**Hibernate**

**Drawback of JDBC**

\* Java programmer should know how to write queries in SQL.  
\* Database dependent (earlier if we use MySQL and later switch to oracle then the queries have to be rewritten.)  
\* Handling the JDBC connections and properly closing the connection is also a big issue. Properly closing the connection is must. So, we needed something easier. Then frameworks came into picture.

**Why frameworks?**

The purpose of framework is to allow designers and developers to focus on building a unique feature for their web-based projects rather than re-inventing by coding. Framework is specially created to help you boost the performance and efficiency of your web app development task.

**What is framework?**

A framework is a structure that you can build software on. It serves as a foundation, so you're not starting entirely from scratch. Frameworks are typically associated with a specific programming language and are suited to different types of tasks.

**What is Hibernate Framework?**

Hibernate is a framework which provides some abstraction layer, meaning that the programmer does not have to worry about the implementations, hibernate does the implementations for us internally like establishing a connection with the database, writing query to perform CRUD operations etc.  
It is a java framework which is used to develop persistence logic.  
Persistence logic means to store and process the data for long use. More precisely Hibernate is an open-source, non-invasive, light-weight java ORM (Object-relational mapping) framework to develop objects which are independent of the database software and make independent persistence logic in all JAVA, JEE.  
Hibernate is an ORM Tool.

**What is ORM and how it works?**

Object-Relational Mapping (ORM) is a technique that lets you send query and manipulate data from a database using an object-oriented paradigm. When talking about ORM, most people are referring to a library that implements the Object-Relational Mapping technique, hence the phrase an ORM.  
An ORM library is a completely ordinary library written in your language of choice that encapsulates the code needed to manipulate the data, so you don't use SQL anymore; you interact directly with an object in the same language you're using.

**Explain how hibernate works?**

Hibernate is a framework which is used to develop persistence logic which is independent of Database software.  
In JDBC to develop persistence logic we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic which are independent of database software. In hibernate every class will be mapped to table and every object will be mapped to column or table data. To tell the program which class to be converted as table we have to use annotation the line before class declaration, we have to declare an annotation as @Entity which maps the class or creates the table for that class and object i.e., variable created will be considered as column.  
By initializing the objects, we can insert data into table.  
To declare any column as Primary key we will use @Id annotation before declaring that variable.  
In a Class we can have only one primary key or @Id.

**How to obtain connection to database?**

In hibernate we have EntityManagementFactory(), EntityManager(), and EntityTransaction().  
EntityManagerFactory will have createEntityManagerFactory() method of Persistence class which helps in establishing connection to database and will create table for @Entity class.  
EntityManager is used to perform crud operations on database. Every factory requires a manger.  
EntityTransaction is used whenever we are making changes to database or operation which will affect the database then we will use entity transaction.

**Methods we use in Hibernate**

First, we will use setters to set the values to object and we will use persist () method and pass the object as parameter which will save the object as table columns.  
Remove () method will delete the values based on object. Before deleting an object, we have to make sure that the object is present in the database, so for that we have find () method which will take class name and primary key as arguments and will return the object if it is present else it will return null.  
merge () is used to alter the table, it will also take object as the parameter.  
For executing queries which don’t have particular methods we will use CreateQuery("HQL") method.

**Program**

Project using switch of crud operation using Hibernate.  
To avoid confusion or to make a project easy to manage and navigate we will create three different packages as below:

**DTO**: This package should contain all the declaring part or Classes for which we are creating table.

**DAO**: This package should contain only the classes or methods which are required to communicate with the database. This part should not contain any printing statements.

**Controller**: This package contains the classes where it controls the flow and it has all the communicating statement between user and project, such as printing statement, taking input using scanner class etc.

**DTO Layer**

package user\_DTO;

import javax.persistence.Entity;  
import javax.persistence.Id;

@Entity  
public class User {  
@Id  
private int Id;  
private String Name;  
private String Email;  
private int Pwd;  
private long Phone;

public int getId() {

return Id;

}  
public void setId(int id) {  
 Id = id;  
}  
public String getName() {  
 return Name;  
}  
public void setName(String name) {

Name = name;

}

public String getEmail() {

return Email;

}

public void setEmail(String email) {

Email = email;

}

public int getPwd() {

return Pwd;

}

public void setPwd(int pwd) {

Pwd = pwd;

}

public long getPhone() {

return Phone;

}

public void setPhone(long phone) {

Phone = phone;

}

}

**DAO Layer**

package user\_DAO;

import java.util.List;

import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.EntityTransaction;  
import javax.persistence.Persistence;  
import javax.persistence.Query;  
import user\_DTO.User;

public class UserDAO {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

EntityTransaction transaction=manager.getTransaction();

public void saveuser(User user)

{

transaction.begin();

manager.persist(user);

transaction.commit();

}

public User getUserbyId(int id)

{

User u=manager.find(User.class, id);

return u;

}

public List<User> getAllUsers()

{

Query q=manager.createQuery("select u from User u",User.class);

List<User> lt=q.getResultList();

return lt;

}

public void updateUser(User u)

{

transaction.begin();

manager.merge(u);

transaction.commit();

}

public void deleteUserByID(User u)

{

transaction.begin();

manager.remove(u);

transaction.commit();

}

public List<User> getUserbyEMailPassword(String email,int pwd)

{

Query q=manager.createQuery("select u from User u where u.Email=?1 and u.Pwd=?2");

q.setParameter(1, email);

q.setParameter(2, pwd);

List<User> lt=q.getResultList();

return lt;

}

}

CRUD Operation

package controller;

import java.util.List;  
import java.util.Scanner;  
import user\_DAO.UserDAO;  
import user\_DTO.User;

public class AllinOne {

public static void main(String[] args) {

Scanner sc=new Scanner(System.in);

boolean b=true;

while(b)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Select one option:");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("1.Insert data \n2.Fetch user details by Id\n 3.Fetch all details \n4.Delete data \n5.Fetch details

by Email and Paswword\n6.Update details \n7.Exit");

int a=sc.nextInt();

switch(a){

case 1:

{

System.out.println("Enter ID:");

int id=sc.nextInt();

System.out.println("Enter Name:");

String name=sc.next();

System.out.println("Enter Email:");

String email=sc.next();

System.out.println("Enter Password (Numbers):");

int pwd=sc.nextInt();

System.out.println("Enter Mobile number:");

long phone=sc.nextLong();

//set values to object

User u =new User();

u.setId(id);

u.setEmail(email);

u.setName(name);

u.setPhone(phone);

u.setPwd(pwd);

//pass the object to DAO Layer

UserDAO dao=new UserDAO();

dao.saveuser(u);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Value Inserted...");

break;

}

case 2:

{

System.out.println("Enter ID:");

int id=sc.nextInt();

UserDAO dao=new UserDAO();

User u = dao.getUserbyId(id);

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(u.getId()+" "+u.getName()+" "+u.getEmail()+" "+u.getPwd()+" "+u.getPhone());

break;

}

case 3:

{

UserDAO dao=new UserDAO();

List<User> lt = dao.getAllUsers();

for(User u:lt)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(u.getId()+" "+u.getName()+" "+u.getEmail()+" "+u.getPwd()+" "+u.getPhone());

}

break;

}

case 4:

{

System.out.println("Enter ID:");

int id=sc.nextInt();

UserDAO dao=new UserDAO();

User u = dao.getUserbyId(id);

if(u!=null)

{

dao.deleteUserByID(u);

System.out.println("Data removed...");

}

else

{

System.out.println("ID not found");

}

break;

}

case 5:

{

System.out.println("Enter Email:");

String email=sc.next();

System.out.println("Enter Passowrd:");

int pwd=sc.nextInt();

UserDAO dao=new UserDAO();

List<User> lt = dao.getUserbyEMailPassword(email, pwd);

for(User u:lt)

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println(u.getId()+" "+u.getName()+" "+u.getEmail()+" "+u.getPwd()+" "+u.getPhone());

}

break;

}

case 6:

{

System.out.println("Enter ID:");

int id=sc.nextInt();

UserDAO dao=new UserDAO();

User u = dao.getUserbyId(id);

System.out.println("Enter column which needs to be changed Name/Email/Phone/Pwd");

String val=sc.next();

switch(val)

{

case "Name":{

System.out.println("Enter new Name:");

String name=sc.next();

u.setName(name);

dao.updateUser(u);

System.out.println("Values udated...");

break;

}

case "Email":{

System.out.println("Enter new Email:");

String email=sc.next();

u.setName(email);

dao.updateUser(u);

System.out.println("Values udated...");

break;

}

case "Phone":{

System.out.println("Enter new Phone Number:");

Long name=sc.nextLong();

u.setPhone(name);

dao.updateUser(u);

System.out.println("Values udated...");

break;

}

case "Pwd":{

System.out.println("Enter new Pwd:");

int pwd=sc.nextInt();

u.setPwd(pwd);

dao.updateUser(u);

System.out.println("Values udated...");

break;

}

default:{

System.out.println("Enter proper details:");

}

}

break;

}

case 7:

{

System.out.println("\*\*\*\*\*\*\*\*\*\*Thank You\*\*\*\*\*\*\*\*\*\*\*\*");

b=false;

break;

}

default:

{

System.out.println("Enter proper details:");

}

}

}

}

}

**Note:-**

To make a table name different from class name we use entity property name.  
i.e., @Entity(name="Employee\_Details")

If we want to specify name for column other than variable name, we use column annotation  
i.e., @Column(name="Employee\_details")

To make a column Not Null we have to use nullable property of annotation column  
i.e., @Column(nullable=true);

To make a column Unique we have to use unique property of annotation column  
i.e., @Column(unique=true);

To restrict the length, we use length property but we don’t usually use this as it is already done in the frontend.   
i.e., @Column(length=20)

**Mapping**

Hibernate mappings are one of the key features of hibernate, they help in establishing the relationship between two database tables as attributes in our model that allows us to easily navigate the associations in our model and criteria queries.  
we can establish either uni-directional or bi-directional i.e., we can either model them as an attribute on only one of the associated entities or on both. It will not impact your database mapping tables, but it defines in which direction you can use the relationship in your model and criteria queries.  
The relationship that can be established between entities are-

one to one — it represents the one-to-one relationship between two tables.

one to many/many to one — it represents the one-to-many relationship between two tables.

many to many — it represents the many to many relationships between two tables.

**One TO One example:**

One question will have one answer and similarly one answer will have one question so there will be One-to-one relation between them

Ex:

Question Class

package oTomapping;

import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.OneToOne;

@Entity

public class QuestionUni{

@Id

private int q\_id;

private String questions;

public int getQ\_id() {

return q\_id;

}

public void setQ\_id(int q\_id) {

this.q\_id = q\_id;

}

public String getQuestions() {

return questions;

}

public void setQuestions(String questions) {

this.questions = questions;

}

@OneToOne

AnswersUni a;

public AnswersUni getA() {

return a;

}

public void setA(AnswersUni a) {

this.a = a;

}

}

Answer Class

package oTomapping;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class AnswersUni {

@Id

private int a\_id;

private String answers;

public int getA\_id() {

return a\_id;

}

public void setA\_id(int a\_id) {

this.a\_id = a\_id;

}

public String getAnswers() {

return answers;

}

public void setAnswers(String answers) {

this.answers = answers;

}

}

Inserting Data

package oTomapping;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.EntityTransaction;

import javax.persistence.Persistence;

public class InsertingValueUni {

public static void main(String[] args) {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

EntityTransaction transaction=manager.getTransaction();

QuestionUni q=new QuestionUni();

q.setQ\_id(1);

q.setQuestions("What is Java?");

AnswersUni a=new AnswersUni();

a.setA\_id(1);

a.setAnswers("Java is an oop language");

q.setA(a);

transaction.begin();

manager.persist(q);

manager.persist(a);

transaction.commit();

System.out.println("Value inserted...");

}

}

Getting Value

package oTomapping;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

public class GettingValuesUni {

public static void main(String[] args) {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

QuestionUni q = manager.find(QuestionUni.class, 1);

if(q!=null)

{

System.out.println(q.getQ\_id()+"."+q.getQuestions());

AnswersUni a = q.getA();

System.out.println(a.getAnswers());

}

}

}

Question Bidirection

package oTomapping;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.OneToOne;

@Entity

public class QuestionBi{

@Id

private int q\_id;

private String questions;

public int getQ\_id() {

return q\_id;

}

public void setQ\_id(int q\_id) {

this.q\_id = q\_id;

}

public String getQuestions() {

return questions;

}

public void setQuestions(String questions) {

this.questions = questions;

}

@OneToOne

AnswersBi a;

public AnswersBi getA() {

return a;

}

public void setA(AnswersBi a) {

this.a = a;

}

}

Answer Bidirectional

package oTomapping;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.OneToOne;

@Entity

public class AnswersBi {

@Id

private int a\_id;

private String answers;

public int getA\_id() {

return a\_id;

}

public void setA\_id(int a\_id) {

this.a\_id = a\_id;

}

public String getAnswers() {

return answers;

}

public void setAnswers(String answers) {

this.answers = answers;

}

@OneToOne

QuestionBi q ;

public QuestionBi getQ() {

return q;

}

public void setQ(QuestionBi q) {

this.q = q;

}

}

Inserting Bidirection

package oTomapping;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.EntityTransaction;

import javax.persistence.Persistence;

public class InsertingValueBi {

public static void main(String[] args) {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

EntityTransaction transaction=manager.getTransaction();

QuestionBi q=new QuestionBi();

q.setQ\_id(2);

q.setQuestions("What is JDBC?");

AnswersBi a=new AnswersBi();

a.setA\_id(2);

a.setAnswers("Java DataBase Connectivity");

q.setA(a);

a.setQ(q);

transaction.begin();

manager.persist(q);

manager.persist(a);

transaction.commit();

System.out.println("Value inserted...");

}

}

Getting Bidirection

package oTomapping;

import javax.persistence.EntityManager;  
import javax.persistence.EntityManagerFactory;  
import javax.persistence.Persistence;

public class GettingValuesBi {

public static void main(String[] args) {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

QuestionBi q = manager.find(QuestionBi.class, 1);

if(q!=null)

{

System.out.println(q.getQ\_id()+"."+q.getQuestions());

}

AnswersBi a = q.getA();

System.out.println(a.getAnswers());

AnswersBi ans=manager.find(AnswersBi.class,2);

if(ans!=null)

{

System.out.println(ans.getA\_id()+"."+ans.getAnswers());

QuestionBi b = ans.getQ();

System.out.println(b.getQuestions());

}

}

}

**OnetoMany Example:**

One Hospital will have many branches so there will be one to many relationship

Hospital

package otmmapping;

import java.util.List;  
import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.OneToMany;

@Entity

public class Hospital {

@Id

private int h\_id;

private String h\_name;

private String owner\_name;

public int getH\_id() {

return h\_id;

}

public void setH\_id(int h\_id) {

this.h\_id = h\_id;

}

public String getH\_name() {

return h\_name;

}

public void setH\_name(String h\_name) {

this.h\_name = h\_name;

}

public String getOwner\_name() {

return owner\_name;

}

public void setOwner\_name(String owner\_name) {

this.owner\_name = owner\_name;

}

@OneToMany

List<Branch> b;

public List<Branch> getB() {

return b;

}

public void setB(List<Branch> b) {

this.b = b;

}

}

Branch

package otmmapping;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Branch {

@Id

private int b\_id;

private String b\_name;

public int getB\_id() {

return b\_id;

}

public void setB\_id(int b\_id) {

this.b\_id = b\_id;

}

public String getB\_name() {

return b\_name;

}

public void setB\_name(String b\_name) {

this.b\_name = b\_name;

}

}

Inserting Values

package otmmapping;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.EntityTransaction;

import javax.persistence.Persistence;

public class InsertingValues {

public static void main(String[] args) {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

EntityTransaction transaction=manager.getTransaction();

Hospital h=new Hospital();

h.setH\_id(1);

h.setH\_name("KIMS");

h.setOwner\_name("Kumarswamy");

Branch b1=new Branch();

b1.setB\_id(1);

b1.setB\_name("Bangalore");

Branch b2=new Branch();

b2.setB\_id(2);

b2.setB\_name("Pune");

List<Branch> b=new ArrayList<Branch>();

b.add(b1);

b.add(b2);

h.setB(b);

transaction.begin();

manager.persist(h);

manager.persist(b1);

manager.persist(b2);

transaction.commit();

}

}

Getting Values

package otmmapping;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

public class GettingValues {

public static void main(String[] args) {

EntityManagerFactory factory=Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

Hospital h = manager.find(Hospital.class, 1);

System.out.println(h.getH\_id()+"."+h.getH\_name()+"-"+h.getOwner\_name());

List<Branch> lt = h.getB();

for(Branch a:lt)

{

System.out.println(a.getB\_id()+"."+a.getB\_name());

}

}

}

ManyToOne Example:

Many branches of Qspiders come under Test Yantra Company so there will be many to one relationship between them

Qspider class

package mtomapping;

import javax.persistence.Entity;  
import javax.persistence.Id;  
import javax.persistence.ManyToOne;

@Entity

public class Qspider {

@Id

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@ManyToOne

TestYantra ty;

public TestYantra getTy() {

return ty;

}

public void setTy(TestYantra ty) {

this.ty = ty;

}

}

TestYantra Class

package mtomapping;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.ManyToOne;

@Entity

public class Qspider {

@Id

private int id;

private String name;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

@ManyToOne

TestYantra ty;

public TestYantra getTy() {

return ty;

}

public void setTy(TestYantra ty) {

this.ty = ty;

}

}

ManyToMany Example:

Many companies can have many projects so there will be manytomany relation between them

Company Class

package mtmmapping;

import java.util.List;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

@Entity

public class Company1 {

@Id

private int c\_id;

private String c\_name;

@ManyToMany

List<Project> p;

public int getC\_id() {

return c\_id;

}

public void setC\_id(int c\_id) {

this.c\_id = c\_id;

}

public String getC\_name() {

return c\_name;

}

public void setC\_name(String c\_name) {

this.c\_name = c\_name;

}

public List<Project> getP() {

return p;

}

public void setP(List<Project> p) {

this.p = p;

}

}

Projects Class

package mtmmapping;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Project {

@Id

private int p\_id;

private String p\_name;

public int getP\_id() {

return p\_id;

}

public void setP\_id(int p\_id) {

this.p\_id = p\_id;

}

public String getP\_name() {

return p\_name;

}

public void setP\_name(String p\_name) {

this.p\_name = p\_name;

}

}

Inserting Data

package mtmmapping;

import java.util.ArrayList;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.EntityTransaction;

import javax.persistence.Persistence;

public class InsertingMTM {

public static void main(String[] args) {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

EntityTransaction transaction = manager.getTransaction();

Company1 c1=new Company1();

c1.setC\_id(1);

c1.setC\_name("Infosys");

Company1 c2=new Company1();

c2.setC\_id(2);

c2.setC\_name("Cisco");

Project p1=new Project();

p1.setP\_id(1);

p1.setP\_name("WCMS");

Project p2=new Project();

p2.setP\_id(2);

p2.setP\_name("Library");

List<Project> p=new ArrayList<Project>();

p.add(p1);

p.add(p2);

c1.setP(p);

Project p3=new Project();

p3.setP\_id(3);

p3.setP\_name("Electricity");

Project p4=new Project();

p4.setP\_id(4);

p4.setP\_name("Attendence");

List<Project> q=new ArrayList<Project>();

q.add(p3);

q.add(p4);

c2.setP(q);

transaction.begin();

manager.persist(c1);

manager.persist(c2);

manager.persist(p1);

manager.persist(p2);

manager.persist(p3);

manager.persist(p4);

transaction.commit();

}

}

Fetching Data

package mtmmapping;

import java.util.List;

import javax.persistence.EntityManager;

import javax.persistence.EntityManagerFactory;

import javax.persistence.Persistence;

public class GetMTM {

public static void main(String[] args) {

EntityManagerFactory factory = Persistence.createEntityManagerFactory("dev");

EntityManager manager=factory.createEntityManager();

Company1 c1 = manager.find(Company1.class, 1);

System.out.println(c1.getC\_id()+"."+c1.getC\_name()+" Projects are:");

List<Project> lt1 = c1.getP();

for(Project l:lt1)

{

System.out.println(l.getP\_id()+"."+l.getP\_name());

}

Company1 c2 = manager.find(Company1.class, 2);

System.out.println(c2.getC\_id()+"."+c2.getC\_name()+" Projects are:");

List<Project> lt2 = c2.getP();

for(Project l:lt2)

{

System.out.println(l.getP\_id()+"."+l.getP\_name());

}

}

}

**Note:**

Caching-

For enabling 2nd level cache go to maven repository download hibernate ehcache version same as hibernate version, copy paste the dependency and in persistence.xml file add 2nd level caching properties.

Generated Value-

@GeneratedValue (strategy=GeneratedType.Identity) it is used for automatically generating Id.

Joined Column-

If we don’t want one extra column to get created or in the case of one to many or many to many if we don’t want a table to get created for the foreign key then we can use joined column property in the parent class and mapped by property in the child class and we can avoid the extra column or extra table.

Cascading –

It is used in the case of mapping, implementing this by making any changes to parent class it will affect the child class for example if we want to remove a table which has a child table linked to it, directly we cannot do that first we have to delete the child table then only we can delete parent table but if we use cascading property we can delete directly.