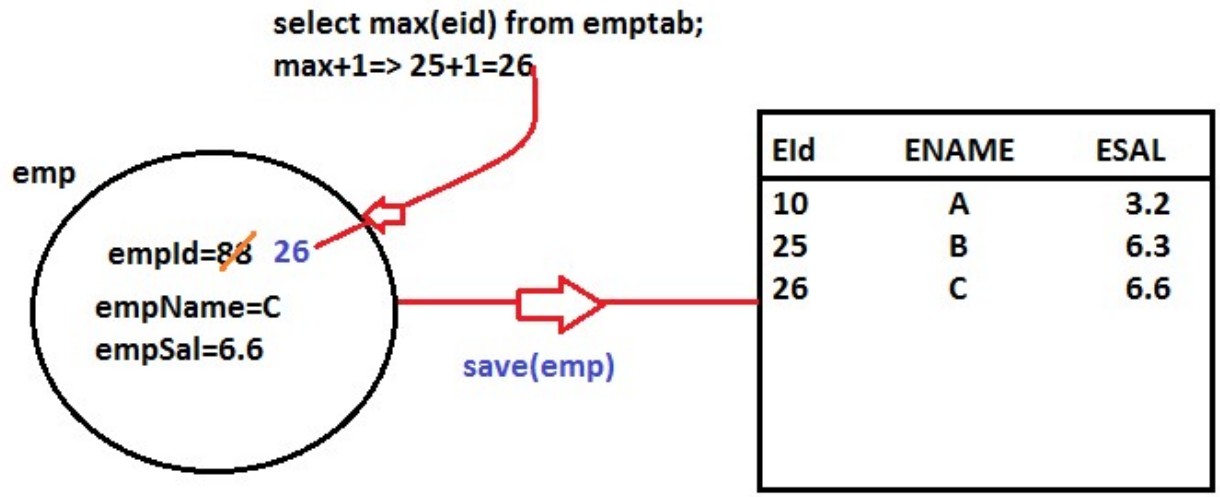
Book my show Tracking id

Mobile EMI No.

Legacy Generators:-

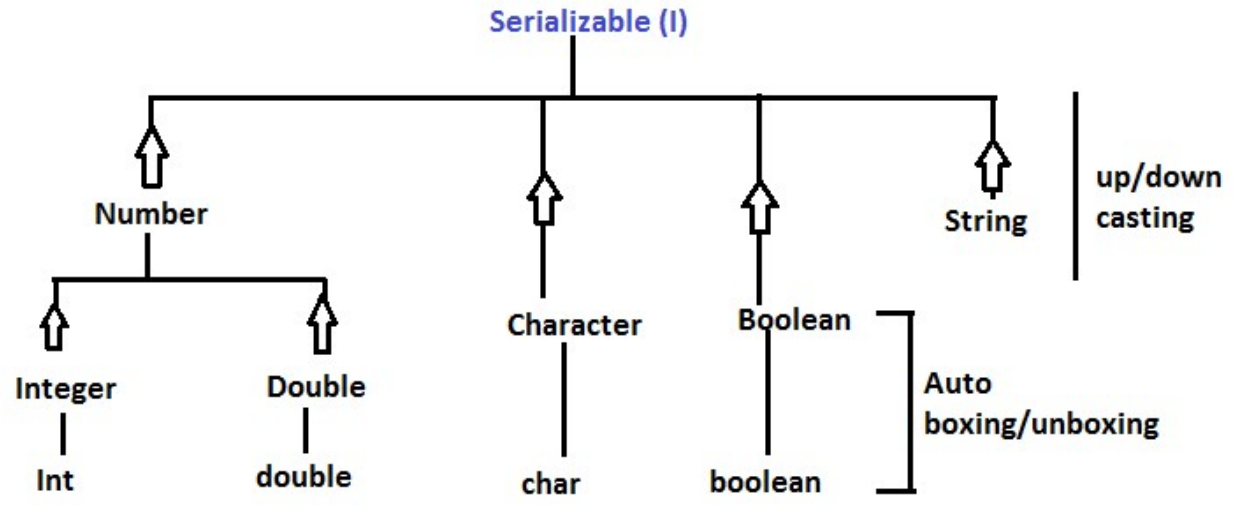
1. **Assigned:-** It is only default generator in hibernate it indicates no value generated by hibernate it means value assign to objects are considered as primary key values.
2. **Increment [max+1] (int-dataType):-**This generator will execute to get the maximum primary key value from database by using select query if no records found max value is 0 (Starts with one) even value Provided by end-user will be overridden with Max+1 value.

Consider below example:-



Note:-

- It is applicable for all databases.
- It is applicable for int datatypes. (byte, short,int,long)
- If table has no rows then max value is=0 and max+1=1
- If primary key value is provide by end-user will be overridden with generated value.
- To read generated primary key value iuse save method return type that is Java.io.Serializable(I)(empty interface-marker interface).
- Every primary key datatype either directly or indirectly should be connected to Serivalizable



- Save method returns autoboxed and upcasted value in serializable format. Programmer should downcast and autounboxed to read primary key value.

Example code:-

```
Test class:- // cfg, sf,ses,tx
Employee emp=new Employee();
//emp.setEmpId(88);==➔ Not required.
emp.setEmpName("C");
emp.setEmpSal(6.6);
Serializable s=ses.save(emp);
//down casting
Integer eid=(Integer)s;
//auto unboxing
Int id=eid;
System.out.println("id");
//one line:- Downcast and AutoUnbox.
Int eid=(Integer)ses.save(emp);
System.out.println(eid);
```

3. **UUID:-** it stands for universal unique identifier. This concept is used to generate Hexadecimal number in String format.
   - It is introduced in JDK1.5 also supported by Hibernate as a primary key generator.
   - It will generate one unique nuber by using below details IP-Address, System Date and Time, Version Details (OS,JDK, Database) and one Random number.
   - In Core-Java (Java SE). it is class defined in Java.util.package.
   - It uses 0,1,……9, A/a, B/b, C/c, D/d, E/e, F/f to generate the number
     - (10) (11) (12) (13) (14) (15)

     (Hexadecimal format)

   - It is supported by every database because number generated at java side (Hibernate) not at database side.
   - This hexadecimal number generated in String formate.**(java.lang.String)**

4. **Native:-** This generator is also called as auto increment supported by databases like MySQL , SQL-Sever etc…. not supported by oracle database.
    - This is database specifiec generator, not executed at java side so all databases may not support this.
    - It is also behaves like “max+1” but executed at database side.
    - Databaseshould support auto-increment concept.

5. **SEQUENCE GENERATOR:-

This is mostly used generator in Applications. Works mainly in Oracle Database.

- Sequence is auto-number generation system(concept).
- Every sequence needs two inputs like starting number and step value
- Hibernate f/w creates one sequence (if we use hbm2ddl.auto=create/update) with name:HIBERNATE_SEQUENCE.
- It is a database side gernerator. So, may not work in all database.
- It generates int type value.

==========CREATING SQUENCE==========

SQL Syntax:-

Create sequence <sequence-name>
Start with <int-value>
Increment by <int-value>;

Ex:-

Create sequence SAMPLE
Start with 10
Increment by 1;

==========CALL/EXECUTE SEQUENCE==========

SQL Syntax:

Select <sequence>.nextval from dual;

- here nextval will get value in sequence it is a in-built function in oracle works only for sequence concept.

Ex:-

Select SAMPLE.nextval from dual;

Here values generated by above sequence
Is: 10 ,11,12,13,14,15…….(on evey call)

6. hi_lo (High-Low) Generator:-

it follows HILO Algorithm format to generate numbers.
It is starts from
First row =2 pow 0=1
2nd row = 2 pow 15 =32768
3rd row =onward=last value+2 pow 15

HIBERNATE Generator Example:-==========

To implement generators concept in Hibernate Application

Steps 1:- use below annotation with enum in model class, at @Id level.

Annotations:-

1. **@GeneratedValue**
2. **@GenericGenerator (Old, custom)**

Enum:-

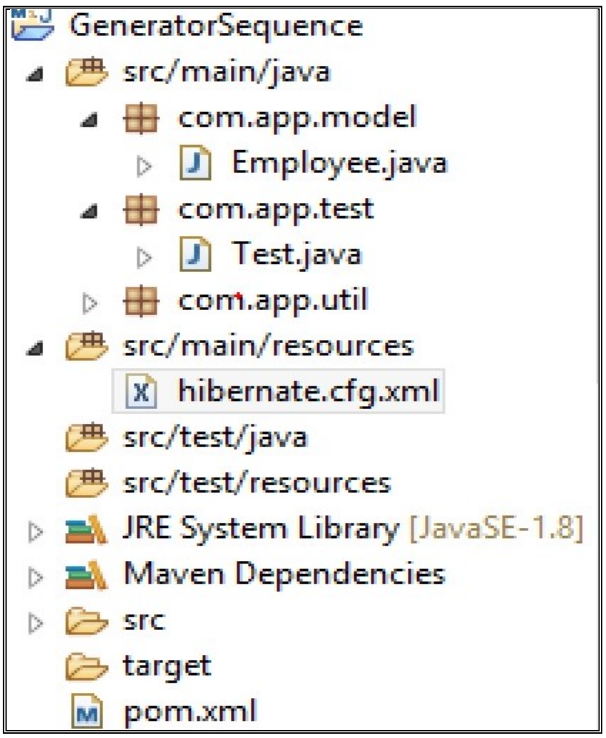
1. **GenerationType:-(AUTO, SEQUENCE,IDENTITY,TABLE)**

Steps 2:- in test class while createing object t model class not required to provide primary key value, if provided it will be ignored by Generator.

Steps 3:- call ses.save(obj) method that returns Generated value as "Serializable" (java.io) format. Here Hibernate "auto box and upcast the value and returns as Serilizable ", programmer should down cast an auto unbox.

Ex:=---------- using SQUENCE GENERATOR----------

1. **Folder structure**



2. **Pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>GeneratorSequence</artifactId>
<version>1.0</version>
<dependencies>
        <dependency>
            <groupId>mysql</groupId>
```

```
                <artifactId>mysql-connector-java</artifactId>
                <version>5.0.5</version>
            </dependency>
            <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>5.1.11.Final</version>
            </dependency>
        </dependencies>
        <build>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.0</version>
                    <configuration>
                        <source>1.8</source>
                        <target>1.8</target>
                    </configuration>
                </plugin>
            </plugins>
        </build>
</project>
```

3. Model class:-

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import  javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
@Entity
@Table(name="emptab1")
public class Employee {
@Id
@GeneratedValue(generator="myemp", strategy=GenerationType.SEQUENCE)
@SequenceGenerator(name="myemp", sequenceName="test_seq")
@Column(name="eid")
private int empId;
@Column(name="ename")
private String empName;
@Column(name="esal")
private double empSal;
```

```
public Employee() {
	super();
}
public int getEmpId() {
	return empId;
}
public void setEmpId(int empId) {
	this.empId = empId;
}
public String getEmpName() {
	return empName;
}
public void setEmpName(String empName) {
	this.empName = empName;
}
public double getEmpSal() {
	return empSal;
}
public void setEmpSal(double empSal) {
	this.empSal = empSal;
}
@Override
public String toString() {
	return "Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + "]";
}
}
```

4. **HibernateUtil**

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf=null;
	static
	{
		sf=new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf()
	{
		return sf;
	}
}
```

5. **Cfg.xml:-**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/test</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Employee"/>
<!-- <mapping class="com.app.model.Address"/> -->
</session-factory>
</hibernate-configuration>
```

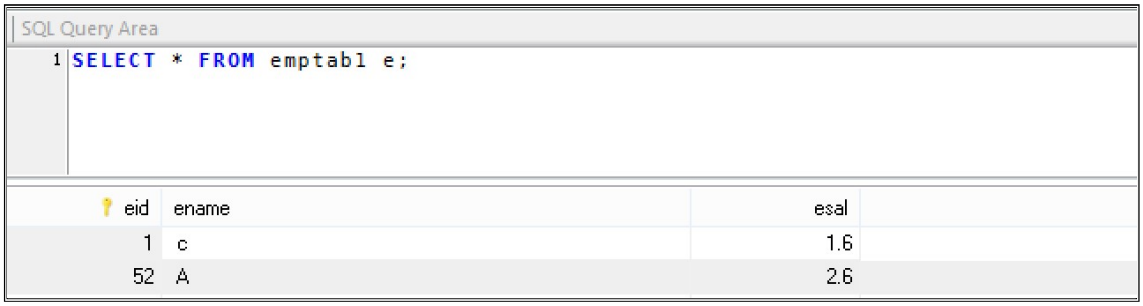
6. Test class:-

```
package com.app.test;
import java.io.Serializable;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
        public static void main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
                tx=ses.beginTransaction();
                Employee e=new Employee();
                e.setEmpName("A");
                e.setEmpSal(2.6);
                Serializable s=ses.save(e);
                //downcat and autounbox
                Integer eid=(Integer)s;
                int empId=eid;
                System.out.println("employee created with id:"+empId);
                tx.commit();
                //ses.close();
                }catch(Exception ex)
        {
                ex.printStackTrace();
        }
```

}}

Output

DataBase Record



- To use legacy generators code should have @GenericGenerator
  ===Example===
  public class Employee {
  @id
  @GeneratedValue(generator="abc")
  @GenericGenerator(name="abc",strategy="uuid")
  @Column(name="eid")
  private String empId;
  ……..
  …..
  }
- We can use uuid, native, increment,……

CUSTOM GENERATORS IN HIBERNATE:-

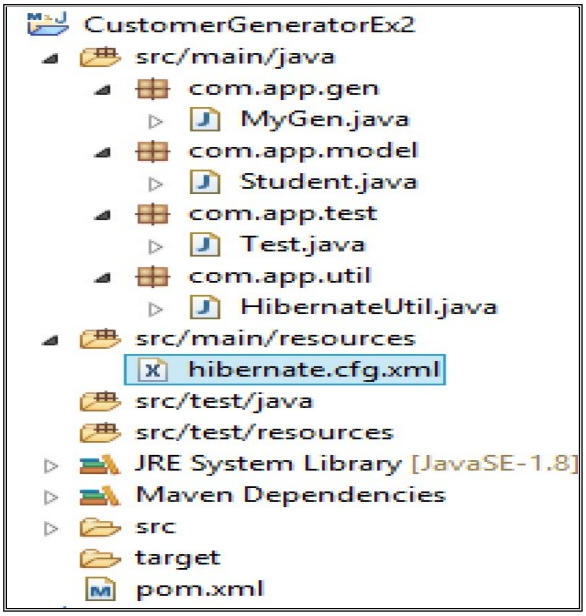
To specify our own format for PrimaryKey use custom Generators concept.
Ex:- Student ID: SAT-85695
Employee ID: EMP-5754
PAN CARD ID: DYBPM1887k
All are used as Primary key and they are implemented using Custom Generators

- Steps to implement custom Gneratores

1. Create one new public class with any name and any package.
2. Implement above class with interface IdentifierGenerator(org.hibernate.id)
3. Override methnod generate() which returns PrimaryKey value as java.io.Serializable
4. In model class use @GenericGenerator and provide strategy as your full class name.
5. Finally in test class , create model class object and save.

=====Example===============

1. **Folder structure**



2. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>CustomerGeneratorEx2</artifactId>
<version>1.0</version>
<dependencies>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.0.5</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
```

```
                                <source>1.8</source>
                                <target>1.8</target>
                            </configuration>
                        </plugin>
                    </plugins>
                </build>
</project>
```

3. **Generator class**

```
package com.app.gen;
import java.io.Serializable;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;
import org.hibernate.HibernateException;
import org.hibernate.engine.spi.SessionImplementor;
import org.hibernate.id.IdentifierGenerator;
publicclass MyGen implements IdentifierGenerator {
        @Override
        public Serializable generate(SessionImplementor session, Object object) throws
HibernateException {
                //String Prefix1="ST";
                //String Prefix2="HIB";
                 String date =new SimpleDateFormat("yyyy-mm-dd").format(new Date());
                intnum=new Random().nextInt(1000);


                String Prifix1 = "ST";
                String Prifix2 = "HIB";
                returnPrifix1+date+Prifix2+"-"+num ;

        }
        }
```

4. **Define Model class**

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;
import org.hibernate.annotations.GenericGenerator;
@Entity
@Table(name="StudentTab1")
publicclass Student {
```

```
	@Id
	@Column(name="sid")
	@GeneratedValue(generator="abc")
	@GenericGenerator(name="abc",strategy="com.app.gen.MyGen")
	private String StdId;
	@Column(name="sname")
	private String StdName;
	@Column(name="sfee")
	privatedoubleStdFee;
	public Student() {
		super();
	}
	public String getStdId() {
		returnStdId;
	}
	publicvoid setStdId(String stdId) {
		StdId = stdId;
	}
	public String getStdName() {
		returnStdName;
	}
	publicvoid setStdName(String stdName) {
		StdName = stdName;
	}
	publicdouble getStdFee() {
		returnStdFee;
	}
	publicvoid setStdFee(doublestdFee) {
		StdFee = stdFee;
	}
	@Override
	public String toString() {
		return"Student [StdId=" + StdId + ", StdName=" + StdName + ", StdFee=" + StdFee
+ "]";
	}
}
```

5. HibernateUtil

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf=null;
	static
```

```
        {
                sf=new Configuration().configure().buildSessionFactory();
        }
        public static SessionFactory getSf()
        {
                return sf;
        }
}
```

6. Hibernate.cfg.xml

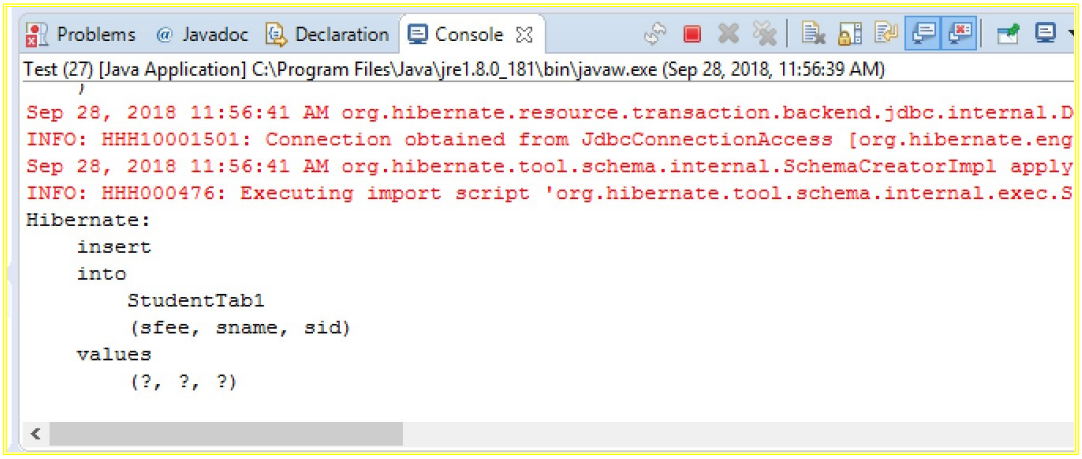
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.model.Student"/>
<!-- <mapping class="com.app.model.Address"/> -->
</session-factory>
</hibernate-configuration>
```

1. Test class ://cfg,sf,ses,tx

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Student;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
                Transaction tx=null;
                try(Session ses=HibernateUtil.getSf().openSession())
                {
                        tx=ses.beginTransaction();
                        Student std=new Student();
                        std.setStdName("mohan");
                        std.setStdFee(23.2);
                        ses.save(std);
```

```
                tx.commit();
                //ses.close();
            }catch(Exception ex)
            {
                ex.printStackTrace();
            }
    }
        }
```

Output



- Define one Generator, it should generate Primarykey in below formate

FORMAT:- USN-<4DIGITrANDOMMnUMBER>-
<YearMonthDayHourMinSec>-SR-<4Digits HexaDecimalNumberRandom>

Ex:- USN-8756-20180830185725-SR-1A2D

❖ **COMPOSITE PRIMARY KEY:-**

Combination of 2 or more column in table behaves as Primary key Type. It is known as Composite PrimaryKey.

- Simple PrimaryKey (or Primary Key) indicate only one columnbehaves as primary key.

***** Hibernate Model class should have Simple

COMPOSITE PK

eid	ename	esal	dsg
101	A		
101	B		
102	A		
NULL	A		
101	NULL		
101	A		

Steps : **To implement Composite PrimaryKey:-**

1. Define one new class used as Primary key data Type, it must implement java.io.Serializable

2. Move (define) all veriables in above class which are involved in composite Primary key creation.
3. On top of this class add @Embeddable annotation.
4. Make HAS-A relation between model

Class and DataType class

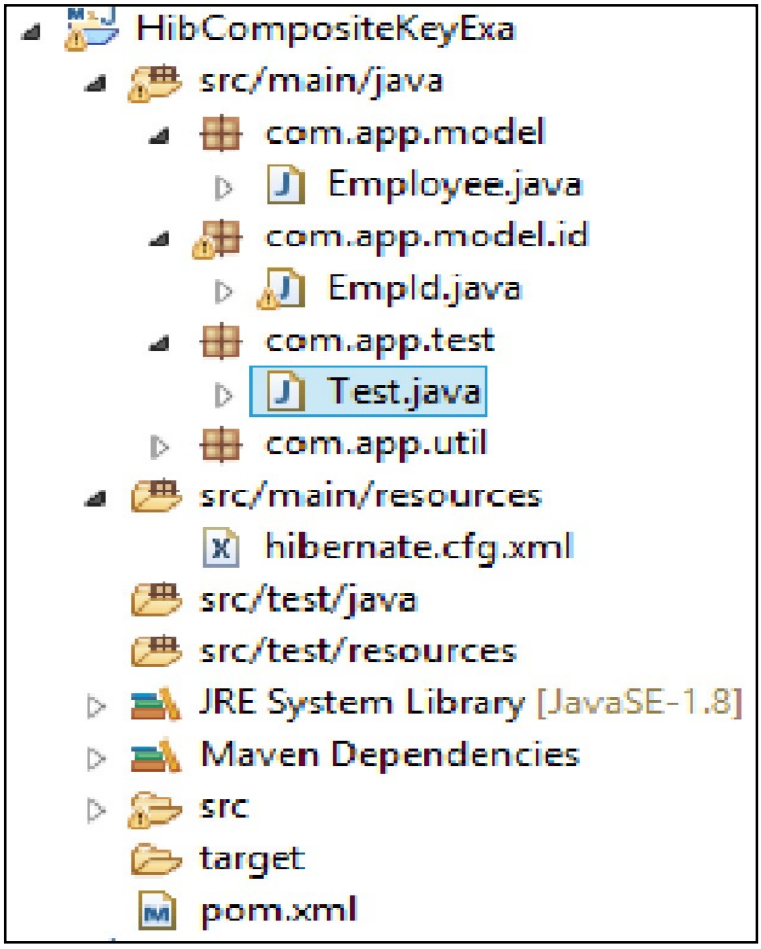
5. Apply @EmbeddedId Annnotation over HAS-A relation.

*** use hbm2ddl.auto=create

6. Write test class and create object and save to Test.

==========code==========

1. Folder structure



2. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
```

```
<artifactId>HibCompositeKeyExa</artifactId>
<version>1.0</version>
<dependencies>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.0.5</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. EmpId primary key class

```
package com.app.model.id;
import  java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Embeddable;
@Embeddable
public class EmpId implements Serializable {
@Column(name="eid")
private int empId;
@Column(name="ename")
private String empName;
public EmpId() {
	super();
}
public int getEmpId() {
	return empId;
```

```
}
public void setEmpId(int empId) {
	this.empId = empId;
}
public String getEmpName() {
	return empName;
}
public void setEmpName(String empName) {
	this.empName = empName;
}
@Override
public String toString() {
	return "EmpId [empId=" + empId + ", empName=" + empName + "]";
}
}
```

4. Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.EmbeddedId;
import javax.persistence.Entity;
import  javax.persistence.Table;
import com.app.model.id.EmpId;
@Entity
@Table(name="emptab2")
public class Employee {
@EmbeddedId
private EmpId eidpk;
@Column(name="esal")
private double empSal;
public Employee() {
	super();
}
public EmpId getEidpk() {
	return eidpk;
}
public void setEidpk(EmpId eidpk) {
	this.eidpk = eidpk;
}
public double getEmpSal() {
	return empSal;
}
```

```
public void setEmpSal(double empSal) {
	this.empSal = empSal;
}
@Override
public String toString() {
	return "Employee [eidpk=" + eidpk + ", empSal=" + empSal + "]";
}
}
```

5. HibernateUtil

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf=null;
	static
	{
		sf=new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf()
	{
		return sf;
	}
}
```

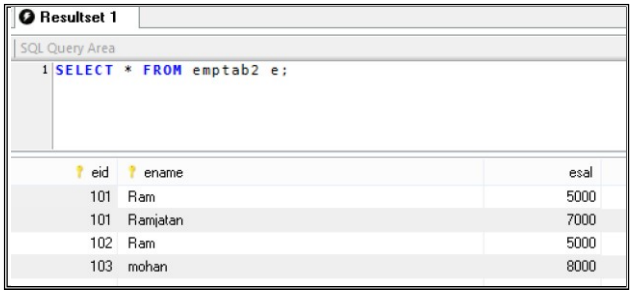
6. Hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.model.Employee"/>
<mapping class="com.app.model.id.EmpId"/>
</session-factory>
</hibernate-configuration>
```

7. Test class

```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.model.id.EmpId;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
	Transaction tx=null;
	try(Session ses=HibernateUtil.getSf().openSession()){
		tx=ses.beginTransaction();
		//Primary key value object
		EmpId eid=new EmpId();
		eid.setEmpId(101);
		eid.setEmpName("Ram");
		//model class object
		Employee emp=new Employee();
		emp.setEidpk(eid);
		emp.setEmpSal(5000);
		ses.save(emp);
		tx.commit();
		//ses.close();
	}catch(Exception ex)
	{
		ex.printStackTrace();
	}
}
}
```

output



BAG AND IDBAG:-

These are special collections given in Hibernate. Both are internally uses List Type only.

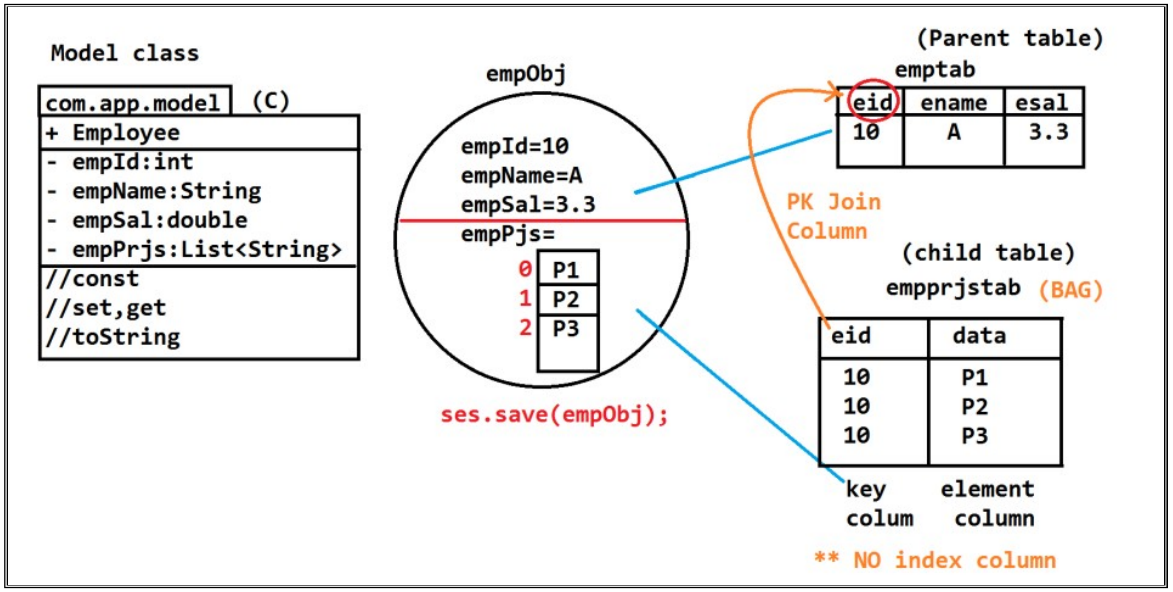
- For list Collection, hibernate creates a child table with 3 columns those are: Key column, index column, and element column,

- For List index number provided by JVM only. Starts from zero (0,1,2….)

BAG:- (List-index column)----

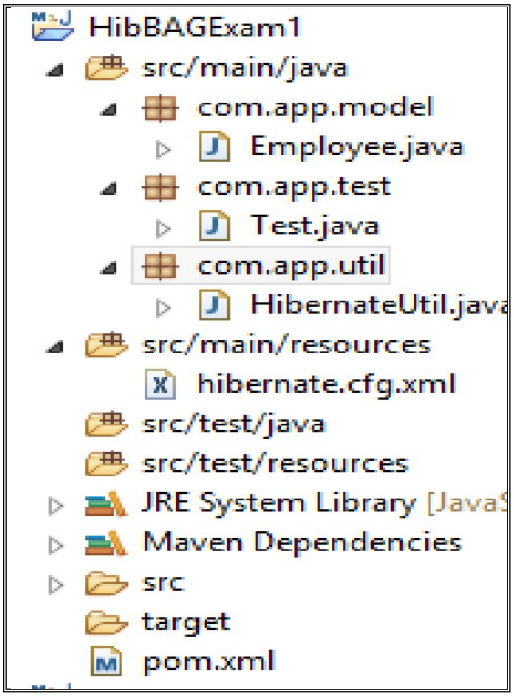
Bag collection is used to store duplicate values without index numbers created by hibernate only.

BAG=List-index Column



Example:----------

1. **Folder Sturcture**



========Code============

2. **Pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>HibBAGExam1</artifactId>
<version>1.0</version>
<dependencies>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.0.5</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. Model class

```
package com.app.model;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
import javax.persistence.JoinColumn;
import javax.persistence.Table;
@Entity
@Table(name="emptab")
public class Employee {
@Id
@Column(name="eid")
private int empId;
@Column(name="ename")
private String empName;
@Column(name="esal")
private double empSal;
@ElementCollection
@CollectionTable(name="empprjstab",joinColumns=@JoinColumn(name="eid"))
@Column(name="data")//element clumn
private List<String> empPrjs=new ArrayList<String>();
public Employee() {
	super();
}
public int getEmpId() {
	return empId;
}
public void setEmpId(int empId) {
	this.empId = empId;
}
public String getEmpName() {
	return empName;
}
public void setEmpName(String empName) {
	this.empName = empName;
}
public double getEmpSal() {
	return empSal;
}
public void setEmpSal(double empSal) {
	this.empSal = empSal;
}
public List<String> getEmpPrjs() {
	return empPrjs;
}
public void setEmpPrjs(List<String> empPrjs) {
	this.empPrjs = empPrjs;
}
@Override
```

```
public String toString() {
	return "Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", empPrjs=" + empPrjs + "]";
}
}
```

4. **HibernateUtil**

```
package com.app.util;

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

public class HibernateUtil {

	private static SessionFactory sf=null;
	static
	{
		sf=new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf()
	{
		return sf;
	}
}
```

5. **Hibernate.cfg.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.model.Employee"/>
<!-- <mapping class="com.app.model.id.EmpId"/> -->
</session-factory>
</hibernate-configuration>
```

6. **Test class:**

```
package com.app.test;
```

```
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) {
	Transaction tx=null;
	try(Session ses=HibernateUtil.getSf().openSession()){
		tx=ses.beginTransaction();
		Employee e=new Employee();
		e.setEmpId(10);
		e.setEmpName("A");
		e.setEmpSal(3.3);
		e.getEmpPrjs().add("p1");
		e.getEmpPrjs().add("p2");
		e.getEmpPrjs().add("p3");
		ses.save(e);
		tx.commit();
		ses.close();
		System.out.println("done");
	}catch(Exception ex)
	{
		ex.printStackTrace();
}}}
```

Output:

```
Hibernate:

    alter table empprjstab
        add constraint FK53lxcmvnof75tg8fpjpxu2qjq
        foreign key (eid)
        references emptab (eid)
Sep 28, 2018 11:19:43 PM org.hibernate.tool.schema.internal.SchemaCreatorImpl applyIm
INFO: HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.Scr
Hibernate:
    insert
    into
        emptab
        (ename, esal, eid)
    values
        (?, ?, ?)
Hibernate:
    insert
    into
        empprjstab
        (eid, data)
    values
        (?, ?)
```

Emptab record

Resultset 1

SQL Query Area

```
1 SELECT * FROM emptab e;
```

eid	ename	esal
10	A	3.3

SQL Query Area

```
1 SELECT * FROM empprjstab e;
```

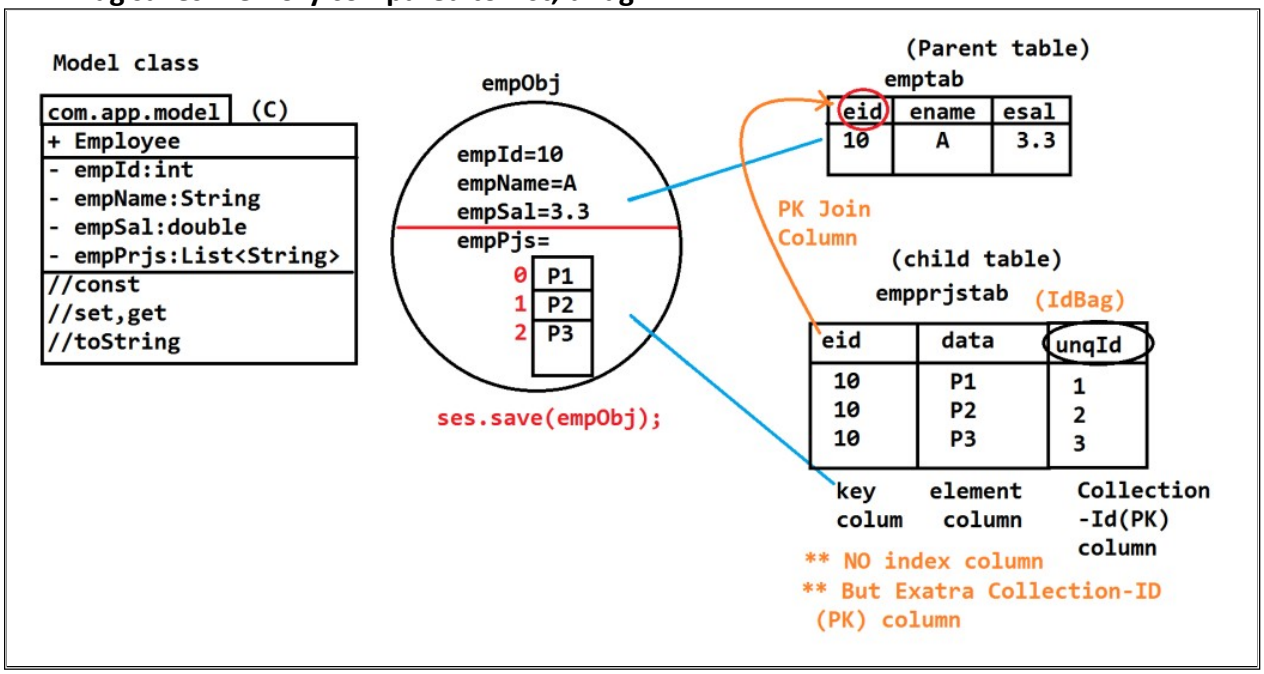
eid	data
10	p1
10	p2
10	p3

ID-BAG:-

It is a special collection provided by hibernate, internally uses List type only.

ID-BAG=List-index column+Collection-Id column.

- **Id-Bags are faster compared to List, Bag in data retrival.**
- **Bag saves memory compared to List,IdBag**

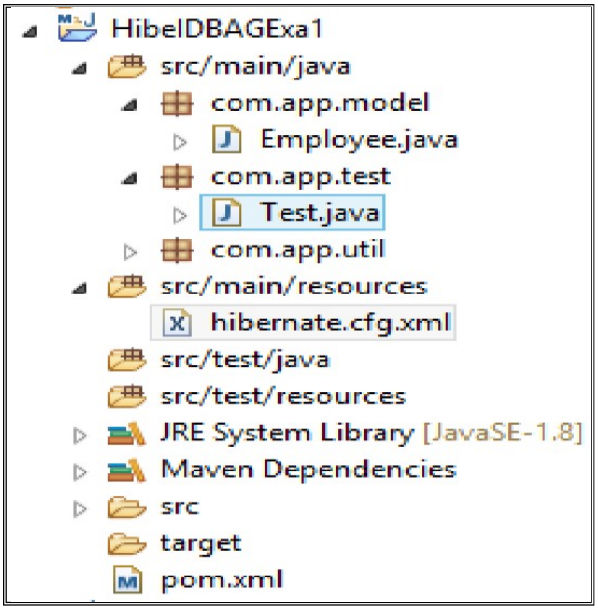


- for this extra column we must specify 3 details columnName,generator,datatype using @CollectionId Annotation.

For generator use @GeneriGeneratior.

==========code==========

1. **folder structure**



2. **same as before pom.xml file**
3. **Model class**

```
package com.app.model;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Table;
import org.hibernate.annotations.CollectionId;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;
@Entity
@Table(name="emptab")
@GenericGenerator(name="mygen",strategy="increment")
public  class Employee {
@Id
@Column(name="eid")
private int empId;
@Column(name="ename")
private String empName;
@Column(name="esal")
private double empSal;
@ElementCollection
@CollectionTable(name="empprjstab",joinColumns=@JoinColumn(name="eid"))
```

```
@CollectionId(columns=@Column(name="unqId"),generator="mygen",type=@Type(type="long
")
)
@Column(name="data")
private List<String>empPrjs=new ArrayList<String>();
public Employee() {
	super();
}
public int getEmpId() {
	return empId;
}
public void setEmpId(int empId) {
	this.empId = empId;
}
public String getEmpName() {
	return empName;
}
public void setEmpName(String empName) {
	this.empName = empName;
}
public double getEmpSal() {
	return empSal;
}
public void setEmpSal(double empSal) {
	this.empSal = empSal;
}
public List<String> getEmpPrjs() {
	return empPrjs;
}
public void setEmpPrjs(List<String> empPrjs) {
	this.empPrjs = empPrjs;
}
@Override
public String toString() {
	return "Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + ", empPrjs=" + empPrjs + "]";
}
}
```

4. **Same as before HibernateUtil program**

5. **Hibernate.cfg.xml**

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

```
"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">create</property>
<mapping class="com.app.model.Employee"/>
<!-- <mapping class="com.app.model.id.EmpId"/> -->
</session-factory>
</hibernate-configuration>
```

6. **Test class**

```
package  com.app.test;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import  com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
        public static void main(String[] args) {
        Transaction tx=null;
        try(Session  ses=HibernateUtil.getSf().openSession()){
                tx=ses.beginTransaction();
                Employee e=new Employee();
                e.setEmpId(11);
                e.setEmpName("A");
                e.setEmpSal(3.3);
                e.getEmpPrjs().add("p1");
                e.getEmpPrjs().add("p2");
                e.getEmpPrjs().add("p3");
                ses.save(e);
                tx.commit();
                ses.close();
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output

```
Hibernate:

    alter table empprjstab
        add constraint FK531xcmvnof75tg8fpjpxu2qjq
        foreign key (eid)
        references emptab (eid)
Sep 29, 2018 12:01:30 AM org.hibernate.tool.schema.internal.SchemaCreatorImpl apply
INFO: HHH000476: Executing import script 'org.hibernate.tool.schema.internal.exec.S
Hibernate:
    insert
    into
        emptab
        (ename, esal, eid)
    values
        (?, ?, ?)
Hibernate:
    select
        max(unqId)
    from
        empprjstab
Hibernate:
    insert
```

***) Native SQL:-**

Hibernate supports writing SQL Queries for multiple row operation. Supports both select and non-select operations.

*** compared to HQL, SQL is faster but SQL Dependends on Database (Native mode), where as HQL is DB independent.

*** if application uses fixed database (No DB Change) then only use Native SqL

** here native means Database Specific.

** No Dialect is involved in Native SQL (So no Conversion, it is faster)

Coding steps for Namtive SQL:-

1. Create SQL String (either for select or non-select operation)
   String sql="……";
2. Create Native Query object using SQL String.
   NativeQuery q=ses.createNativeQuery(sql);
3. Set Parameters ot query if exist
   Susng q.setParameter(name,data);
4. Execute query using list() or uniqueResult() for select operations executeUpdate()for non-select opeartions.

   **** package is: org.hibernate.query**

   ** NativeQuery supports positional Parameters om SQL, index number starts from one(1) [1,3,…..] where as HQL index number starts from zero(0) [0,1,2….]

   Ex:-- No model class required. Only .cfg.xm., util classes

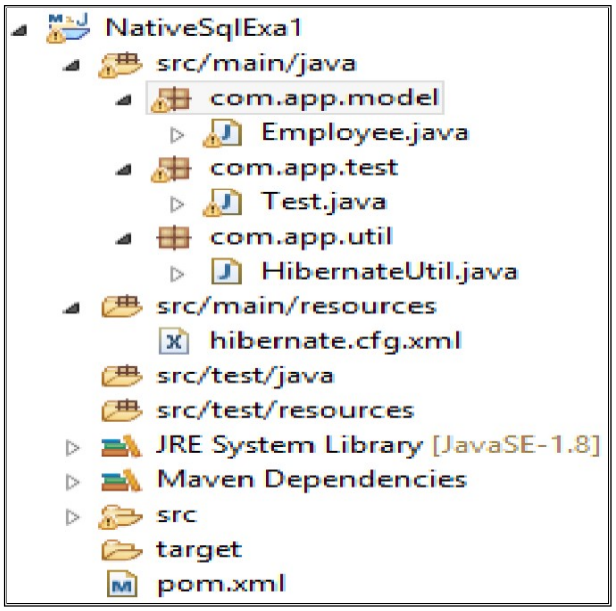
   - Test class:// cfg,sf,ses.
   - //1. Create SQl String

String sql="select ename,esal, form emptab where eid>=?";
//2. Create NativeQuery object
NativeQuery q=ses.createnativeQuery(sql);
//3. Provide parameters

```
        q.setParameter(1,10);
//4. Execute query
List<Object[]> obs=q.list();
//print data
for (Object[]ob:obs){
System.out.println(ob[0]+","+ob[1]);
}
```

Native SQL

1. **Folder structure**



2. **Pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>NativeSqlExa1</artifactId>
<version>1.0</version>
<dependencies>
        <dependency>
                <groupId>org.hibernate</groupId>
                <artifactId>hibernate-core</artifactId>
                <version>5.2.17.Final</version>
        </dependency>
        <dependency>
                <groupId>mysql</groupId>
                <artifactId>mysql-connector-java</artifactId>
                <version>5.1.6</version>
```

```
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.7.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
			<plugin>
				<artifactId>maven-assembly-plugin</artifactId>
				<executions>
					<execution>
						<phase>package</phase>
						<goals>
							<goal>single</goal>
						</goals>
					</execution>
				</executions>
				<configuration>
					<archive>
						<manifest>
							<mainClass>com.app.test.Test</mainClass>
						</manifest>
					</archive>
					<descriptorRefs>
					<descriptorRef>jar-withdependencies</descriptorRef>
					</descriptorRefs>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. Model class

```
package com.app.model;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.CollectionTable;
import javax.persistence.Column;
import javax.persistence.ElementCollection;
```

```
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.Table;
import org.hibernate.annotations.CollectionId;
import org.hibernate.annotations.GenericGenerator;
import org.hibernate.annotations.Type;
@Entity
@Table(name="emptab")
public class Employee {
@Id
@Column(name="eid")
private int empId;
@Column(name="ename")
private String empName;
@Column(name="esal")
private double empSal;
public Employee() {
	super();
}
public int getEmpId() {
	return empId;
}
public void setEmpId(int empId) {
	this.empId = empId;
}
public String getEmpName() {
	return empName;
}
public void setEmpName(String empName) {
	this.empName = empName;
}
public double getEmpSal() {
	return empSal;
}
public void setEmpSal(double empSal) {
	this.empSal = empSal;
}
@Override
public String toString() {
	return "Employee [empId=" + empId + ", empName=" + empName + ", empSal=" +
empSal + "]";
}
}
```

4. HibernateUtil

```
package com.app.util;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtil {
	private static SessionFactory sf=null;
	static
	{
		sf=new Configuration().configure().buildSessionFactory();
	}
	public static SessionFactory getSf()
	{
		return sf;
	}
}
```

5. Hibernate.cfg.xml (without dialect use also work)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

To converts this List<Object[]> to List<T>
Use method query.addEntity<T.class>
addEntity() method converts result from Object[] to model class object. (finally List<T>)

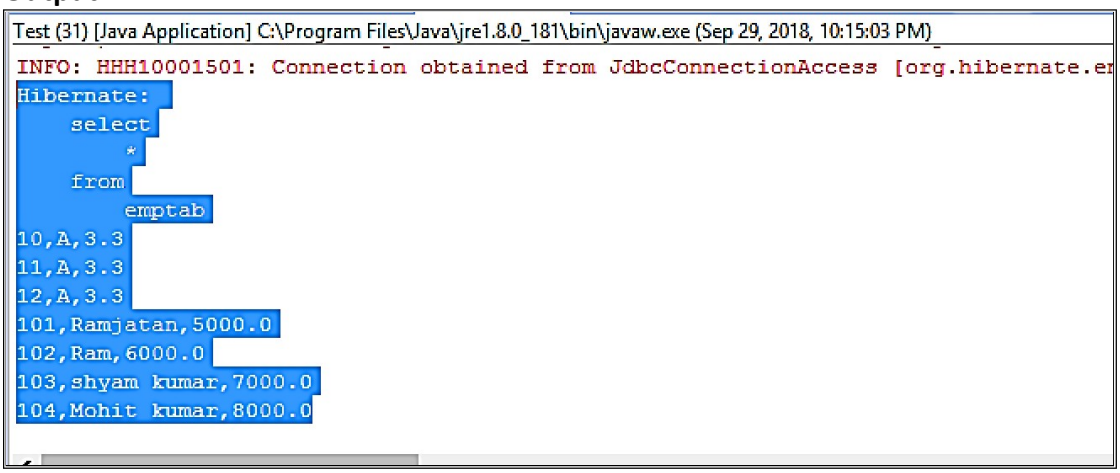
EX#1. (without addEntity-full loading)

Test class:(before insert the record after that native query write)

```
package com.app.test;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.NativeQuery;
import com.app.util.HibernateUtil;
```

```
publicclass Test {
	publicstaticvoid main(String[] args) {
	try(Session ses=HibernateUtil.getSf().openSession()){
	String sql1="select *from emptab";
	NativeQueryq=ses.createNativeQuery(sql1);
	List<Object[]>obs=q.list();
	for(Object[]ob:obs){
	System.out.println(ob[0]+","+ob[1]+","+ob[2]);
	}
	}catch(Exception ex)
	{
		ex.printStackTrace();
	}
}
}
```

Output:-



Ex#2: using addEntity –full loading

```
package com.app.test;
import java.util.List;
import org.hibernate.Session;
import org.hibernate.query.NativeQuery;
import  com.app.model.Employee;
import com.app.util.HibernateUtil;
publicclass Test {
	publicstaticvoid main(String[] args) {
	try(Session ses=HibernateUtil.getSf().openSession()){
		String sql="select *from emptab";
		NativeQueryq=ses.createNativeQuery(sql);
		q.addEntity(Employee.class);
		List<Employee>emps=q.list();
		emps.forEach(System.out::println);
```

```
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output:

```
Test (31) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 29, 2018, 10:38:4
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
Hibernate:
    select
        *
    from
        emptab
Employee [empId=10, empName=A, empSal=3.3]
Employee [empId=11, empName=A, empSal=3.3]
Employee [empId=12, empName=A, empSal=3.3]
Employee [empId=101, empName=Ramjatan, empSal=5000.0]
Employee [empId=102, empName=Ram, empSal=6000.0]
Employee [empId=103, empName=shyam kumar, empSal=7000.0]
Employee [empId=104, empName=Mohit kumar, empSal=8000.0]
```

*) SQL is case-insensitive (both upper and lower case letters are accepted)
*) if query returns one row use method uniqueResult().
*) Transaction required for non-select SQL operations.

=======EX====non –select operation----

Test class:

```
package com.app.test;
import org.hibernate.Session;
import  org.hibernate.Transaction;
import org.hibernate.query.NativeQuery;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
        Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
                tx=ses.beginTransaction();
                String sql="update emptab set ename=:a, esal=:b where eid=:c";
                NativeQueryq=ses.createNativeQuery(sql);
                q.setParameter("a","Ramesh kumar");
                q.setParameter("b",5000);
                q.setParameter("c",10);
                intcount =q.executeUpdate();
                tx.commit();
                ses.close();
                System.out.println(count);
```

```
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output:

```
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hiberna
Hibernate:
    update
        emptab
    set
        ename=?,
        esal=?
    where
        eid=?
1
```

SQL Query Area

```
1 SELECT * FROM emptab e;
```

eid	ename	esal
10	Ramesh kumar	5000
11	A	3.3
12	A	3.3
101	Ramjatan	5000
102	Ram	6000
103	shyam kumar	7000
104	Mohit kumar	8000

❖ **Validation API in Hibernate:-**

Validations are used to avoid invalid data input to Application.

1). Validations are two types. Those are

a). Client[Browser] side Validations

b). server side Validations.

⇨ Client side Validations are implemented using Scripting Language .

Ex:- Java Script…

⇨ Server side validations are implemented using Programming Languages.

⇨ Ex:- Java, Hibernate Validation Apl, Spring Validation API.

❖ To get Validation API, we should add below dependency in Pom.xml;

```
<dependency>
        <groupId>org.hibernate</groupId>
        <artifactid>hibernate-validator</artifactId>
        <version>5.2.5.Final</version>
```

</dependency>

=========or==========

- Download Jars:-
  a. Hibernate-validator-5.x.y.jar and
  b. Validation-api-1.0.jar, manually and add to build path.
- Hibernate validation API is used to validate model class object data, before insert/update into database table.

Example:-----Code----------

*) All validation Annotation are given iin package "javax.validation.constraints"

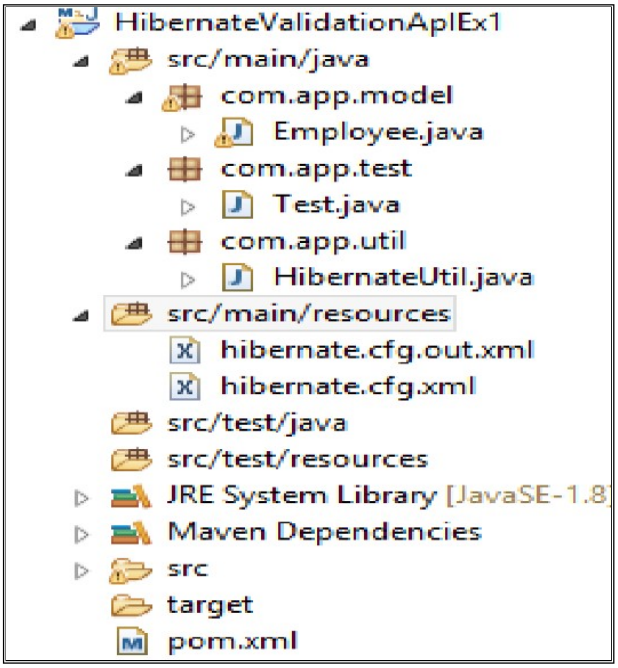
String type Validatation

a. **@NotNull:** It will not accept null values as input data.
b. **@Size (min,max):** it is used to provide min and max size of a input string
c. **@Pattern(regex):** it is used to provide a Format to input String.

***** All validation Annotation provide their default messages, to define our own use message=" " attribute to validation Annotation.**

====== 1. Model class code=====

1. **Folder Structure**



2. **Pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>HibeIDBAGExa1</artifactId>
<version>1.0</version>
<dependencies>
```

```
		<dependency>
	<groupId>org.hibernate</groupId>
	<artifactId>hibernate-validator</artifactId>
	<version>5.2.5.Final</version>
</dependency>
<dependency>
<groupId>javax.el</groupId>
<artifactId>javax.el-api</artifactId>
<version>3.0.1-b06</version>
</dependency>
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.0.5</version>
		</dependency>
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.1.11.Final</version>
		</dependency>
	</dependencies>
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import  javax.validation.constraints.Pattern;
import javax.validation.constraints.Size;
```

```
@Entity
@Table(name="emptab2")
public  class Employee {
	@Id
	@Column(name="eid")
	private int empId;
	@NotNull(message="Please Entr Name")
	@Size(min=2,max=5,message="Enter 2-5 chars only")
	@Pattern(regexp="[A-Za-z]{2,5}",message="please enter Uppdercase 2-5 char only")
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	private double empSal;
	public Employee() {
		super();
	}
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName + ", empSal="
+ empSal + "]";
	}
	}
```

4. **Same as before(HibernateUtil,hibernate.cfg.xml)**
5. **test class:**

```
package  com.app.test;
import org.hibernate.Session;
```

```
import  org.hibernate.Transaction;
import  com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
        public static void main(String[] args) {
        Transaction tx=null;
        try(Session  ses=HibernateUtil.getSf().openSession()){
                tx=ses.beginTransaction();
                Employee e1=new Employee();
                e1.setEmpId(55);
                e1.setEmpName("Ram");
                e1.setEmpSal(33.3);
                ses.save(e1);
                tx.commit();
                //ses.close();
        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output

```
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
Hibernate:

    create table emptab2 (
         eid integer not null,
         ename varchar(255),
         esal double precision,
         primary key (eid)
    )
Sep 30, 2018 6:24:05 AM org.hibernate.resource.transaction.back
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess
Sep 30, 2018 6:24:05 AM org.hibernate.tool.schema.internal.Sche
INFO: HHH000476: Executing import script 'org.hibernate.tool.sc
Hibernate:
    insert
    into
         emptab2
         (ename, esal, eid)
    values
         (?, ?, ?)
```

*) hibernate throws :constrainViolation Exception " if conditation is not matched with model class data.

- **Numeric Type validations:-**

  @Min(value=….)and @Max(value=……)

*) Date Type Validations:-

@Past:- It must be past date only

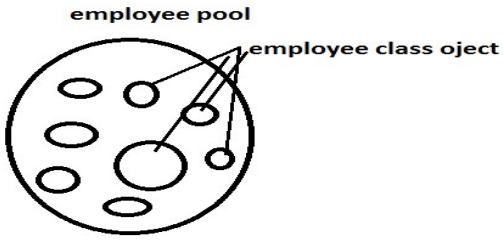
Ex:- Date of birth, Date of Creation

@Future:- it must be upcoming date

Ex: - Date of Expire, date of release
***) Boolean Type:**-
@AssertTrue: Expecting true only
@AssertFalse:expecting false only
----------Pattern API:---- core java Ex----

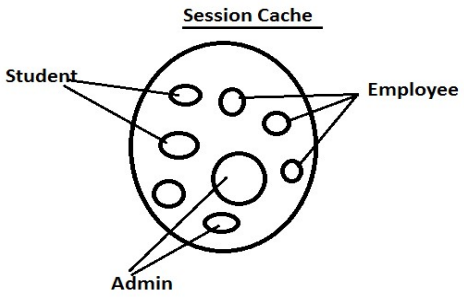
```
Package com.app;
Public  class Test {
Public static void main(String arr[])
{
String sname=”Sam”;
//1. Create Pattern object
Pattern p=new Pattern.compile(“[A-Za-z]{3,6}”);
//2. Compare exp with input String
Matcher m=p.matcher(sname);
//3. Get result (mateched or not?)
Boolean flag =m.matches();
Sysout(flag);
}
}
```

Temporarly memory in hibernate :
Hibernate supports two types of temporarlly memory
Pool: it is a group of similar object that is all object related to one class
Ex:-



Hibernate supports connection pool concept that is group of connection object
Cache:- it is a group of desimilar object it means group of object belongs to different classes.
Hibernate supports three types of Cache , those are:

a. Session Cache
b. Query Cache
c. Session Factory Cache
Ex:-



C3P0(See Three Pee OOO):-it is a 3rd party connection poll service provided for java application.

***)** It creates and manages connection Poll without communicating to Programmer (Automated).

***)** It needs only input configuraton details by programmer, like

- ⇨ Minimum no.of connections in pool
- ⇨ Maximum no.of connections in pool
- ⇨ One connection can handle how many operations.
- ⇨ When connection should be marked as not use (Idle)
- ⇨ When an un-used connection should be removed from Poll (timeout).

*). C3P0 is interated with hibernate and Pooling is automated by hibernate.

*). In Hibernate Configuration file (.cfg.xml) we must provide below keys with example values.

----------keys and description----------

Hibernate.c3p0.min_size=Minimum number of JDBC connections in the pool. Hiberte default:1

Hibernate.c3p0.max_size=Maximum number of JDBC connections in the pool. Hibernate default :100.

Hibernate.c3p0.timeout=when an idle connection is removed from the pool (in second). Hibernate default:0,never expire.

Hibernate.c3p0.max_statements= Number of prepared statements will be provided. Hibernate default:0

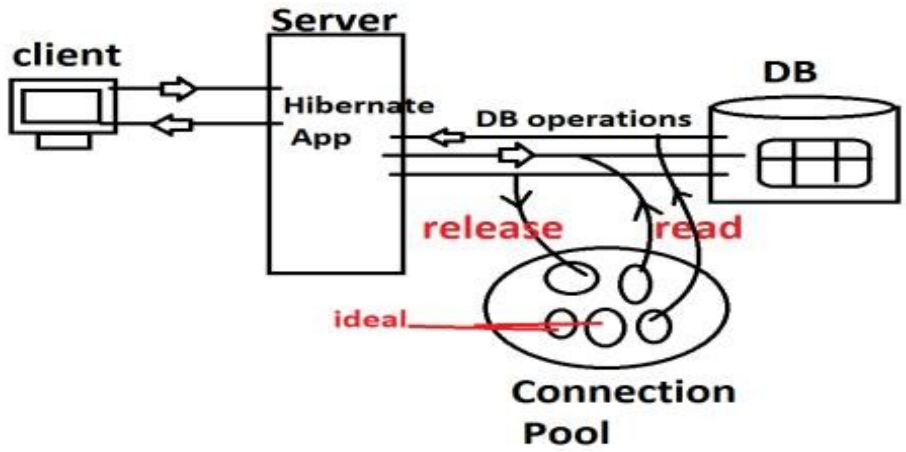
Hibernate.c3p0.idle_test_period= idle time in seconds before a connection is automatically validated. Hibernate default:0

Q). why and When connection pool is used?

Ans.=> In case of large applications, that is used by more no.of users and to provide fast service use connection pool.

Q). How it works?

Ans.=>

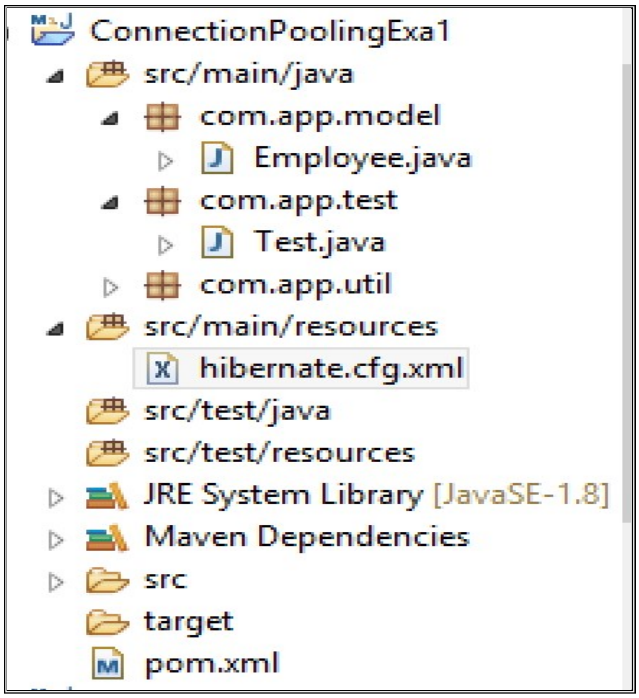


Steps to imiplement Connection Poling in hibernate Application:-

1. Add Jars to Project
   In case of maven
   <dependency>
   <groupId>org.hibernate</groupId>
   <artifactId>hibernate-c3p0</artifactid>
   <version>5.2.17.Final</version>
   </dependency>
   - If maven not used then copy jars from
   ⇨ Hibernate-release-5.2.17.Final
   ⇨ Lib > optional > c3p0 (3 jars)
   - In hibernate.cfg.xml provide below key=value pairs
     <property name="hibernate.c3p0.min_size">5</property>
     <property name="hibernate.c3p0.max_size">20</property>
     <property name="hibernate.c3p0.timeout">300</property>
     <property name="hibernate.c3p0.max_statements">50</property>
     <property name="hibernate.c3p0.idle_test_period">3000</property>
   - Write any one example and run it.
   - Check in console :Initializing c3p- pool…

==========code==========

1. **Folder structure**



2. Pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>ConnectionPoolingExa1</artifactId>
<version>1.0</version>
<dependencies>
<dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.0.5</version>
</dependency>
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>5.1.11.Final</version>
                </dependency>
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-c3p0</artifactId>
<version>5.2.17.Final</version>
</dependency>
        </dependencies>
```

```
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.8.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>
	</build>
</project>
```

3. Model class

```
package com.app.model;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="emptab2")
public  class Employee {
	@Id
	@Column(name="eid")
	private int empId;
	@Column(name="ename")
	private String empName;
	@Column(name="esal")
	private double empSal;
	public Employee() {
		super();
	}
	public int getEmpId() {
		return empId;
	}
	public void setEmpId(int empId) {
		this.empId = empId;
	}
	public String getEmpName() {
		return empName;
	}
	public void setEmpName(String empName) {
		this.empName = empName;
```

```
	}
	public double getEmpSal() {
		return empSal;
	}
	public void setEmpSal(double empSal) {
		this.empSal = empSal;
	}
	@Override
	public String toString() {
		return "Employee [empId=" + empId + ", empName=" + empName + ", empSal="
+ empSal + "]";
	}
}
```

4. **Same as before (HibernateUtil)**

5. **Hibernate.cfg.xml(Connection pooling)**

Hibernate.cfg.xml

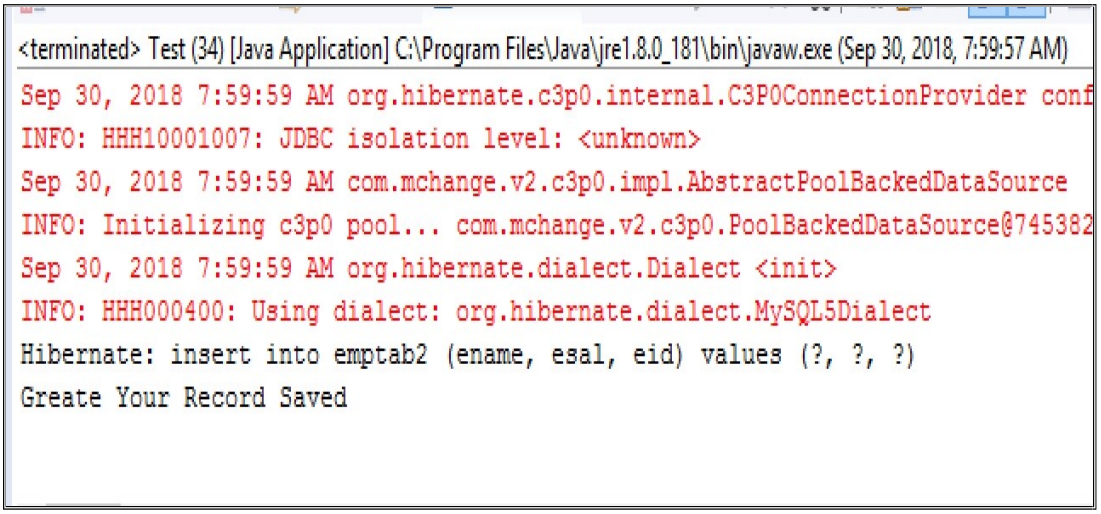
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
      "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
      "http://hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.
          connection.provider_class">org.hibernate.connection.C3P0ConnectionProvider
</property>
<property name="hibernate.c3p0.min_size">7</property>
<property name="hibernate.c3p0.max_size">53</property>
<property name="hibernate.c3p0.timeout">100</property>
<property name="hibernate.c3p0.max_statements">50</property>
<property name="hibernate.c3p0.idle_test_period">1000</property>
<mapping class="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

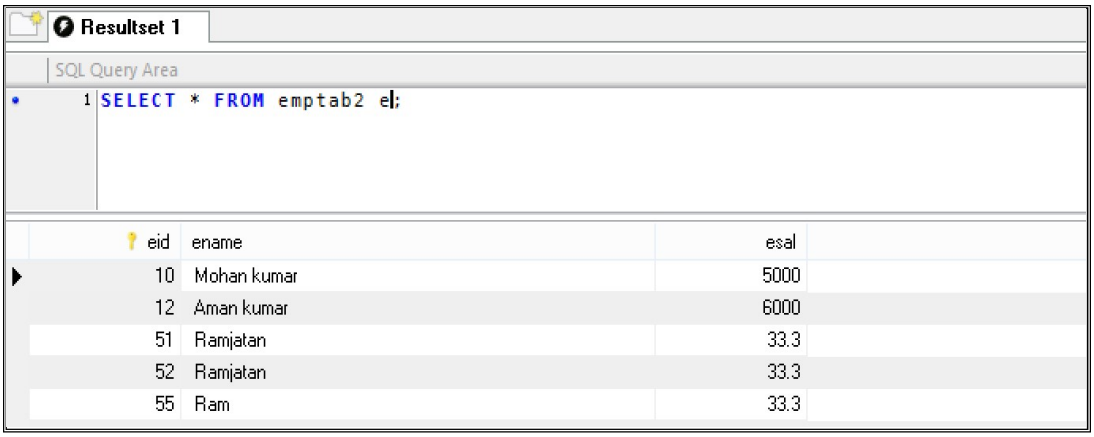
6. **Test class**

```
package  com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
```

```
import  com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
	public static void main(String[] args) throws InterruptedException {
	Transaction tx=null;
	try(Session ses=HibernateUtil.getSf().openSession()){
		tx=ses.beginTransaction();
		Employee e1=new Employee();
		e1.setEmpId(10);
		e1.setEmpName("Mohan kumar");
		e1.setEmpSal(5000);
		try {
			Thread.sleep(1000);
		}catch(InterruptedException e)
		{
			e.printStackTrace();
		}
		ses.save(e1);
		tx.commit();
		//ses.close();
		System.out.println("Greate Your Record Saved");
	}catch(Exception ex)
	{
		ex.printStackTrace();
	}
}
}
```

Output :





Cache

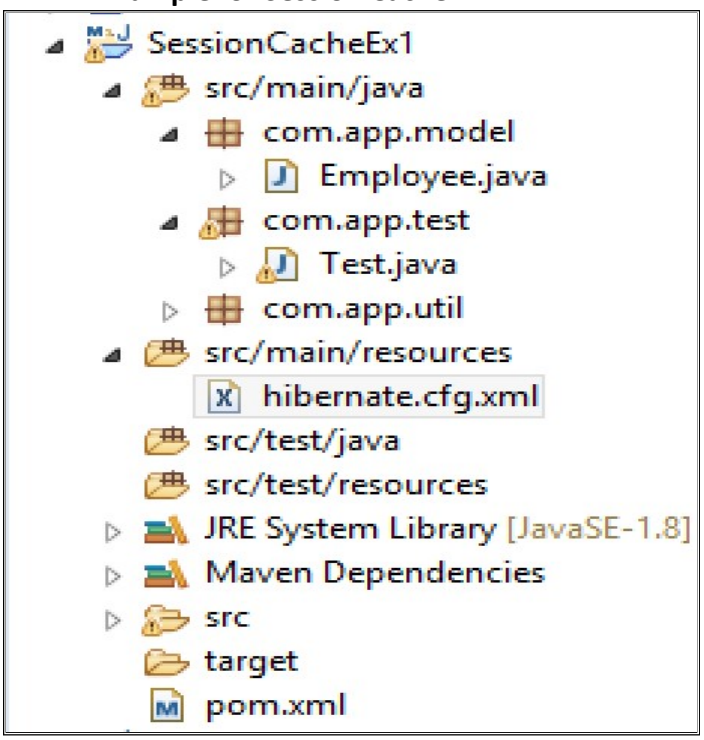
Cache:- it is a temporally memory which holds groups of different Types of objects at one place.
Ex: one Employee object, 3 admin object, 12 student objects etc.
Q). why chche?
Ans. To reduce network calls between Hibernate Application and database while accessing same data multiple times.
In realtiime Application and Database servers are placed in different locations and both are connected using network.

a. In JDBC Programming " No cache" concept available, in Hibernate cache concept is exist.
========Types of Caches in Hibernate========
Hibernate supporst 3 types of caches. Those are:
1. Session Cache- I Level Cache
2. Session Factory Cache –II Level Cache
3. Query Cache (Sub type of Session Cache)

1. **Session Cache:-** By default Hbiernate Framework enabled this cache and managed itself. Programmer can not handle this Cache type.

- **When openSession() method is called Session Cache will be created and ses.close()** called then it will be

======Example for Session Cache====



1. **Pom.xml (**same as before like pom.xml file)
2. **Model class ,HibernateUtil class,hibernate.cfg.xml (**same as before like)
3. **Test class**(before insert one more Record)

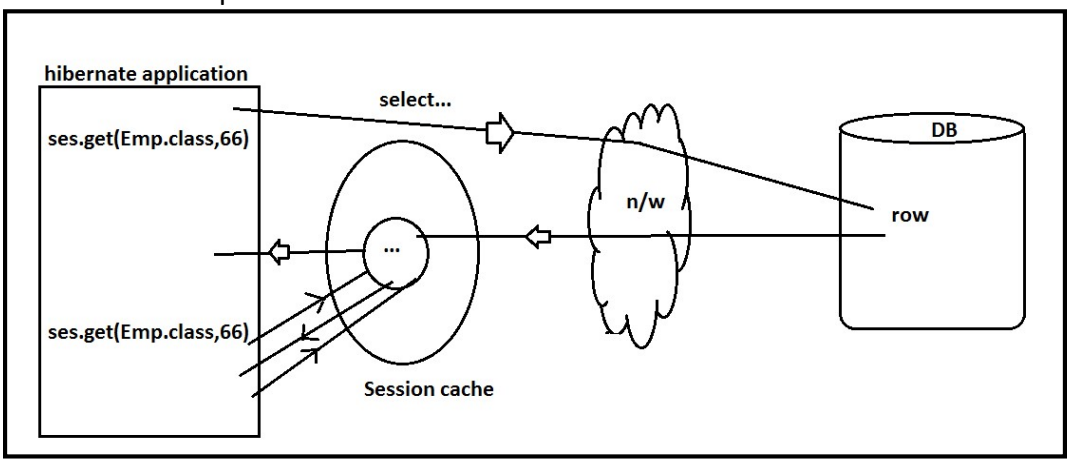
```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
public class Test {
    public static void main(String[] args) {
            try(Session ses=HibernateUtil.getSf().openSession()){
            Employee e1=ses.get(Employee.class,101);
            System.out.println(e1);
            Employee e2=ses.get(Employee.class,101);
            System.out.println(e2);
    }catch(Exception ex)
    {
            ex.printStackTrace();
    }
}
}
```

output

```
Test (35) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Sep 30, 2018, 10:39:59 AM)
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org
Hibernate:
    select
        employee0_.eid as eid1_0_0_,
        employee0_.ename as ename2_0_0_,
        employee0_.esal as esal3_0_0_
    from
        emptab2 employee0_
    where
        employee0_.eid=?
Employee [empId=101, empName=Ramjatan, empSal=5000.0]
Employee [empId=101, empName=Ramjatan, empSal=5000.0]
```

4. ** Here we are Calling get() method twice with same Id, So cache returns object on @nd time get. Finally one network call is made.

*** No.of Select queries=no.of networkcalls

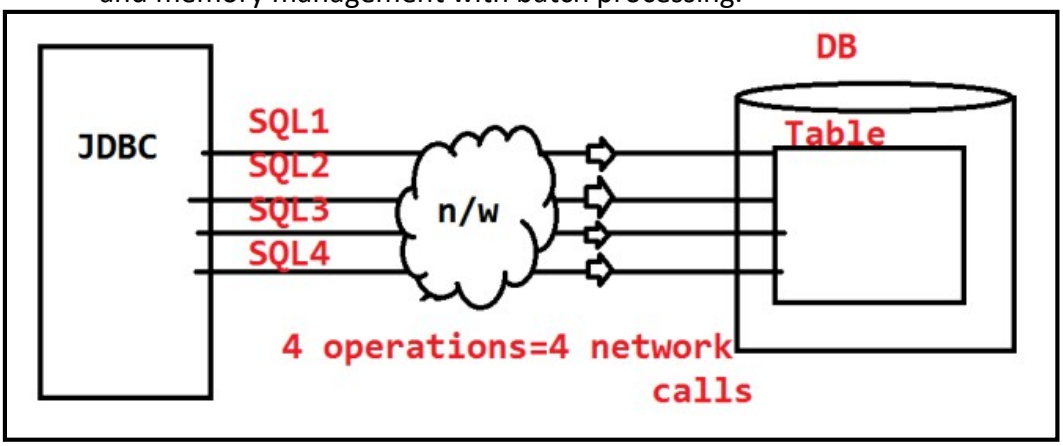


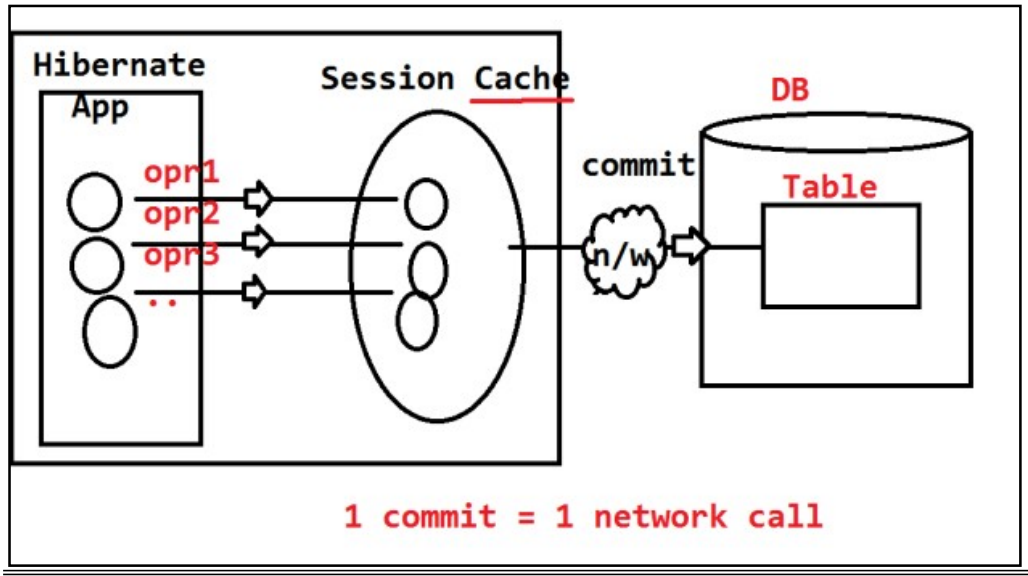
Session Cache also works for non-select operations. Here network calls are made when tx.commit() is called.

If tx.rollback() is called then no n/w calls are made.

*) Compared to JDBC, Hibernate reduces network calls by using implicit batch processing.

*) we can reduce network calls in JDBC also but programmer should do external coding and memory management with batch processing.



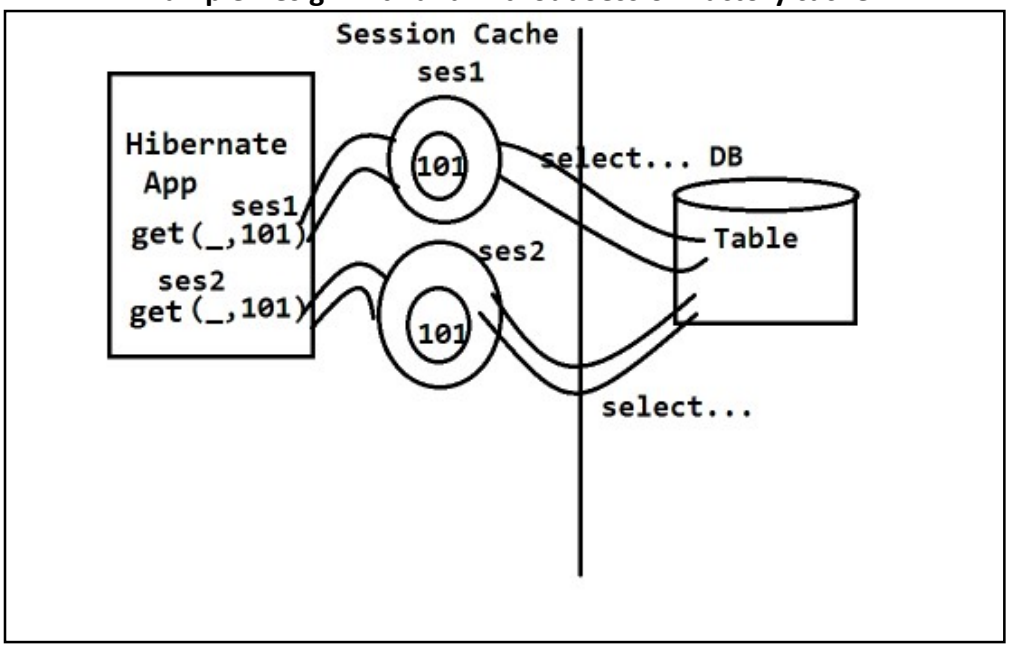


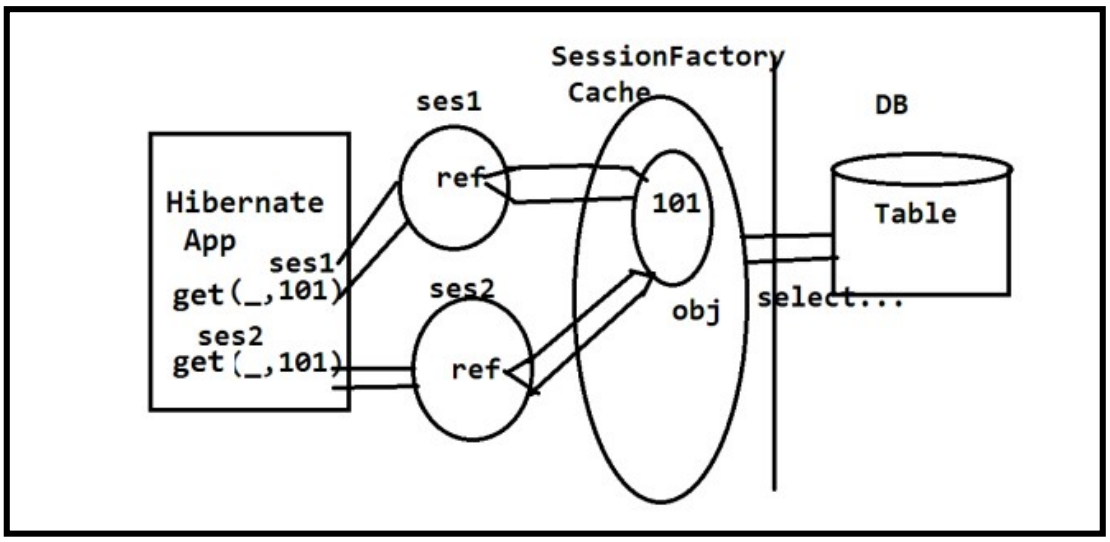
2. Session Factory Cache:-

This one must be handled by Programmer only for commonly accessed modules in Application (Not for all modules).

- Two session caches will not communicates with each other, for common data share.
- In this case use only Global cache management ie. Called as SessionFactory Cache, also called as II level cache.

===Example Design with and without Session Factory cache===

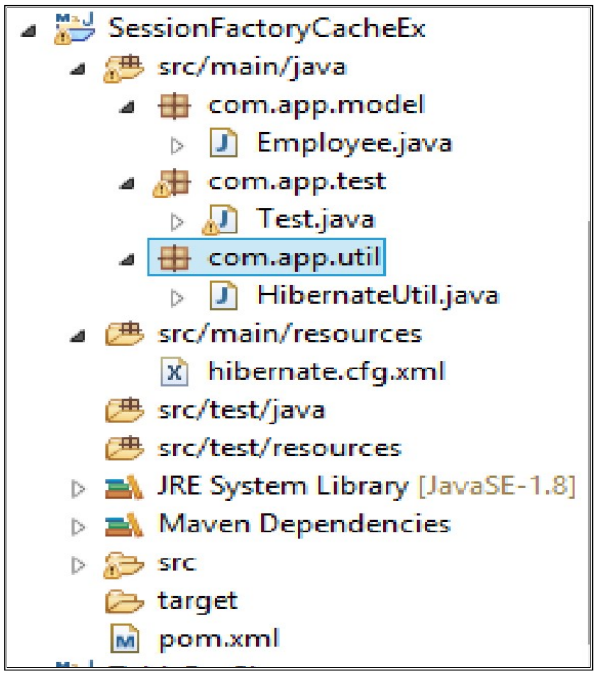




Steps To implement SessionFactory Cache:

==========code with folder structure==========

1. **Folder Structure**



2. **pom.xml**

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.nareshittech</groupId>
<artifactId>SessionFactoryCacheEx</artifactId>
<version>1.0</version>
<dependencies>
```

```
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>5.0.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-ehcache -->
<dependency>
<groupId>org.hibernate</groupId>
<artifactId>hibernate-ehcache</artifactId>
<version>5.2.17.Final</version>
</dependency>
<!-- https://mvnrepository.com/artifact/net.sf.ehcache/ehcache-core -->
<dependency>
<groupId>net.sf.ehcache</groupId>
<artifactId>ehcache-core</artifactId>
<version>2.6.6</version>
</dependency>
<dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>5.1.11.Final</version>
                </dependency>
</dependencies>
<build>
                <plugins>
                        <plugin>
                                <groupId>org.apache.maven.plugins</groupId>
                                <artifactId>maven-compiler-plugin</artifactId>
                                <version>3.8.0</version>
                                <configuration>
                                        <source>1.8</source>
                                        <target>1.8</target>
                                </configuration>
                        </plugin>
                </plugins>
        </build>
</project>
```

3. **Model class ,HibernateUtil class** (same as before program like)
4. **Hibernate.cfg.xml**

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

```
<session-factory>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ram</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">system</property>
<property name="hibernate.dialect">org.hibernate.dialect.MySQL5Dialect</property>
<property name="hibernate.show_sql">true</property>
<property name="hibernate.cache.use_second_level_cache">false</property>
<property name="hibernate.cache.provider_class">org.hibernate.cache.EhCacheProvider
</property>
<property
name="hibernate.cache.region.factroy_class">org.hibernate.cache.ehcache.EhCacheRegionFact
ory</property>
<property name="hibernate.hbm2ddl.auto">update</property>
<mapping class="com.app.model.Employee"/>
</session-factory>
</hibernate-configuration>
```

5. Test class:-

```
package com.app.test;
import org.hibernate.Session;
import org.hibernate.Transaction;
import com.app.model.Employee;
import com.app.util.HibernateUtil;
publicclass Test {
        publicstaticvoid main(String[] args) {
                Transaction tx=null;
        try(Session ses=HibernateUtil.getSf().openSession()){
                Employee e1=ses.get(Employee.class,102);
                System.out.println(e1);
                Employee e2=ses.get(Employee.class,102);
                System.out.println(e2);

        }catch(Exception ex)
        {
                ex.printStackTrace();
        }
}
}
```

Output:-

```
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
Sep 30, 2018 12:33:47 PM org.hibernate.resource.transaction.backend.jdbc.interna
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.
Hibernate: select employee0_.eid as eid1_0_0_, employee0_.ename as ename2_0_0_,
Employee [empId=102, empName=Ram Kumar, empSal=6000.0]
Employee [empId=102, empName=Ram Kumar, empSal=6000.0]
```

** Expected output: select query executed (printed)only once.

Note:-

1. To create 2nd Level cache, we should use one 3rd party provider API>
   Ex: EasyHibernateCache (EHCache)
2. It can be integrated to handle both SessionFactoryand query cache.

Major keys are:

Use_secnd_level_cach(default is false)
Must be set true.

3. Cache Memory created by:
   ** provider_class=EhCacheProvider
4. At model class level (which Is most frequently used) add @Cache with CacheConcurrencyStrategy(enum).
   Possible values are:
   READ_ONLY=Cache for select operation
   READ_WRITE= Cache for select and non-select operations
   TRASACTIONAL= Cache for non-select operation
   NONSTRICT_READ_WRITE= Cache for both select and non-select opr. Start Cache for current session. Ignore old session, for any updated data.

---

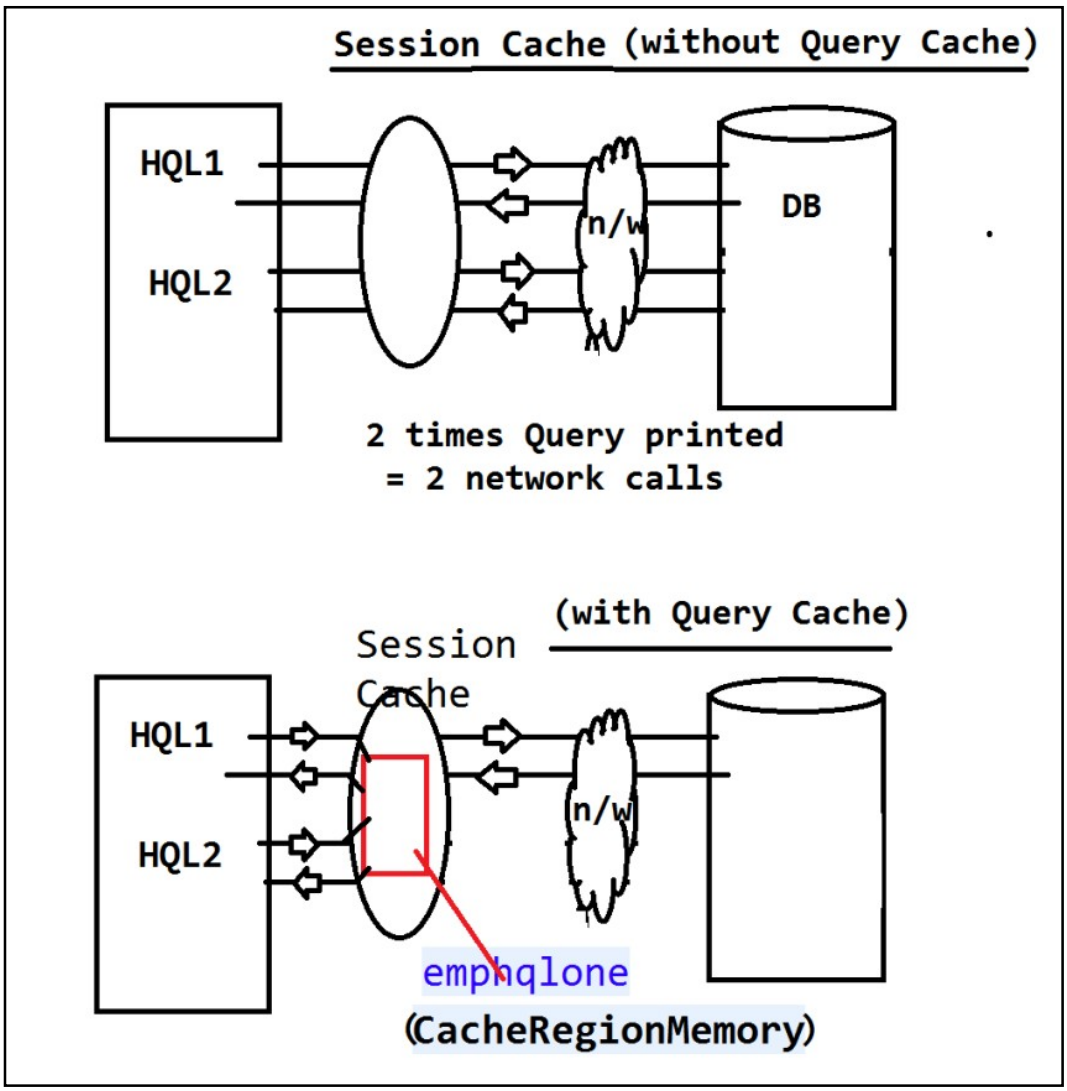
3. **Query Cache:-** it is a sub-type of SessionCache.
   - By default cache is applied for single row operations, ie:save(), update(), delete(),get(),load(), saveorUpdate()…..
   - But HQL any return multiple rows so, cache is disable for this.
   - Here Query cache enable cache management for HQL also.but only for few or specific queries.

   Q. how to enable?
   Ans.
   One key for enable this proves:
   <property name="hibernate.cache.use_query_cache">true</property>
   Another key for creating memory for HQL result Store [region memory]

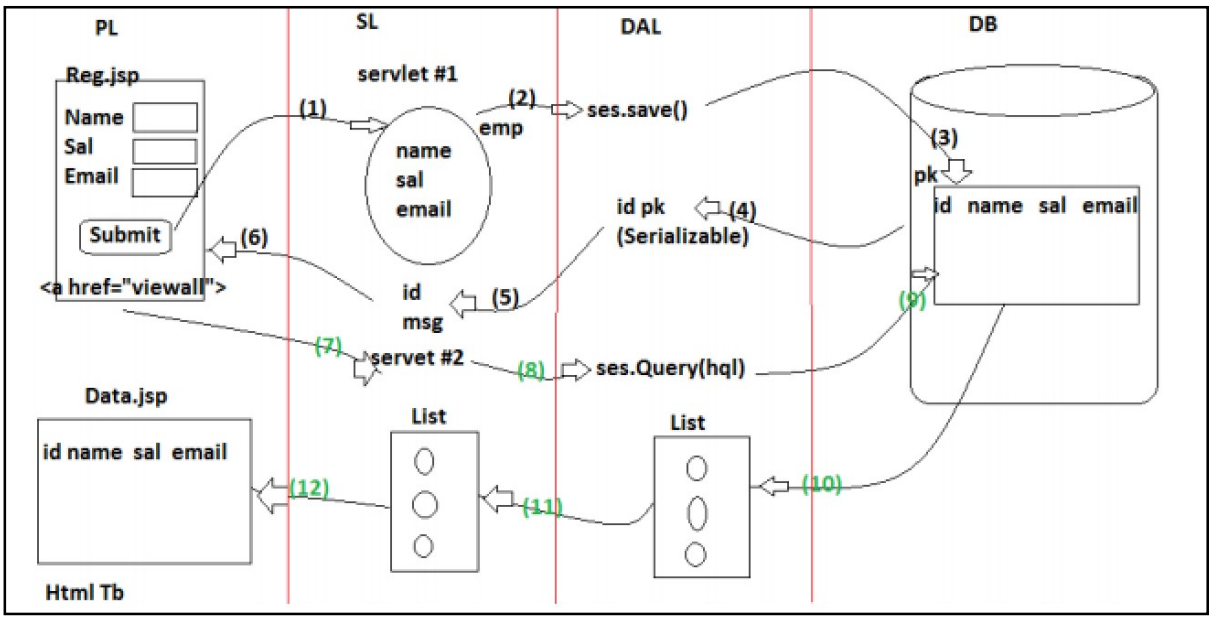
<property
name="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

- Test class:-//cfg.sf,ses.
  Session sesq=Hibernateutil.getSf().openSession();
  Employee e1=(Employee)ses1.createQuery("form com.app.Employee where empId=?")
  .setParameter(0,66)
  .setCacheable(true)
  .setCacheRegion("emphqlone")
  .uniqueResult();
  Sysout(e1);

  Employee e2=(Employee)ses1.createQuery("form com.app.Employee where empId=?")
  .setParameter(0,66)
  .setCacheable(true)
  .setCacheRegion("emphqlone")
  .uniqueResult();
  Sysout(e2);
  Expected output: Select … query printed only once.



SERVLET AND HIBERNATE INTIGRATION APPLICATION



POJI-POJO Design Pattern:-

It is used to develop layer coding in loolsely –coupled manner. Ie. If Layer#1. Is modified then it should not effect layer#2 code.

POJI= plain old java interface (I)

POJO= plain old java object ©

In place of test class in hibernate, POJI-POJO is used to perform DB Operation.

Coding Steps:-

1. Configure Server in eclipse/STS
   - Server tab > Right click > new > Server
   - Choose Apache Tomcat > next > Browser
   - Choose location > next Finish
2. Create Maven Project
   - File > New > Maven Project > next
   - Search for "webApp"
   - Choose "maven-Archtype-webapp" > next
   - Enter Details
     Group Id: org.nareshittech
     artifactId: ServletHibernateApp
     version : 1.0
   - Finish button
3. Configuration Server
   - Right click on Project > build Path
   - Configuration build path > library tab

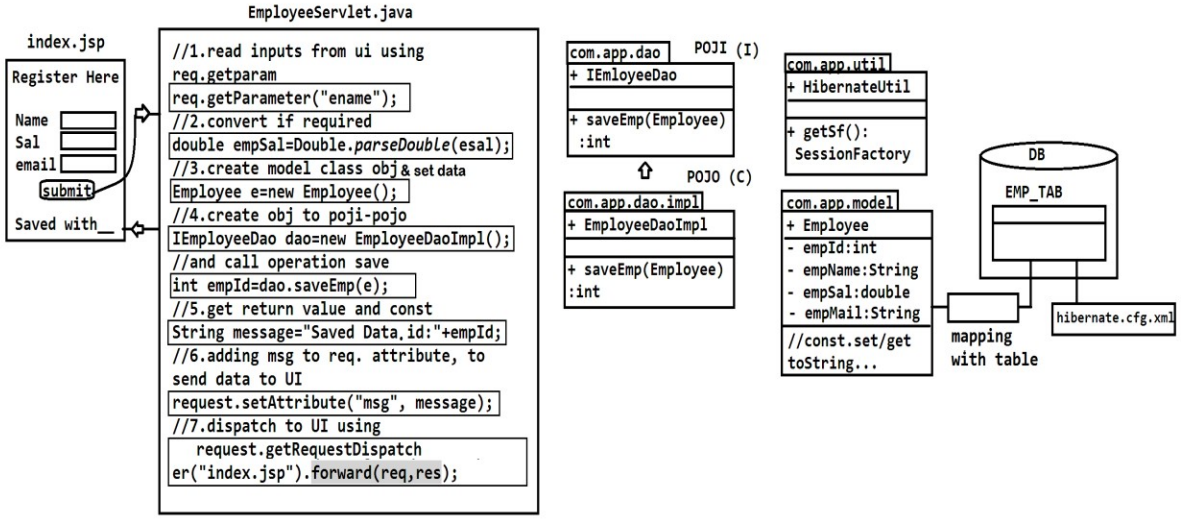
- Click on “Add Libray”
- Choose server runtime > select Tomcat
- Finish

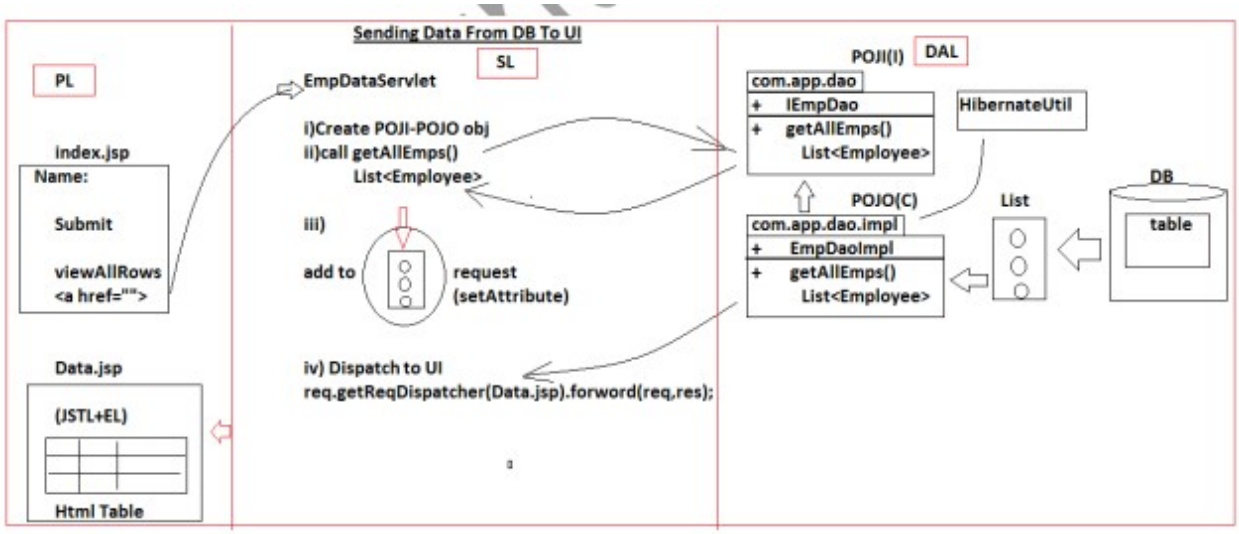
4. Provids POM.xml details
- Dependencies for hibernate, maven ,JSTL
- Build plugins

5. Update Maven Project
- Right Click on Project > Maven
- Update Project > Finish

==========coding order===

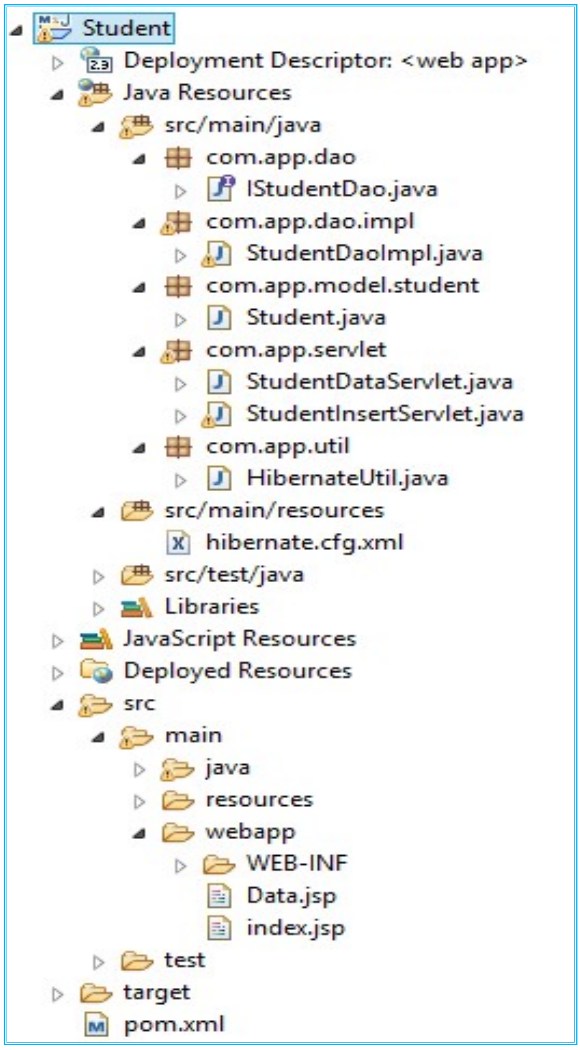
1. Model class =Employee.java
2. Cfg. File= Hibernate.cfg.xml
3. Util class= HibernateUtil.java
4. POJI _POJO= IEmployeeDao.java and EmployeeInsertServlet.java
5. Register page= index.jsp
6. RegServlet =employee inset servlet.java
7. View page =Data.jsp
8. View servlet=EmployeeFeatchServlet.java





==========CODING ==========

1. Folder Structure



1. **Model class (Student.java)**

```
package com.app.model.student;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
/**
 * @author Ram
 *
 */
@Entity
@Table(name="StudentTab")
public class Student {
	@Id
	@Column(name="sid")
	private Integer stdId;
	@Column(name="sname")
	private String stdName;
	@Column(name="sfee")
	private Double stdFee;
	@Column(name="sage")
	private Integer age;
	@Column(name="smob")
	private Integer mob;
	public Student() {
		super();
	}
	public Student(Integer stdId, String stdName, Double stdFee, Integer age, Integer
mob) {
		super();
		this.stdId = stdId;
		this.stdName = stdName;
		this.stdFee = stdFee;
		this.age = age;
		this.mob = mob;
	}
	public Integer getStdId() {
		return stdId;
	}
	public void setStdId(Integer stdId) {
		this.stdId = stdId;
	}
	public String getStdName() {
		return stdName;
```

```
	}
	public void setStdName(String stdName) {
		this.stdName = stdName;
	}
	public Double getStdFee() {
		return stdFee;
	}
	public void setStdFee(Double stdFee) {
		this.stdFee = stdFee;
	}
	public Integer getAge() {
		return age;
	}
	public void setAge(Integer age) {
		this.age = age;
	}
	public Integer getMob() {
		return mob;
	}
	public void setMob(Integer mob) {
		this.mob = mob;
	}
	@Override
	public String toString() {
		return "Student [stdId=" + stdId + ", stdName=" + stdName + ", stdFee=" +
stdFee + ", age=" + age + ", mob="
				+ mob + "]";
	}
}
```

Hibernate.cfg.xml

```
<?xmlversion="1.0"encoding="UTF-8"?>
<!DOCTYPEhibernate-configurationPUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<propertyname="connection.driver_class">com.mysql.jdbc.Driver</property>
<propertyname="connection.url">jdbc:mysql://localhost:3306/test</property>
<propertyname="connection.username">root</property>
<propertyname="connection.password">system</property>
<propertyname="dialect">org.hibernate.dialect.MySQL5Dialect</property>
<propertyname="show_sql">true</property>
<propertyname="format_sql">true</property>
<propertyname="hbm2ddl.auto">update</property>
```

```
<mappingclass="com.app.model.student.Student"/>
</session-factory>
    </hibernate-configuration>
```

HibernateUtil.java

```
package com.app.util;

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

publicclass HibernateUtil {
privatestatic SessionFactory sf;
static
{
try {
        Configuration cfg=new Configuration().configure();
        sf=cfg.buildSessionFactory();
} catch (HibernateException e) {
        e.printStackTrace();
}
}
publicstatic SessionFactory getSf()
{
        returnsf;
}
}
```

POJI-POJO(Dao,DaoImp)

IStudentDao.java

```
package com.app.dao;

import java.util.List;

import com.app.model.student.Student;

publicinterface IStudentDao {
        publicint saveStudent(Student s);
        public List<Student> getAllStud();
//      public long getCount();
}
```

StudentDaoImp

```
package com.app.dao.impl;
```

```
import java.io.Serializable;
import java.util.List;

import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.query.Query;

import com.app.dao.IStudentDao;
import com.app.model.student.Student;
import com.app.util.HibernateUtil;

publicclass StudentDaoImpl implements IStudentDao {

	@Override
	publicint saveStudent(Student s) {
		Transaction tx=null;
		intstdId=0;
		try(Session ses=HibernateUtil.getSf().openSession())
		{
			tx=ses.beginTransaction();
			Serializable st=ses.save(s);
			stdId=(Integer)st;
			tx.commit();

		}catch(Exception ex)
		{
			tx.rollback();
			ex.printStackTrace();
		}
		returnstdId;
	}



	@Override
	public List<Student> getAllStud() {
		//it will select data from emptab
		String hql="from "+Student.class.getName();
		//List<Student> stud=null;
		Session ses=HibernateUtil.getSf().openSession();
			Queryq=ses.createQuery(hql);
			List<Student>stud=q.list();
			//send this data to UI ti display as a Table
			returnstud;
```

```
        }
}
```

Index.jsp

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"isELIgnored="false"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html; charset=ISO-8859-1">
<title>student page</title>
</head>
<body>
<center>
<formaction="insertStud"method="post">
<h1Style="background-color:yellow;">WELCOME TO STUDENT PAGE!!</h1>
<hr/>
<tablestyle="background-color:FloralWhite;font-family:arial;width:300px;height:200px;">
<tr>
<td><label>STUDENT-ID</label></td>
<td><inputtype="text"name="sid"></td>
</tr>
<tr>

<td><label>STUDENT-NAME</label></td>
<td><inputtype="text"name="sname"></td>
</tr>
<tr>
<td><label>STUDENT-FEE</label></td>
<td><inputtype="text"name="sfee"></td>
</tr>
<tr>

<td><label>AGE:-     </label></td>
<td><inputtype="text"name="age"></td>
</tr>
<tr>

<td><label>MOBILe:-  </label></td>
```

```
<td><inputtype="text"name="mob"></td>
</tr>
<tr>
<td></td>
<td><inputtype="submit"value="Insert"><inputtype="reset"></td>
</tr>

</table>

</form>
${message}
<br>
<ahref="StudData">Show Record</a>
</center>
</body>
</html>
```

StudentInsertServlet.java

```
package com.app.servlet;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.app.dao.IStudentDao;
import com.app.dao.impl.StudentDaoImpl;
import com.app.model.student.Student;

/**
 * Servlet implementation class StudentInsertServlet
 */
publicclassStudentInsertServletextends HttpServlet {

	publicvoid doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		//1. read the data from index.jsp file
		String sid=request.getParameter("sid");
		String stdName=request.getParameter("sname");
		String sfee=request.getParameter("sfee");
		String sage=request.getParameter("age");
		String smob=request.getParameter("mob");
```

```
                //2. parse the data
                intstdId=Integer.parseInt(sid);
                doublestdFee=Double.parseDouble(sfee);
                intAge=Integer.parseInt(sage);
                intMob=Integer.parseInt(smob);
                //3. create the object in model class
                Student st=new Student();
                //4. set the data in model class
                st.setStdId(stdId);
                st.setStdName(stdName);
                st.setStdFee(stdFee);
                st.setAge(Age);
                st.setMob(Mob);
                //5. create the object POJO-POJI
                IStudentDao dao=new StudentDaoImpl();
                //6.call operation saveStudent
                intresStd=dao.saveStudent(st);
                //7. final message
                String msg="Student '"+resStd+"' created";
                //8. add message
                request.setAttribute("message", msg);
                //9. request dispatcher
                request.getRequestDispatcher("index.jsp").forward(request,response);

        }
}
```

Data.jsp (view page)

```
<%@pagelanguage="java"contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"isELIgnored="false"%>
<%@taglibprefix="c"uri="http://java.sun.com/jsp/jstl/core"%>
<%@pageimport="com.app.model.*"%>
<!DOCTYPEhtmlPUBLIC"-//W3C//DTD HTML 4.01
Transitional//EN""http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1>Welcome to Student Data</h1>
<tableborder="1">
<tr>
```

```
        <th>STUDENT-ID</th>
        <th>STUDENT NAME</th>
        <th>STUDENT-FEE</th>
        <th>AGE</th>
        <th>MOBILE</th>
</tr>
<c:forEachitems="${list}"var="s">
<tr>
<td><c:outvalue="${s.stdId}"/></td>
<td><c:outvalue="${s.stdName}"/></td>
<td><c:outvalue="${s.stdFee}"/></td>
<td><c:outvalue="${s.age}"/></td>
<td><c:outvalue="${s.mob}"/></td>
</tr>
</c:forEach>
</table>
<br/>
<c:forEachbegin="1"end="${np}"var="i">
        <ahref="StudData?pn=${i}">
        <c:outvalue="${i}"/></a>  
</c:forEach>
</body>
</html>
```

StudentDataServlet.java

```
package com.app.servlet;
import java.io.IOException;
import java.util.List;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.app.dao.IStudentDao;
import com.app.dao.impl.StudentDaoImpl;
import com.app.model.student.Student;
/**
 * Servlet implementation class StudentDataServlet
 */
@SuppressWarnings("serial")
public class StudentDataServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                        //1. create object to POJI-POJO
                        IStudentDao dao=new StudentDaoImpl();
```

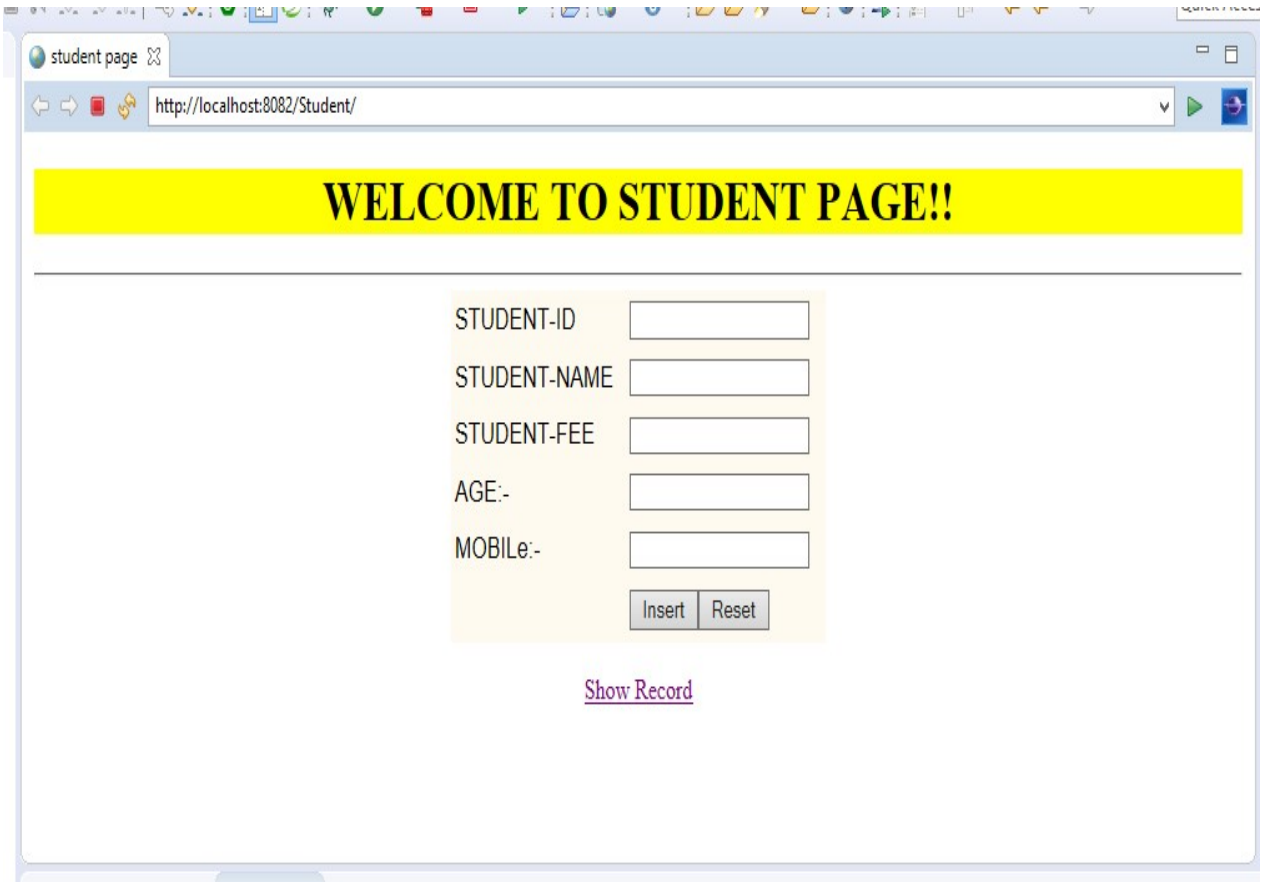
```
                    //2. call get operation
                    List<Student> st=dao.getAllStud();
                    //3. add list to request memory
                request.setAttribute("list",st);

                    //4. dispatch to UI
                    request.getRequestDispatcher("Data.jsp").forward(request,
response);
    }

}
```

Run:-

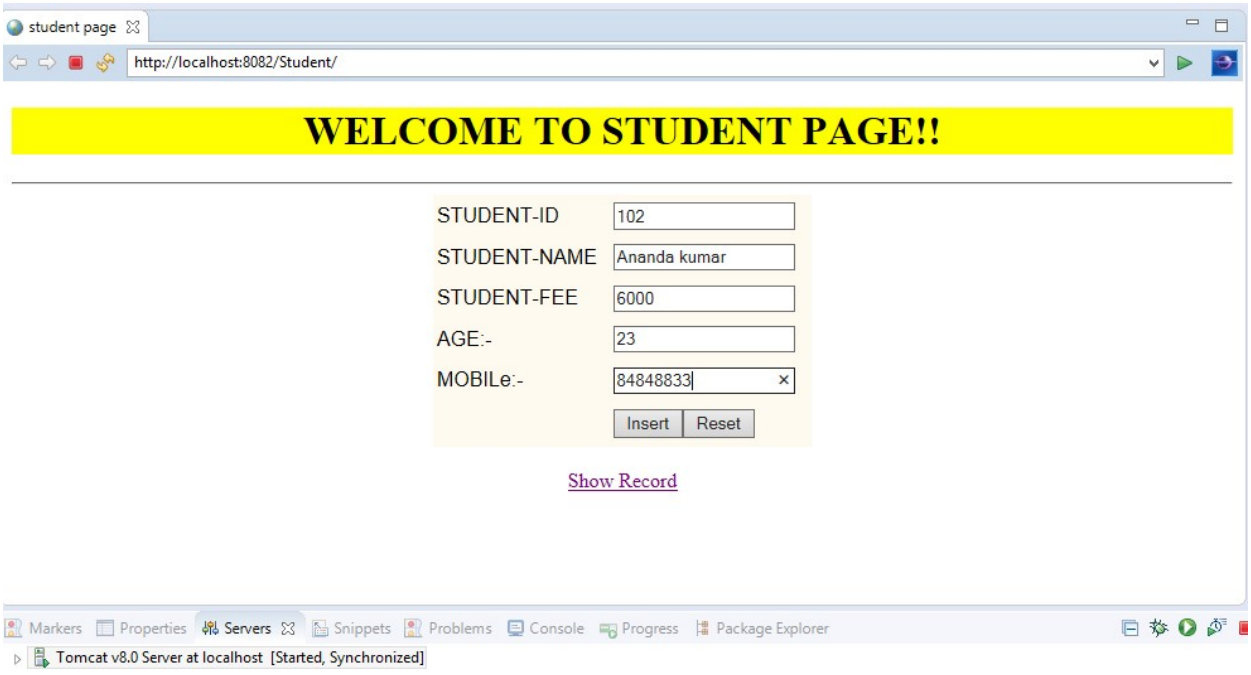
a. Right click on project name > Run As > Run On Server

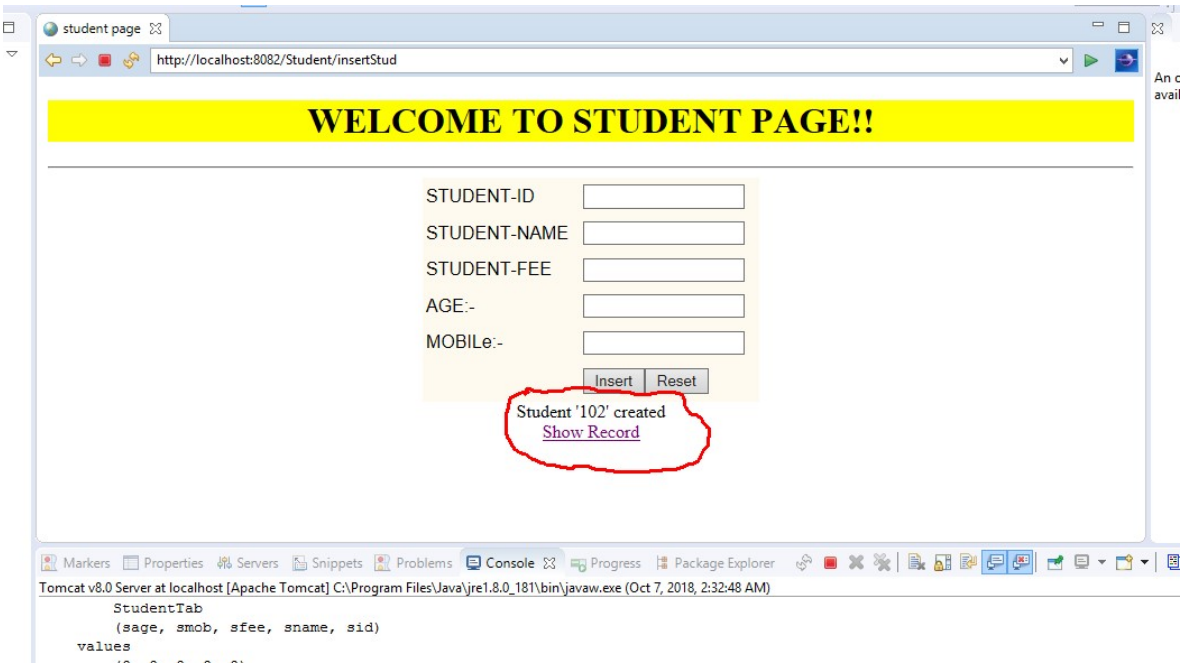
Output:-

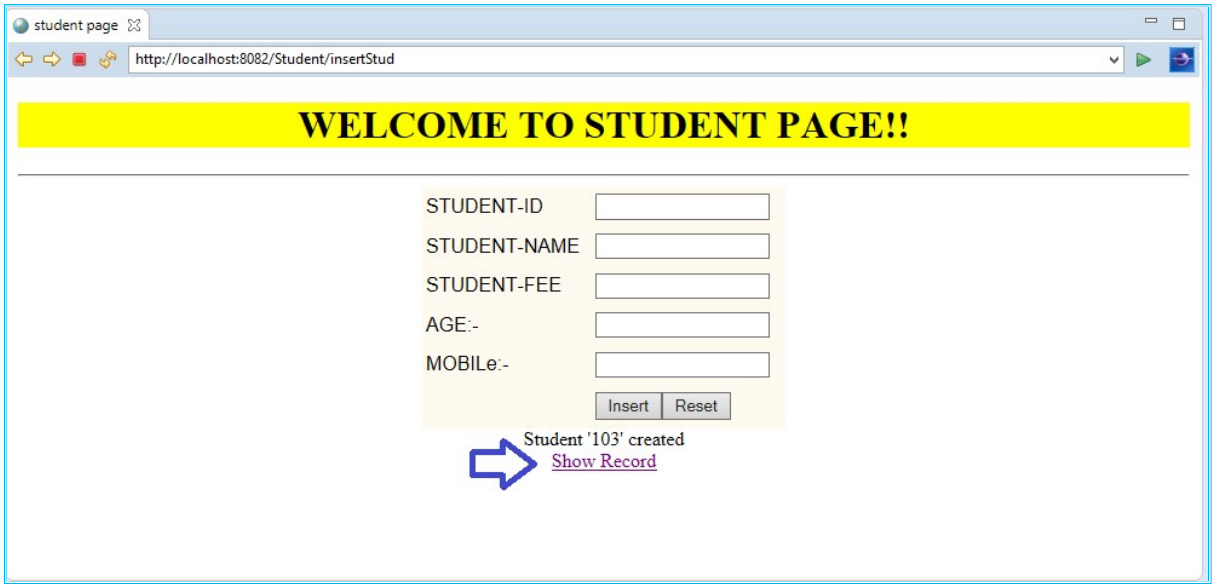
1. First

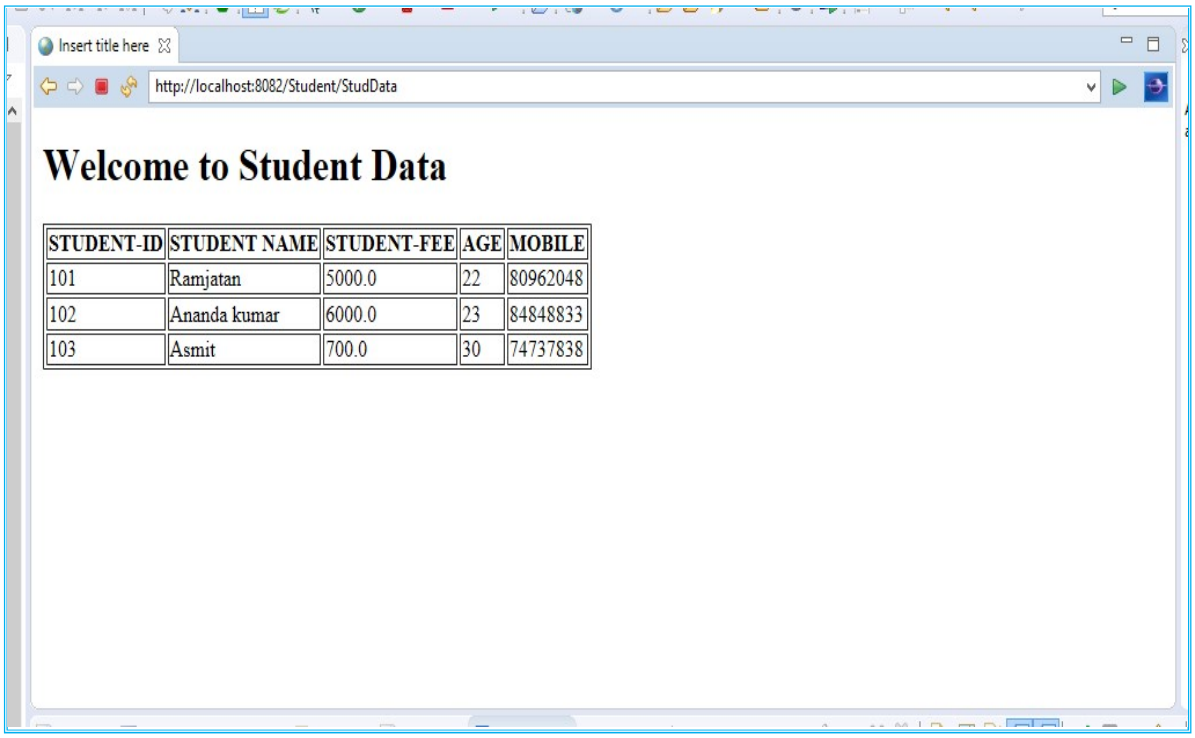


2. Second


student page
http://localhost:8082/Student/
WELCOME TO STUDENT PAGE!!
STUDENT-ID
102
STUDENT-NAME
Ananda kumar
STUDENT-FEE
6000
AGE:-
23
MOBILe:-
84848833
Insert
Reset
Show Record
Markers
Properties
Servers
Snippets
Problems
Console
Progress
Package Explorer
Tomcat v8.0 Server at localhost [Started, Synchronized]


student page
http://localhost:8082/Student/insertStud
WELCOME TO STUDENT PAGE!!
STUDENT-ID
STUDENT-NAME
STUDENT-FEE
AGE:-
MOBILe:-
Insert
Reset
Student '102' created
Show Record
Markers
Properties
Servers
Snippets
Problems
Console
Progress
Package Explorer
Tomcat v8.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Oct 7, 2018, 2:32:48 AM)
StudentTab
(sage, smob, sfee, sname, sid)
values


student page
http://localhost:8082/Student/insertStud
WELCOME TO STUDENT PAGE!!
STUDENT-ID
STUDENT-NAME
STUDENT-FEE
AGE:-
MOBILe:-
Insert
Reset
Student '103' created
Show Record


Insert title here
http://localhost:8082/Student/StudData
Welcome to Student Data
STUDENT-ID | STUDENT NAME | STUDENT-FEE | AGE | MOBILE
101 | Ramjatan | 5000.0 | 22 | 80962048
102 | Ananda kumar | 6000.0 | 23 | 84848833
103 | Asmit | 700.0 | 30 | 74737838