***Hibernate***

* *"Persist" means "lives on after the application is shut down" (Persistence simply means to Store Permanently.)*
* *Fetch Data (get the data from database)*
* *ORM – Object Relational Mapping*

***Pro’s*** *- can execute the more than one queries at one database*

*instead of writing queries sending the objects to process the query.3 components in hibernate*

* *Connection management.*
* *Transaction management.*
* *Object relational mapping.*

*Connection management:*

* *maintain the connection is li8 difficult so better to use 3rd party tools like c3p0 or dbcp(provided by*
* *apache).*

*Transaction management:*

* *in normal jdbc you can execute only 1 database statement*
* *in hibernate you can execute more than 1 database*

*ORM:*

* *mapping between relational db and oops*
* *create a sessions to make use(session.save, session.update)*
* *pass the object to session, so session internally parse into query and map the result.*
* *in hibernate mapping is very clear its easy to understand what value mapping to what*

*Hibernate has two configurations files*

* *customer.hbm.xml*
* *hibernate..cfg.xml*

***Main Steps***

* *Configure hibernate*

Configuration con = new Configuration().configure().addAnnotationClass(Alien.class)

* *Open the session*

Session session = SessionFactory.openSession()

* *Transaction*

Transaction tx = session.beginTransaction();

***Setup***

* *Hibernate Dependency libraries*

<dependency>

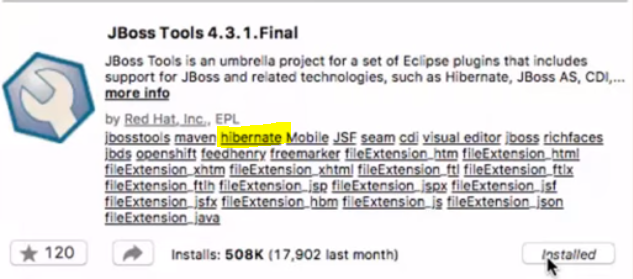
<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

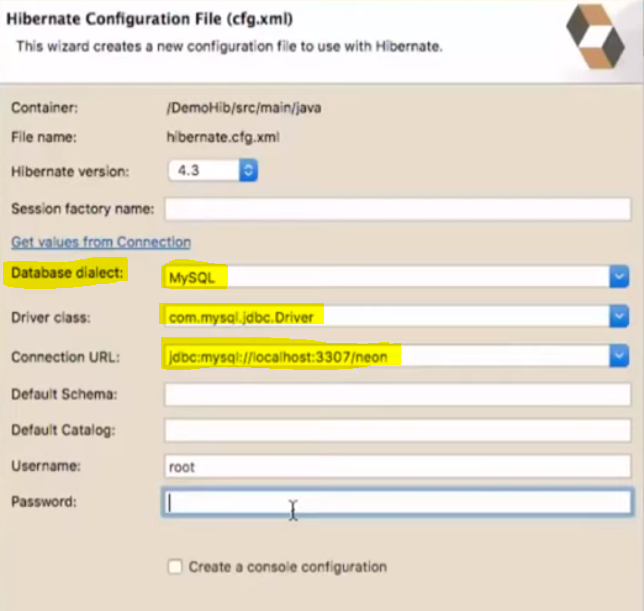
<version>5.3.1.Final</version>

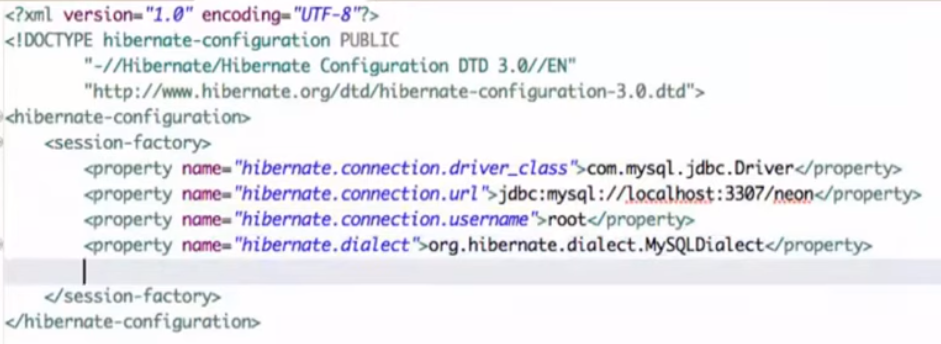
</dependency>

* *Add hibernate plugin to eclipse*
  + *Add JBoss tools*



* *Configuration*
  + *Eclipse -> right click on project -> new (Others) -> hibernate -> select hibernate.cfg.xml*

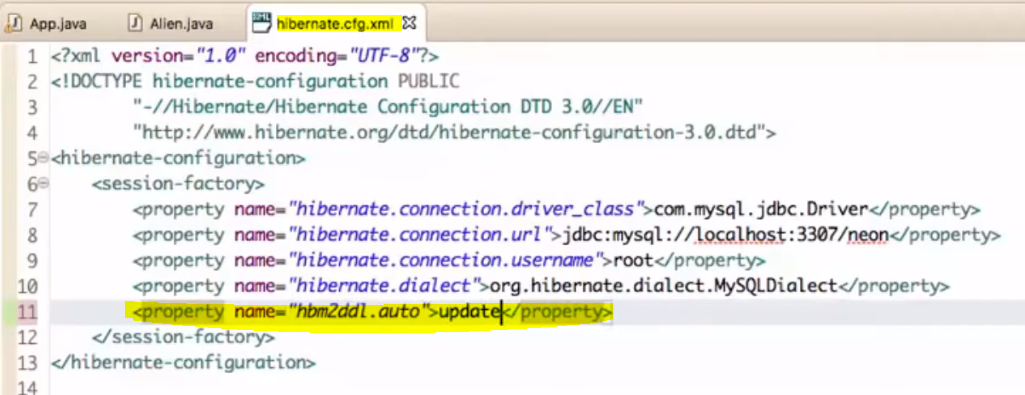




* *If you are making changes in Database using hibernate you should follow ACID properties*
  + *Atomicity*
  + *Consistency*
  + *Isolation*
  + *Durability*

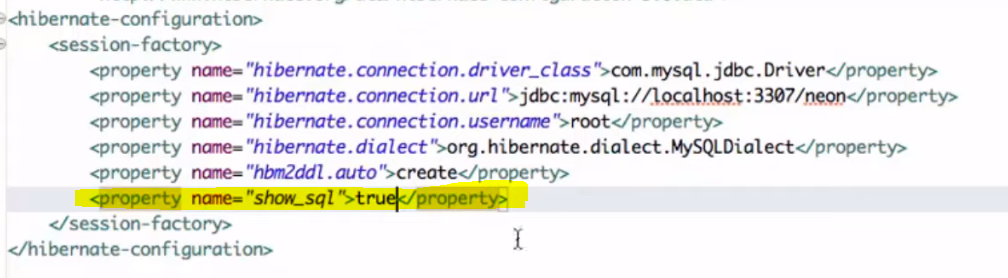
***Create a table using Hibernate***

In hibernate.cfg.xml



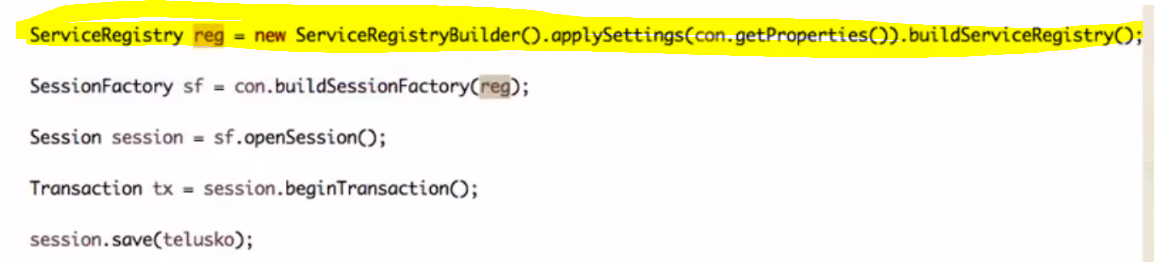
*Update will update the table if no table new table created without losing data*

***To see SQL query***

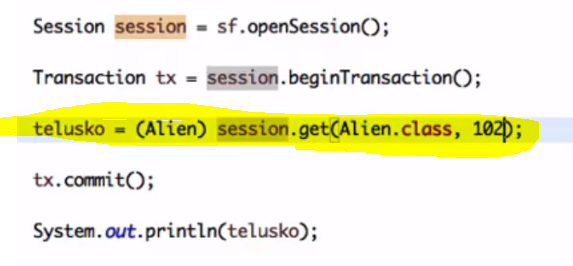


***Service Registry***

serviceRegistry reg = new ServiceRegistryBuilder(). applySettings(con.getProperties()).buildServiceRegistry();



***Fetch data***



*StudentObj=( StudentObj)Session.get(class, id(primaryKey) )*

***To persist data from objects inside object***

* *if pojo has one multiple objects --> while creating a table using hibernate will conflict with column rows*
* *A student pojo with multiple fields**including laptop field and laptop has multiple fields.*
* *If you want to persist laptop fields into DB table use @embadable*
* *if you use @Entity it will create new table*

***Hibernate Configuration***

*<property name="show\_sql">true</property> 🡪 can see what query executing*

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

*<property name="hibernate.hbm2ddl.auto">create</property> 🡪 create new table every time*

*Scenario 1*

|  |  |
| --- | --- |
| *@Entity*  *Public class Laptop*  *{*  *@Id*  *Private int lid;*  *Private String lname;*  *//Many to one*  *@ManyToOne*  *Private student std;*  *//Many to Many*  *@ManyToMany*  *Private List<Student> std*  *}* | *@Entity*  *Public class Student*  *{*  *@Id*  *Private int rollNo;*  *Private String name;*  *Private int marks;*  *//Many to one*  *@OneToMany(mappedBy =’sid’)*  *Private List<Laptop> laptops*  *//Many to Many*  *@ManyToMany(mappedBy =’sid’)*  *Private List<Laptop> laptops*  *}* |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Laptop*   |  |  |  | | --- | --- | --- | | *Lid* | *Lname* | *RollNo* | | *101* | *Dell* | *1* | | *102* | *HP* | *1* | | *103* | *MacBook* | *2* | | *Student*   |  |  |  | | --- | --- | --- | | *RollNo* | *Name* | *Marks* | | *1* | *Sandeep* | *50* | | *2* | *Kalyani* | *75* | | *3* | *Aarvi* | *95* | |

***Fetch # Eager, Lazy(Default)***

*Hibernate has 2 types of fetching*

* *Eager:  is a design pattern in which data initialization occurs on the spot*
  + @OneToOne(fetch=FetchType.EAGER)
* *Lazy(Default):  is a design pattern which is used to defer initialization of an object if it’s possible*

***Hibernate Caching***

* *Hibernate has 2 levels of cache* 
  + *Level 1 cache is user independent*
  + *Level 2 cache will have shared data. it’s not default available*
    - *To use level 2 cache, you need to configure using 3rd party dependency*
    - *Most used 3rd party tools: eh-cache, os, swam*
    - *Dependency*
      * *<!--provides all features-->*
      * *<dependency>*

*<groupId>org.ehcache</groupId>*

*<artifactId>ehcache</artifactId>*

*<version>3.4.0</version>*

*</dependency>*

*<!—this for integration to hibernate -->*

* + - * *<dependency>*

*<groupId>org.hibernate</groupId>*

*<artifactId>hibernate-ehcache</artifactId>*

*<version>5.3.10.Final</version>*

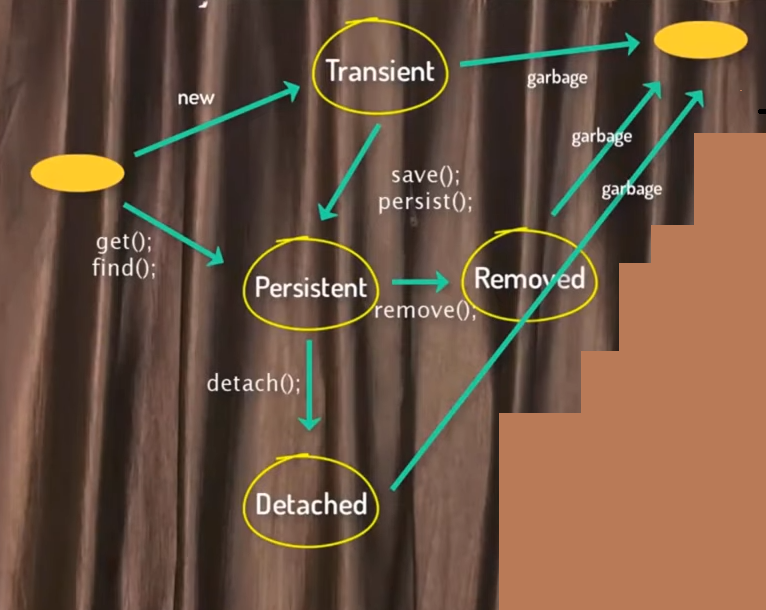
*</dependency>*

* + *Add config in H.C.X*
  + *And add annotation at @Entity level*
    - *@Cacheable // it makes this is available for caching*
    - *@Cache // to implement cache strategy*

***Hibernate Query Language (HQL)***

|  |  |
| --- | --- |
|  |  |
| Select \* from student | From Student |
|  |  |

***Hibernate Object States/Persistence life cycle***

******

***Difference between get() and load()***

* + - * Get()
        + will always hit the database. Though you not using the object
        + get will pass original object
        + exception: if no data available to fetch it return null or NullPointerException
      * Load()
        + if you not use object load won’t hit the database
        + load will pass proxy object
        + exception: if no data available to fetch it return ObjectNotFoundException

***what is JPA & JPA Implementation***

***Annotations***

* *@Entity – @Entity declares the class as an entity (i.e. a persistent POJO class), @Id declares the identifier property of this entity. The other mapping declarations are implicit. The class Flight is mapped to the Flight table, using the column id as its primary key column.*
  + *@Entity (name= “EntityName”)*
* *@Id – it makes field as primary key*
* *@Table (name = “TableName”) 🡪 it will change only table name not entity name*
* *@Column (name = “yourOwnColumnName”) 🡪 it change the column name*
* *@Transient 🡪 these field will not store in DB*
* *@OneToMany*

*Private List<Laptops> laptops*

***Questions***

***Q. What are the benefits of detached objects?***

*Answer:*

*Detached objects can be passed across layers all the way up to the presentation layer without having to use any DTOs (Data Transfer Objects). You can later re-attach the detached objects to another session.*

***Q. What are the pros and cons of detached objects?***

*Answer:*

***Pros:***

*"When long transactions are required due to user think-time, it is the best practice to break the long transaction up into two or more transactions. You can use detached objects from the first transaction to carry data all the way up to the presentation layer. These detached objects get modified outside a transaction and later on re-attached to a new transaction via another session.*

***Cons:***

*"In general, working with detached objects is quite cumbersome, and better to not clutter up the session with them if possible. It is better to discard them and re-fetch them on subsequent requests. This approach is not only more portable but also more efficient because - the objects hang around in Hibernate's cache anyway.*

*" Also from pure rich domain driven design perspective it is recommended to use DTOs (DataTransferObjects) and DOs (DomainObjects) to maintain the separation between Service and UI tiers.*

***12. What are the benefits of using Hibernate template?***

*Following are some key benefits of using Hibernate template:*

*a. Session closing is automated.*

*b. Interaction with hibernate session is simplified.*

*c. Exception handling is automated.*

*hibernate.cfg.xml*

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

*<!DOCTYPE hibernate-configuration PUBLIC*

*"-//Hibernate/Hibernate Configuration DTD//EN"*

*"http://hibernate.sourceforge.net/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/hibtutorial</property>*

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

*<property name="hibernate.connection.password">sandeep</property>*

*<property name="hibernate.connection.pool\_size">10</property>*

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

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

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

*<mapping resource="customer.hbm.xml" />*

*</session-factory>*

*</hibernate-configuration>*

***Code***

App.java

|  |
| --- |
| **package** com.sandeep.hiber;  **import** java.util.List;  **import** java.util.Map;  **import** java.util.Random;  **import** javax.persistence.EntityManager;  **import** javax.persistence.EntityManagerFactory;  **import** javax.persistence.Persistence;  **import** org.hibernate.Criteria;  **import** org.hibernate.SQLQuery;  **import** org.hibernate.Session;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;  **import** org.hibernate.cfg.Configuration;  **import** org.hibernate.query.Query;  **import** org.hibernate.service.ServiceRegistry;  **public** **class** App  {  **public** **static** **void** main( String[] args )  {  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JPA\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/    Student aarvi = **new** Student();  aarvi.setName("Aarvi");  aarvi.setRollNo(1243);  aarvi.setGroup("CSE");  aarvi.setMarks(900);  EntityManagerFactory emf = Persistence.createEntityManagerFactory("my-persistence-unit");  EntityManager em = emf.createEntityManager();    em.getTransaction().begin();  em.persist(aarvi);  em.getTransaction().commit();    System.***out***.println(aarvi);      /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JPA\*END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  System.***out***.println( "Hello World!" );    Laptop laptopDell = **new** Laptop();  laptopDell.setLid(101);  laptopDell.setLname("Dell");    Student sandeep = **new** Student();  sandeep.setName("Sandeep");  sandeep.setRollNo(1243);  sandeep.setGroup("CSE");  sandeep.setMarks(900);  sandeep.setLaptop(laptopDell);    //many to many relation  laptopDell.getStudentList().add(sandeep);    Configuration config = **new** Configuration().configure().addAnnotatedClass(Student.**class**).addAnnotatedClass(Laptop.**class**);  ServiceRegistry registry = **new** ServiceRegistryBuilder().applySetting(config.getProperties()).buildServiceRegistry();  SessionFactory sf = config.buildSessionFactory(registry);    Session session1 = sf.openSession();  session1.beginTransaction();    /\*//insert 50 entries to DB  \* Random random = new Random();    for(int i=1; i<=50; i++) {  Student s = new Student();  s.setRollNo(i);  s.setName("Name" + i);  s.setMarks(random.nextInt(100));  }  \*/    // print all the values from DB  /\* Query query = session1.createQuery("from Student");  List<Student> students = query.list();    for (Student s: students) {  System.out.println(s);  }\*/    // Students who have marks more then 50  /\* Query query = session1.createQuery("from Student where rollno > 50");  List<Student> students = query.list();    for (Student s: students) {  System.out.println(s);  }  \*/  // get student info of roll number =7    /\*Query query = session1.createQuery("from Student where rollno =7");  Student students = (Student) query.uniqueResult();\*/    // get specific columns in table    Query query = session1.createQuery("select rollno, name, marks from Student where rollno > 50 ");  // Query query = session1.createQuery("select rollno, name, marks from Student s where s.rollno > 50 "); // Both are same  List<Object[]> students = (List<Object[]>) query.list();    **for**(Object[] student : students) {  System.***out***.println(student);  }    // get sum of marks who has marks more than 50  /\* int b = 50;  Query sumQuery = session1.createQuery("select sum(marks) from Student where rollno > :b "); // sum of marks  sumQuery.setParameter("b", b);  Long marks = (Long) query.uniqueResult();    System.out.println(marks);\*/  // use SQL Query  /\* SQLQuery sqlQuery = session1.createSQLQuery("select \* from Student where marks > 60 ");  sqlQuery.addEntity(Student.class);  List<Student> studentList = sqlQuery.list();  for (Student s: studentList) {  System.out.println(studentList);  }\*/    // use SQL Query(Native Query) specific rows  SQLQuery sqlQuery = session1.createSQLQuery("select name, marks from Student where marks > 60 ");  sqlQuery.setResultTransformer(Criteria.ALIAS\_TO\_ENTITY\_MAP);  List<Student> studentList = sqlQuery.list();  **for** (Object object: studentList) {  Map map = (Map) object;  System.***out***.println(map.get("name")+ ":" + map.get("marks"));  }      /\* Query q1 = session1.createQuery("from table where id =101");  q1.setCacheable(true); // it will cache the query  sandeep =(Student)q1.uniqueResult();      session1.save(laptopDell);  session1.save(sandeep);  session1.getTransaction().commit();  session1.close();\*/      /\* Session session2 = sf.openSession();  session2.beginTransaction();    Query q2 = session2.createQuery("from table where id =101");  q2.setCacheable(true); // this it will use from cache  sandeep =(Student)q2.uniqueResult();    session2.save(laptopDell);  session2.save(sandeep);  session2.getTransaction().commit();  session2.close();  \*/    }  } |

Laptop.java

|  |
| --- |
| **package** com.sandeep.hiber;  **import** java.util.ArrayList;  **import** java.util.List;  **import** javax.persistence.Cacheable;  **import** javax.persistence.Entity;  **import** javax.persistence.Id;  **import** javax.persistence.ManyToMany;  **import** javax.persistence.ManyToOne;  **import** org.hibernate.annotations.Cache;  **import** org.hibernate.annotations.CacheConcurrencyStrategy;  @Entity  @Table(name = "Laptop\_Table")  @Cacheable  @Cache(usage=CacheConcurrencyStrategy.READ\_ONLY)  **public** **class** Laptop {    @Id  **private** **int** lid;  **private** String lname;    @ManyToOne // it avoid create Student\_Laptop mapping table and write (mappedBy="student") in Student class  **private** Student student;    @ManyToMany  **private** List<Student> studentList = **new** ArrayList<Student>();    **public** **int** getLid() {  **return** lid;  }  **public** **void** setLid(**int** lid) {  **this**.lid = lid;  }  **public** String getLname() {  **return** lname;  }  **public** **void** setLname(String lname) {  **this**.lname = lname;  }  **public** Student getStudent() {  **return** student;  }  **public** **void** setStudent(Student student) {  **this**.student = student;  }  **public** List<Student> getStudentList() {  **return** studentList;  }  **public** **void** setStudentList(List<Student> studentList) {  **this**.studentList = studentList;  }    } |

Student.java

|  |
| --- |
| **package** com.sandeep.hiber;  **import** java.util.ArrayList;  **import** java.util.List;  **import** javax.persistence.Id;  **import** javax.persistence.ManyToMany;  **import** javax.persistence.OneToMany;  **import** javax.persistence.OneToOne;  **public** **class** Student {    @Id //primary key  **private** **int** rollNo;  **private** String group;  **private** **int** marks;  **private** String name;  @OneToOne  **private** Laptop laptop;  @OneToMany(mappedBy="student")  **private** List<Laptop> laptoptoMany = **new** ArrayList<Laptop>();    @ManyToMany(mappedBy="student")  **private** List<Laptop> manyLaptopstoManyStudent = **new** ArrayList<Laptop>();    **public** **int** getRollNo() {  **return** rollNo;  }  **public** **void** setRollNo(**int** rollNo) {  **this**.rollNo = rollNo;  }  **public** String getGroup() {  **return** group;  }  **public** **void** setGroup(String group) {  **this**.group = group;  }  **public** **int** getMarks() {  **return** marks;  }  **public** **void** setMarks(**int** marks) {  **this**.marks = marks;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** Laptop getLaptop() {  **return** laptop;  }  **public** **void** setLaptop(Laptop laptop) {  **this**.laptop = laptop;  }  **public** List<Laptop> getLaptops() {  **return** laptoptoMany;  }  **public** **void** setLaptops(List<Laptop> laptops) {  **this**.laptoptoMany = laptops;  }  } |

Hibernate.cfg.xml

|  |
| --- |
| <?xml version = *"1.0"* encoding = *"utf-8"*?>  <!DOCTYPE hibernate-configuration SYSTEM  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>      <property name = *"hibernate.connection.driver\_class"*>com.mysql.jdbc.Driver</property>  <!-- Assume test is the database name -->  <property name = *"hibernate.connection.url"*>jdbc:mysql://localhost/test</property>  <property name = *"hibernate.connection.username"*>root</property>  <property name = *"hibernate.connection.password"*>root123</property>  <property name = *"hibernate.dialect"*>org.hibernate.dialect.MySQLDialect</property>  <property name = *hbm2ddl.auto*>create/update</property>  <property name = *show\_sql*>true</property>  <!-- List of XML mapping files -->  <mapping resource = *"Employee.hbm.xml"*/>  <!-- to enable 2nd level cache -->  <property name=*"hibernate.cache.use\_second\_level\_cache"*>true</property>  <property name=*"hibernate.cache.region.factory\_class"*>org.hibernate.cache.ehcache.EhCacheRegionFactory</property>    <property name=*"hibernate.cache.use\_query\_cache"*>true</property>    </session-factory>  </hibernate-configuration> |

Persistence.xml

|  |
| --- |
| <persistence xmlns=*"http://xmlns.jcp.org/xml/ns/persistence"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* version=*"2.2"*  xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/persistence http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_2.xsd"*>  <!-- Define persistence unit -->  <persistence-unit name=*"my-persistence-unit"*>  <properties>  <property name=*"javax.persistence.jdbc.driver"* value=*"com.mysql.jdbc.Driver"* />  <property name=*"javax.persistence.jdbc.url"* value=*"jdbc:mysql://localhost:3306/sandeep"* />  <property name=*"javax.persistence.jdbc.user"* value=*"root"* />  <property name=*"javax.persistence.jdbc.password"* value=*"root"* />  </properties>  </persistence-unit>  </persistence> |