https://github.com/uysolanki/HibernateJars

D:\GL\Hibernate\dependecy downlaoded\hibernate-release-5.2.18.Final\lib\required

State of **Object** is a java application is Temporary 🡪Permanant storage , making the object persistent

We store the object state in a **Relational** DBMS by mapping the object to a database row

Invented by Gavin King in 2001

Hibernate is ORM -**Object Relationship Mapping**

Object in Java 🡪 Mapped to a row in a database table

Its Open source 🡪 Available for everyone without any cost

Its non invasive 🡪 it will not insist in extending a class or implementing an interface

Its light weight🡪 it does not contain additional functionalities; it uses only those functionalities required for object-relational mapping.

High-Performance🡪 supports many different fetching techniques such as caching, lazy initialization, and many more to achieve high performance.

Database Independent 🡪 databases such as Oracle, MySql, Sybase, etc (as it provides ‘Database Dialect -RDBMS Specific **MySQL5Dialect**,’ so we need not write SQL queries.

Traditional way(JDBC) --> Hibernate --> JPA Java Persistance API

Limitation of JDBC

1. Programmer had to register the db driver in the code

Class.forName(com.mysql.jdbc.Driver)

1. Create & open the connection by passing the db credentials

Connection con=DriverManager.getConnection("mysql://localhost../mydb", "root","")

1. Programmer had to Create the database tables

create table emp( eno int primary key....)

1. Write the SQL query and execute it

String query="insert into emp values(?,?,?)"

1. Manage the associations - 1:1, 1:M, M:M

Advantage of Hibernate

Builtin/readymade functions CRUD save, update,delete get/load

1. Basically Internally It is a wrapper on top of JDBC

We will only create an object and give it to hibernate and hibernate will do the remaining part of inserting in to the database.

Hibernate has given methods like save(), update()

The classname will be **mapped** to table name

The attributes will be **mapped** to the column name

The mapping will be done by the help of various **annotations**

**@Entity**

**class Student{**

**@ID**

**@GeneratedValue staregy="Identity" as good using auto\_increment**

**int rno**

**}**

**@Overrides**

-----

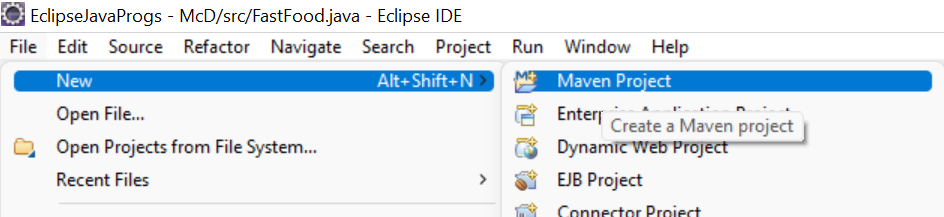
Session factory is an interface so internally it creates object of the SessionFactoryImpln class.

Session factory will be needed to create a session instance,

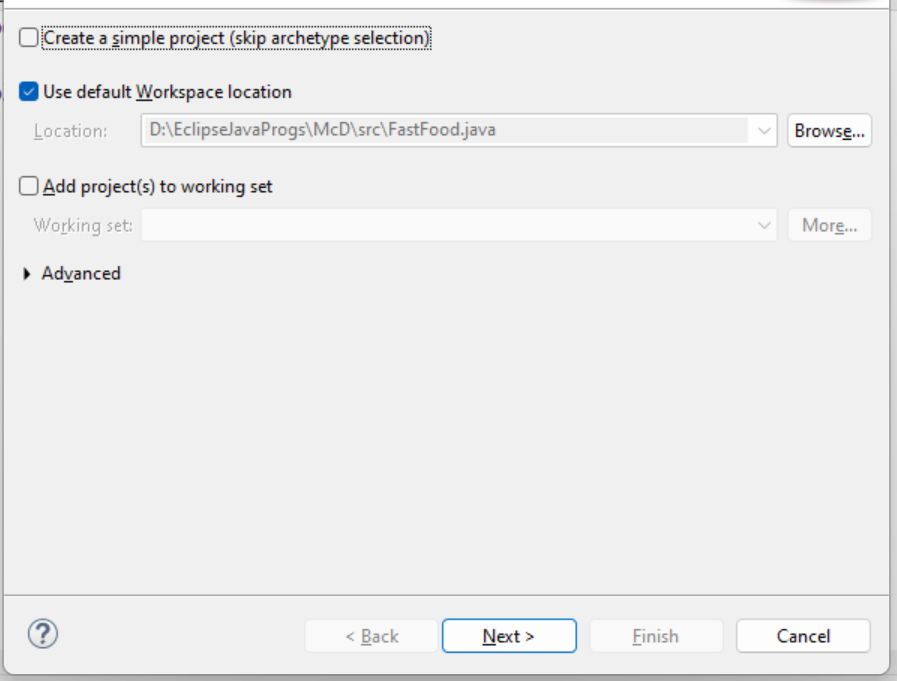
Session (like connection in jdbc) is the API which will give us utility methods to connect & perform the database related CRUD activities

How to create a Hibernate Project in maven

Step 1 :

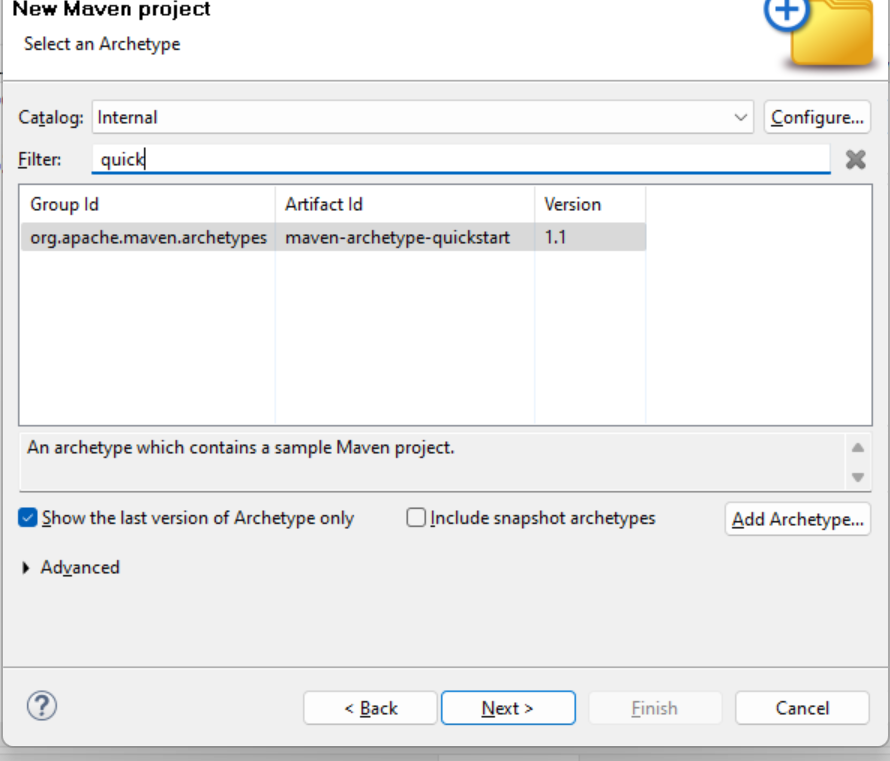


Step 2: Click Next



Step 3 : Catalog : choose Internal

Filter : type webapp , select the Archtye and click next



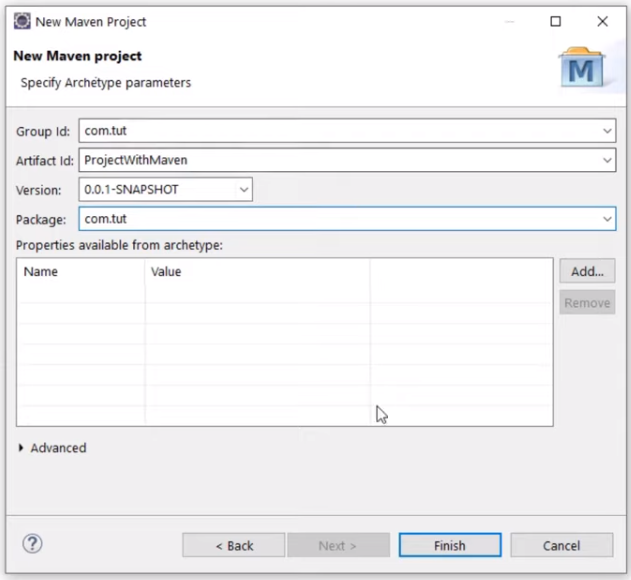
Step 4 :

Group ID : Base package name com.companyname

Artifact ID: Project Name : HIbernateWithMaven

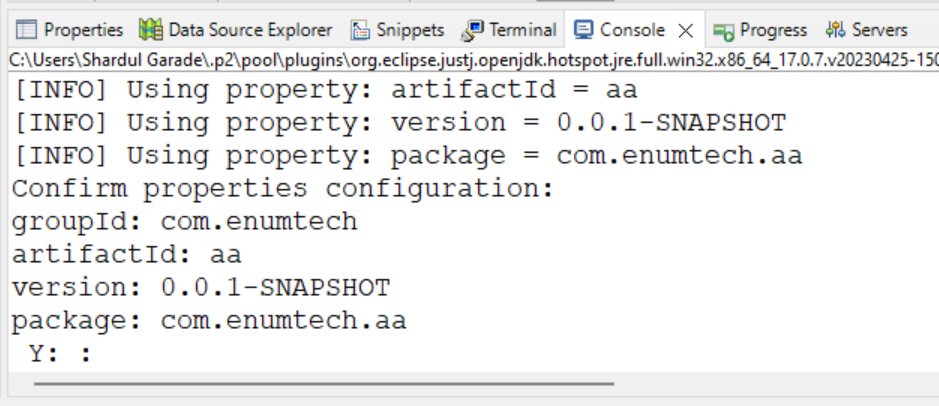
Version : 0.0.1 SNAPSHOT (default)

Package name : (Group ID concat with ArtifactID, but we can keep it same as GroupID) com.companyname



Step 5 : If the project creations is taking very log or is stuck at 33% go to console tab, scroll till the end

On Console it will ask you to enter capital **Y**

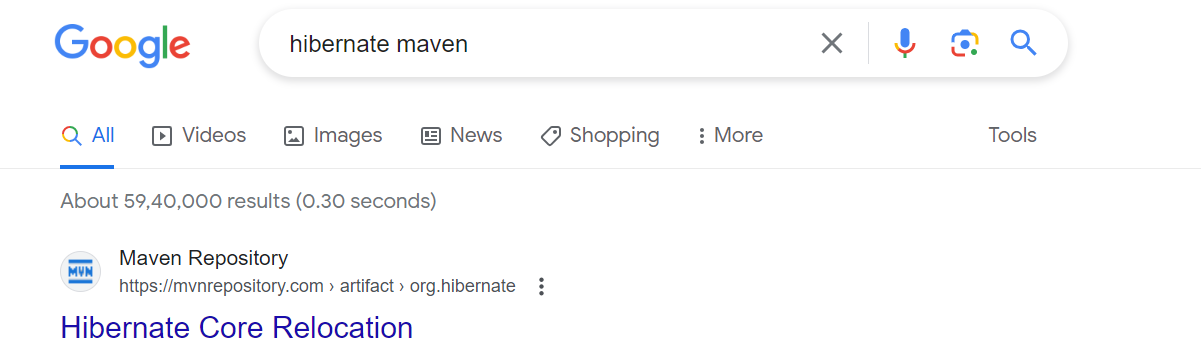
****

**Y**

**Add the dependencies**

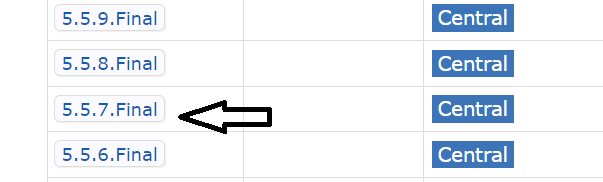
1. **Add the hibernate maven dependency**

**Go to google :**

****

**Go to mvnrepoisotry website**

**5.4.5 version**

****

**Copy the dependency and paste in pom.xml**

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.4.5.Final</version>

<type>pom</type>

</dependency>

</dependencies>

1. **Add the hibernate mysql connector dependency**

**Mysql connector maven**

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java </artifactId>

<version>5.1.15</version>

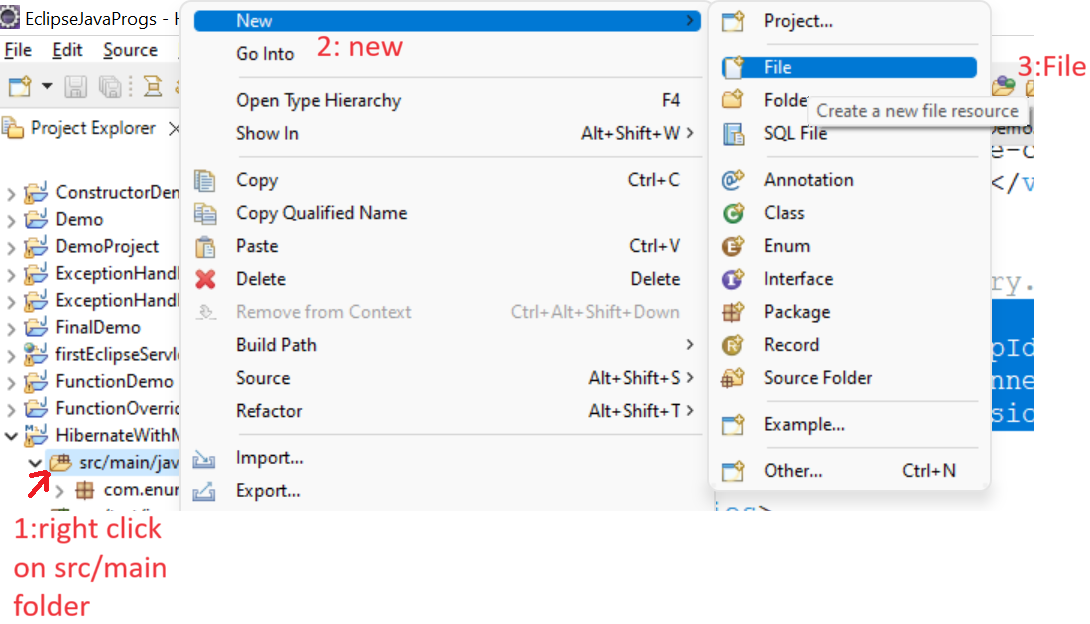
</dependency>

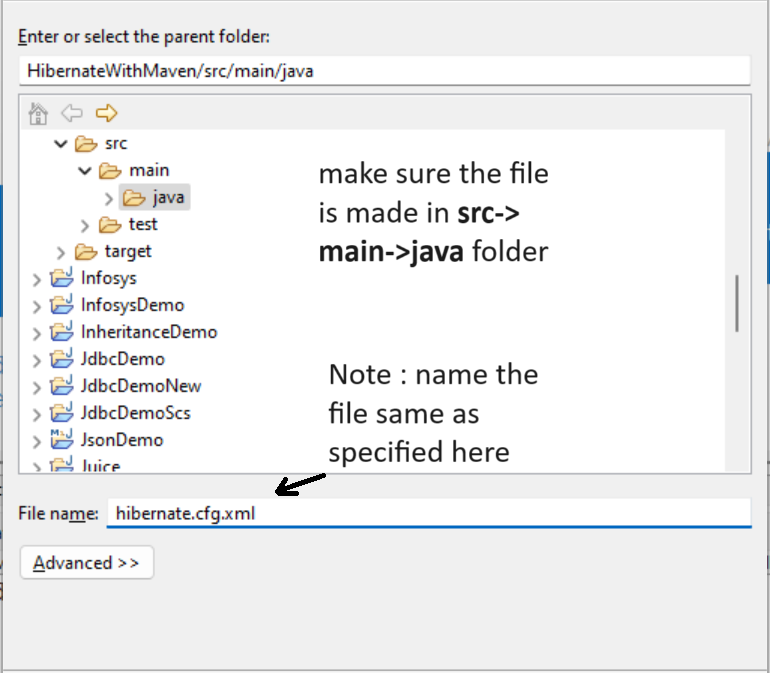
**In Maven depecdencies look for the “hibernate-core- “ jar file if not visible add it externally**

**\_\_\_\_**

**Part 3:**

**Configuring hibernate.cfg.xml file**

****

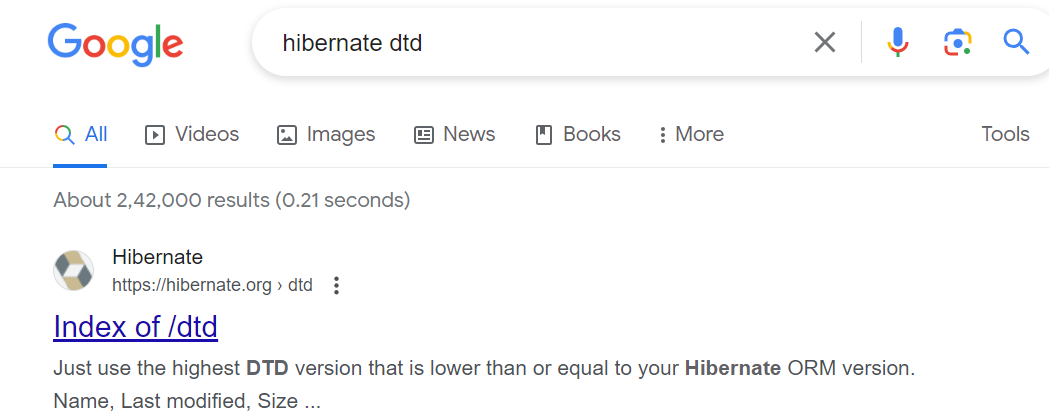
****

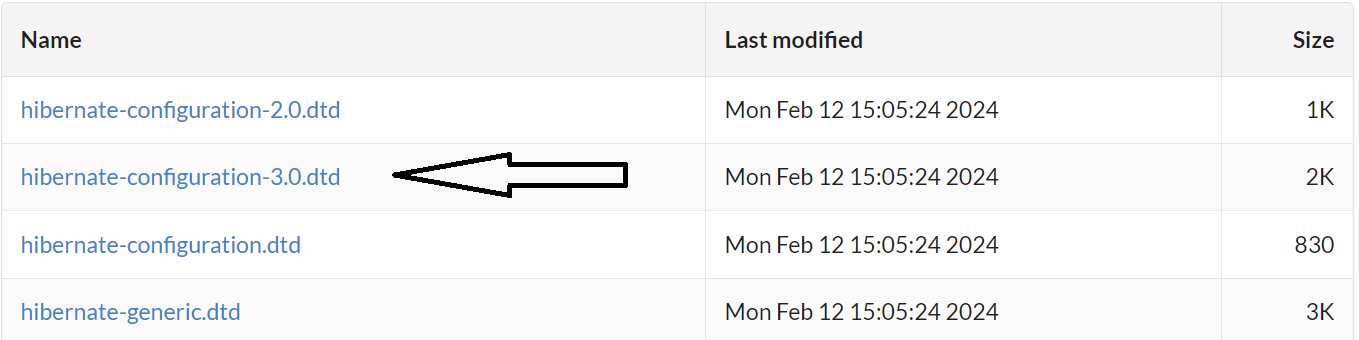
**Add As the first line of the xml file**

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

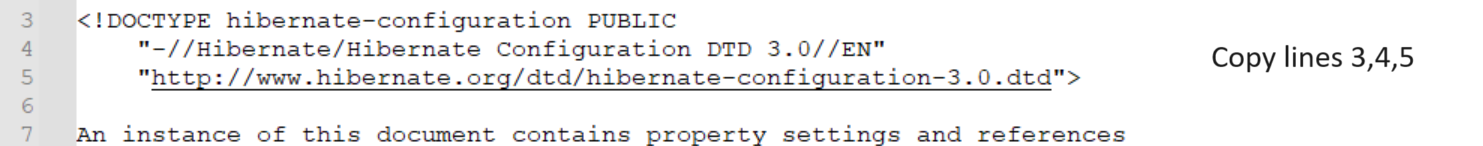
**Need DTD**

**Google-hibernate dtd**

****

****

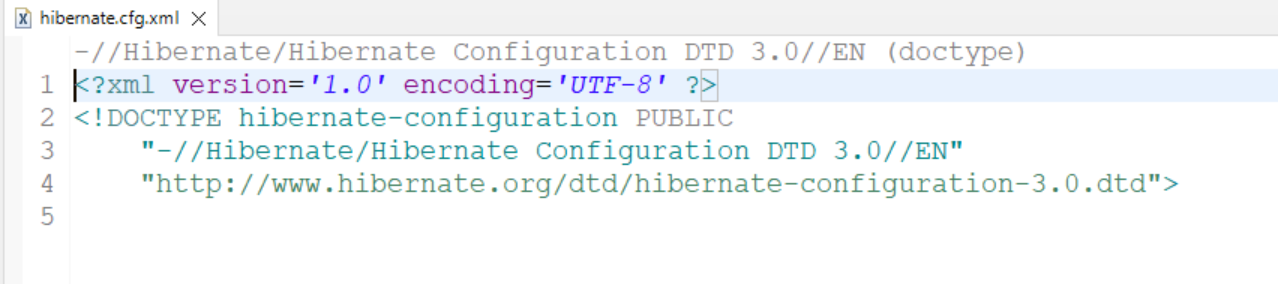
**Open file in notepad**

****

<!DOCTYPE hibernate-configuration PUBLIC

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

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

****

**Do the following config**

<hibernate-configuration>

<session-factory>

<property name=*"connection.driver\_class"*>com.mysql.jdbc.Driver</property>

<property name=*"connection.url"*>jdbc:mysql://localhost:3306/glnewhibernate </property>

<property name=*"connection.username"*>root</property>

<property name=*"connection.passowrd"*></property>

<property name=*"dialect"*>org.hibernate.dialect.MySQL5Dialect</property>

<property name=*"hbm2ddl.auto"*>update</property>

<property name=*"show\_sql"*>true</property>

<property name=*"format\_sql"*>true</property>

</session-factory>

</hibernate-configuration>

<mapping class=*"com.enumtech.entity.Student"*></mapping>

**Create database named “**glnewhibernate**” in mysql**

**create database** glnewhibernate;

**To test whether the project is working**

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

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

SessionFactory factory=cfg.buildSessionFactory();

/\* this factory object will be used to execute the database functions\*/

System.***out***.println(factory);

System.***out***.println(factory.isClosed());

**NOTE : If you place the hibernate.cfg.xml in a different folder/package then you will have to give the whole path of that file**

**e.g if you placed the file in com.gl package then we have to give**

**cfg.configure("com/gl/hibernate.cfg.xml");**

**----**

**Part 4**

**Make an entity class in the com.enumtech.entity package**

**package** com.enumtech.entity;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

@Entity

**public** **class** Student {

@Id

**private** **int** rno;

**private** String sname;

**private** String scity;

**public** Student() {}

**public** Student(**int** rno, String sname, String scity) {

**this**.rno = rno;

**this**.sname = sname;

**this**.scity = scity;

}

**public** **int** getRno() {

**return** rno;

}

**public** **void** setRno(**int** rno) {

**this**.rno = rno;

}

**public** String getSname() {

**return** sname;

}

**public** **void** setSname(String sname) {

**this**.sname = sname;

}

**public** String getScity() {

**return** scity;

}

**public** **void** setScity(String scity) {

**this**.scity = scity;

}

@Override

**public** String toString() {

**return** "Student [rno=" + rno + ", sname=" + sname + ", scity=" + scity + "]";

}

}

**In main method**

Student s1=**new** Student(101,"Alice","Jaipur");

Session session=factory.openSession();

Transaction tx=session.beginTransaction();

session.save(s1);

tx.commit();

session.close();

**Part 5**

**most commonly used annotations**

**@Entity**

**@Table**

**@Id @GeneratedValue =identity**

**@column**

**@Transient -> don’t male this column in database**

**@Temporal ->format the date to remove the time**

**@Lob ->binay object**

**Mapping @ OneToOne, @ OneToMany, @ManyToMany**

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

**private** **int** rno;

@Column(length=20)

**private** String sname;

**private** String scity;

@Transient

**private** **int** x;

@Temporal(TemporalType.***DATE***)

**private** Date dob;

@Lob

**private** **byte** image[];

Part 8

get

get()returns NULL if the object is not found in db & cache

use if you are not sure

get() will first look in the session cache (shop), if the object is found in cache, it will not hit the db(godown).

Student s1=(Student)session.get(Student.**class**, 101);

System.***out***.println(s1);

// get will first try to search for the 101 object in session cache,if not found it will query(“select”) the database [sign of which is SQL query will be printed on the console] and fetch the object and store it in the session cache.

Student s2=(Student)session.get(Student.**class**, 101);

System.***out***.println(s2);

// when subsequent get request is sent for the same 101 object , it will first search in session cache,if found it will not query(“select”) the database [hence no SQL query will be printed on the console] the return the existing 101 object from the session cache.

To confirm if we do a

System.***out***.println(s1==s2);

It will retun “True”

Student s5=(Student)session.get(Student.**class**, 177);

If this 177 object is not there it will retun null

Sysout(s5) will display null

load

load() throws ObjectNotFoundException but will not return NULL

use if you are 100% sure

load() returns a proxy object, lazy initialization

but will hit the db only when any function of that object is invoked, like toString/getter()[except getID].

Student s1=(Student)session.load(Student.**class**, 101);

Here it will fetch create a proxy/dummy object and return its reference[ the db will not queried [no sql query will be displayed on console] as of now, this is called lazy instantiation],

If this 101 object is not there it will throw ObjectNotFoundException

System.***out***.println(s1);

When we invoke any method(toString/getter) on the object, that time it will query the database [the sql query will now be displayed on console]

**[like while using scanner, it does not import the scanner class until u invoke any method using that object]**

Part 9

@Embeddable

**public** **class** Address {..}

public class Student

{

**Private int rno;**

**private** String sname;

**private** String scity;

**private** Address address;

}