**HIBERNATE**

**Introduction**

**Enterprise:** It is a Business Organization or a Group of Organizations running under Single label.

**Enterprise Application:** It is a software Application designed for an enterprise in order to simplify their business processing.

To prepare Enterprise Applications we have to provide the following layers.

1. User Interface Layer

2. Business Processing Layer

3. Data Storage and Access Layer

### **1.** **User Interface Layer:**

Ø It is a top most layer in enterprise applications, it will improve Look and Feel to the enterprise applications.

Ø It will provide starting point to the users in order to interact with enterprise application.

Ø It will provide very good environment to get data from users in order to execute enterprise applications.

Ø It will provide very good environment to perform client side data validations by executing Java script functions.

Ø To prepare this layer we will use a separate logic called as "Presentation Logic".

Ø In enterprise applications, to prepare Presentation Logic we will use the technologies like AWT, SWING, Java FX, Html, JSP, Java Script, Velocity, Freemarker,....

### **2.** **Business Processing Layer:**

Ø It is heart of Enterprise Applications.

Ø It will provide very good environment to define and execute business rules and regulations which are required by the client.

Ø To prepare this Layer we will use a separate logic called as "Business Logic".

Ø To prepare this layer we will use the tech like Servlets, EJBs-Session Beans,....

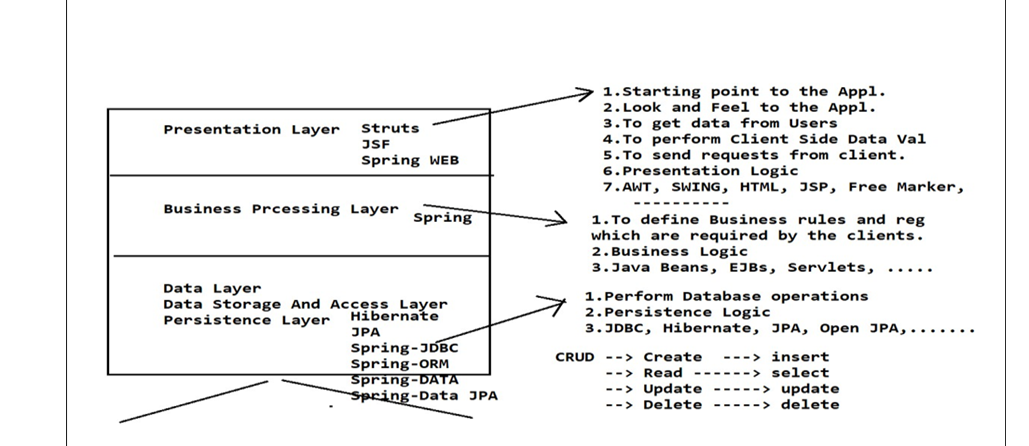
### **3.** **Data Storage and Access Layer:**

Ø It bottom most layer in enterprise applications.

Ø Its main intention is to provide data persistency in Enterprise applications, that is, it will provide very good environment to interact with database from java applications in order to perform database operations.

Ø To create this layer we will use a separate logic called as "Persistence Logic".

Ø To prepare Persistence Logic in Enterprise applications we will use a set of technologies or products like JDBC, Hibernate, JPA, toplink,...



## **Data Persistency:**

Representing Data permanently in Backend systems is called as "Data Persistency".

To achieve Data Persistency in database applications we will use a set of Operations [CRUD] called as Date Persistence Operations".

To achieve Data Persistency in Enterprise Applications we will use a set of tech called as "Data Persistency Tech".

To achieve Data Persistency in enterprise applications we will use the following technologies w.r.t JAVA.

1) Serialization and Deserialization

2) JDBC

3) ORM Implementations

a) Hibernate

b) EJBs Entity Beans

c) JPA

d) Open JPA

e) Toplink

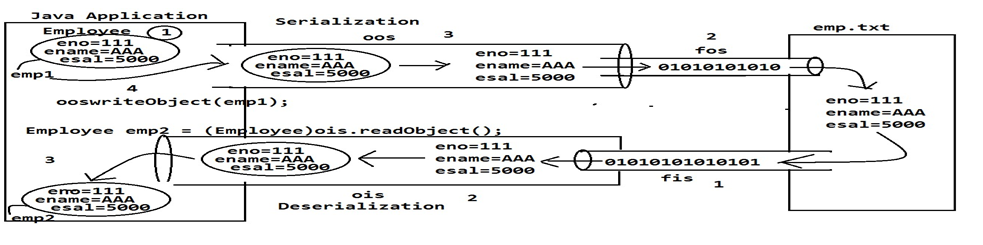
**Data Persistency through Serialization and Deserialization:** The process of Separating Data from an Object is called as "Serialization".

The process of regenerating an object on the basis of Data is called as "Deserialization".

To perform Serialization and Deserialization JAVA has provided the following two byte oriented streams

1. Object OutputStream for Serialization

2. Object InputStream for Deserialization



### **Steps to perform Serialization:**

**1.** **Prepare Serializable class by implementing java.io.Serializable marker interfae:**

class Employee implements Serializable{

}

Employee emp1 = new Employee();

### **2.** **Create FileOutputStream object:**

FileOutputStream fos = new FileOutputStream("emp.txt");

### **3.** **Create ObjectOutputStream for Serialization.**

ObjectOutputStream oos = new ObjectOutputStream(fos);

### **4.** **Write Serializable Object in ObjectOutputStream:**

oos.writeObject(emp1);

### **Steps to perform Deserialization:**

1. Create FileInputStream from Source File:

FileInputStream fis = new FileInputStream("emp.txt");

2.Create ObjectInputStream for Deserialization:

ObjectInputStream ois = new ObjectinputStream(fis);

3.Read Deserialized Object from ObjectInputStream:

Employee emp2 = (Employee)ois.readObject();

§ In Serialization and Deserialization Data Persistency mechanism, we will store data in file system ,where file system is providing permanent storage for our data, so that, Serialization and Deserialization is a Data Persistency Mechanism.

§ In Serialization and Deserialization data persistency mechanism, we will use a flat file to store data, it is platform dependent, it is not suitable for the platform independent front end technologies like JAVA, .NET,....

§ In Serialization and Deserialization Data Persistency mechanism, we will use two byte oriented streams like ObjectOutputStream and ObjectInputStream to perform Serialization and Deserialization, so that, this data persistency mechanism is suitable for only byte oriented data.

§ In this data persistency mechanism, we are using file system to store data , it able to store less data , it able to provide less security and it able to increase data redundancy, it is not suggestible in applications.

§ This persistency mechanism is suitable for Standalone applications, not suitable for enterprise applications.

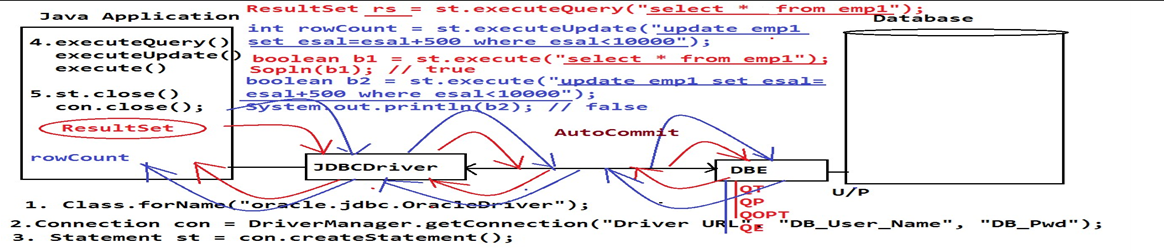
§ If we use this data persistency mechanism in Enterprise applications then we have to provide lot of java code.

§ In Serialization and Deserialization data persistency mechanism we are able to perform only insertion and retrieval operations, we are unable to perform update, delete,... operations.

§ In this data persistency mechanism, query language support is not existed so that database operations are very much difficult.

## **Data Persistency through JDBC:**

JDBC is a step by step process or a technology or an API or the collection of predefined classes and interfaces or an abstraction, it will provide very good environment to interact with database from java applications in order to perform database operations from java applications.



### **To prepare JDBC applications we have to use the following steps.**

1. Load and Register Driver: Class.forNmae(“com.mysql.cj.jdbc.driver”)

2. Enstablish Connection between Java application and Database.

Connection con = DriverManager.getConnection("jdbc:mysql://localhost/mallidb”,”root”,”root”);

3. Create either Statement or PreparedStatement or CallableStatement as per the requirement.

Statement st = con.createStatement();

4. Write and execute SQL Queries.

ResultSet rs = st.executeQuery("select \* from emp1");

5. Close the connection

St.close();

Rs.close();

Con.close();

v In Jdbc Applications, when we establish Connection between java application and database , automatically, the generated Connection will be available in "Auto-Commit" mode, in this mode, if we submit an sql query to the Connection then Connection will carry that sql query to Database Engine and Connection will make Database Engine to execute SQL Query and to results in Database Permanently, this Nature of Connection is able to achieve Data Persistency in JDBC Applications.

v In JDBC applications, we have provide SQL Queries explicitly to perform database operations, So that, Developers must required SQL Awareness.

v In JDBC applications, developers must have explicitly concentration on Driver Management, Connection Management and Statement Management.

v In JDBC Applications, the steps like Load and Register Driver, Establish Connection , Create Statement and Close resources ... are common ion every JDBC Application, we must write repeatedly, it will increase Code duplication .

v In Jdbc applications, we have to hardcode the JDBC Parameters like Driver class name, Driver URL, Database User Name and Databse password,...

v In JDBC , Transactions support is not good.

v In JDBC almost all the exceptions are checked exceptions, we must handle them by providing JAVA code.

v In JDBC applications, if we retrive data from database then that will be stored in ResultSet object at java applications, it is not implementing java.io.Serializable interface and which is not eligible to carry in network.

### **2.** **EJB Vs HIBERNATE:**

**WHAT IS THE DIFFERENCE B/W EJB&HIBERNATE?**

i. In case of EJB’S entity beans, we should require 1-1 relation b/w database table and the entity classes. But it is not mandatory in case of hibernate.in case of hibernate, a single entity class may represent multiple no.of tables and single table may represent multiple no.of entity classes.

ii. EJB’S entity beans is more API dependent because in EJB’S entity beans all the entity classes must implement or extend predefined library provided by EJB API.

iii. hibernate is less API dependent because in Hibernate all the entity classes are by default POJO classes(plain old java object),these are normal java bean classes which should not extend (or) implement any predefined library.

iv. Due to the above reasons, Testing & Debugging is very difficult in entity but which are very simple incase of Hibernate.

v. In case of entity beans, all the bean classes are not participated in inheritance directly. But in case of hibernate, it is possible to provide inheritance relation b/w POJO classes.

vi. EJB’S entity beans should require application server to execute. But hibernate applications will be executed with or without application server.

vii. Due to the above reasons, entity beans portability is very less. Due to the above reason, hibernate portability is very good.

viii. Due to the above reason entity bean is heavy weight data persistency solution. Due to the above reason, hibernate is light weight data persistency solution.

ix. Entity beans are relatively slow and hibernate is faster data persistency solution except startup.

### **3.** **JPA Vs HIBERNATE:**

**What is the difference between JPA and Hibernate?**

· JPA is an abstraction provided by SUN Microsystems and implemented by all Application Server vendors like Web logic Server vendor, JBOSS Server vendor,...................................................................................... JPA provides a set

of conventions to implement ORM rules and regulations.

· Hibernate is a product, it has implemented ORM rules and regulations as per JPA guidelines in order to provide data Persistency in enterprise applications.

## **Hibernate History:**

**Home :** Read Hat

**Objective :** To simplify data persistency in Enterprise

**Author :** Gavin King

**Type :** ORM Implementation Product.

**Open/Licenced :** Open Source Software.

**Initial Version :** Hibernate1.0 [2001]

User versions : Hibernate3.x [2005]

Hibernate4.x [2011]

Latest Version : Hibernate5.x [2017]

Designed On : JAVA

Website : [www.hibernate.org](http://www.hibernate.org/)

Hibernate Features:

1. Hibernate is Database independent, it can be used for any type of Database.

2. Hibernate is applicable for both standalone applications and Enterprise Applications.

3. Hibernate is providing very good support for Associations and Joins.

4. Hibernate is having very good Annotations in order to reduce XML tech dependency in enterprise applications.

5. Hibernate is having very good implementations for Primary key generation algorithms in order to generate and insert a unique primary key value for each and every insertion operation.

6. Hibernate is providing very good Collections support to manage data.

7. Hibernate is having its own query language, which is database independent, Object Oriented, that is, HQL in order to perform database operations.

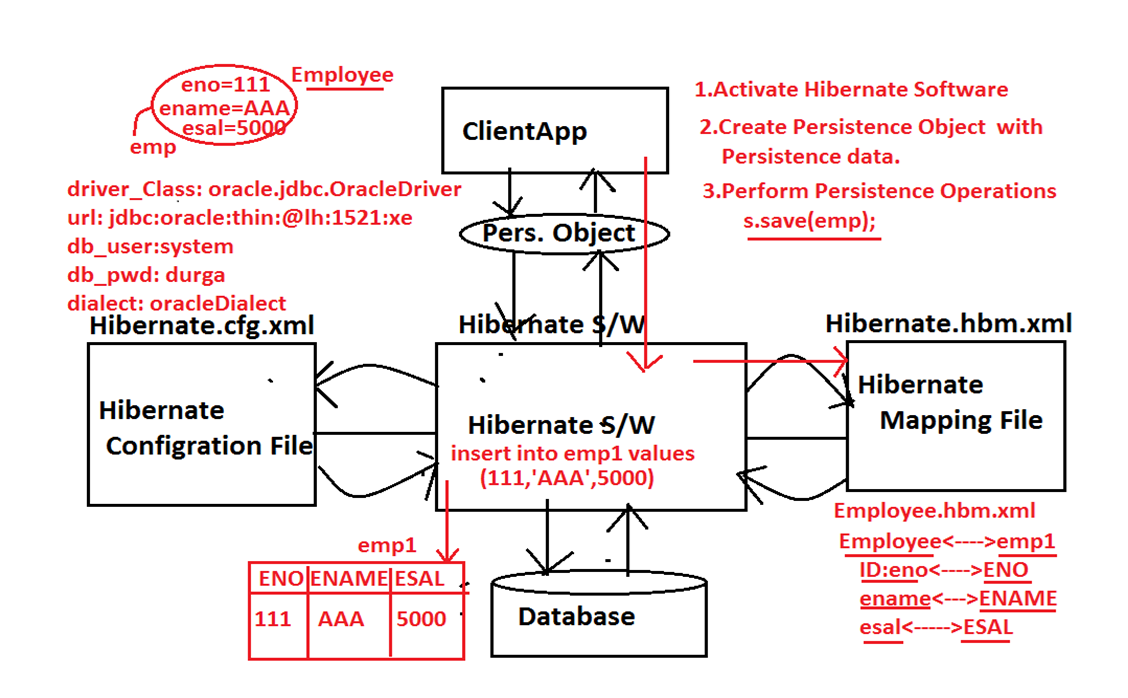
8. Hibernate has provided very good Cache mechanisms in order to reuse the results.

9. Hibernate is having very good support for Connection Pooling mechanisms in order to improve Connection re usability.

10. Hibernate is supported by almost all IDEs and Application Servers.

11. Hibernate is against for SQL Queries in enterprise applications directly, but, if we want to write database dependent sql queries then it is possible to provide database dependent sql queries by using "Native SQL".

12. Hibernate has provided very good transactions support.



as

From the above Arch.,

Hibernate Configuration File will provide all the configuration details like driver class name, driver URL, Database User name , Database password,.............................................................................. which we required to establish connection

with database and to setup JDBC environment.

Hibernate Mapping file will provide all the mapping details like Bean Class name and Database table name, ID property and Primary key column , Normal Properties and Normal Columns,....

1) Client Application will perform the following three actions in Hibernate applications mainly.

2) Activating Hibernate Software, in this case, Hibernate Software will take all configuration details from hibernate configuration file and Hibernate Software will set up the required JDBC Environment to perform database operations.

3) Prepare Persistence Object with the persistence data.

4) Perform Persistence Operation.

When Client Application perform persistence operation, Hibernate Software will perform the following actions.

1) Hibernate Software will take persistence method call and identify persistence Object.

2) Hibernate Software will take all mapping details from hibernate mapping file like database table name and all column names on the basis of Persistence object.

3) Hibernate Software will prepare database dependent sql query on the basis of table names and column names and with the persistence object provided data.

4) Hibernate Software will execute the generated database dependent sql query and perform the required persistence operatio

# **Steps to prepare First Hibernate Application**

**Steps to prepare First Hibernate Application:**

1) Prepare Persistence Class or Object.

2) Prepare Mapping File.

3) Prepare Hibernate Configuration File

4) Prepare Hibernate Client Application

**Note:** To run Hibernate Applications we have to download all Hibernate JAR file and we have to them for our Hibernate application.

## **1)** **Prepare Persistence Class or Object:**

The main intention of Persistence class or object in Hibernate applications is to manage Persistence data which we want to store in Database or by using this we want to perform the database operations like select, update, delete,

In Hibernate applications, to prepare Persistence classes we have to use the following Guidelines

1) In Hibernate applications Persistence classes must be POJO classes [Plain Old Java Object], they must not extend or implement predefined Library.

2) In hibernate Applications Persistence classes must be public, Non abstract and non-final.

Where the main intention to declare persistence classes as public is to bring persistence classes scope to Hibernate software in order to create objects.

Where the main intention to declare persistence classes as Non abstract is to allow to create Objects for Persistence classes

Where the main intention to declare persistence classes as Non final is to allow to extend one persistence class to another persistence class as per the requirement.

3) In Persistence classes all Properties must be declared as per database table provided columns, where names are not required to be matched, but, data types must be compatible.

4) In Persistence classes, all properties must be declared as private in order to improve Encapsulation.

5) In Persistence classes , we must define a separate set of setXXX () and getXXX () methods for each and every property

6) In persistence classes, we must declare all methods are public.

7) In Persistence classes, if we want to provide any constructor then it must be public and 0- arg constructor, because, while creating object for persistence class Hibernate software will search and execute only public and 0-arg constructor.

8) In Hibernate applications , if we want to provide our own comparison mechanisms while comparing Persistence objects then it is suggestible to Override equals(--) method.

9) In Hibernate applications, if we want to provide our own hash code values then it is suggestible to Overrid hashCode () method.

10) In Hibernate applications, we will use POJO classes, which are not extending and implementing predefined library, but, it is suggestible to implement java.io.Serializable marker interface in order to make eligible Persistence object for Serialization and Deserialization.

11) In Persistence classes, we have to declare a property as an ID property, it must represent primary key column in the respective table.

example

**public** **class** Employee

{

**private** **int** eid;

**private** String ename;

**private** **int** esal;

**public** **int** getEid() {

**return** eid;

}

**public** String getEname() {

**return** ename;

}

**public** **int** getEsal() {

**return** esal;

}

**public** **void** setEid(**int** eid) {

**this**.eid = eid;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **void** setEsal(**int** esal) {

**this**.esal = esal;

}

}

| **2. Prepare Mapping File:**    The main intention of mapping file in Hibernate applications is to provide mapping between a class  ,id property and normal properties from Object Oriented Data Model and a table, primary key column and normal columns from Relational data model.    In Hibernate applications, mapping file is able to provide the mapping details like Basic OR mapping, Component mapping, inheritance mapping, Collections mapping, Associations mapping,    To prepare mapping file in Hibernate applications we have to provide mapping file name with the following format.    **POJO\_Class\_Name.hbm.xml**    The above format is not mandatory, we can use any name but we must provide that intemation to the hibernate software.    In Hibernate applications, we can provide any no of POJO classes, w.r.t each and every POJO class we can define a separate mapping file.    **Note:** In Hibernate applications, it is possible to configure more than POJO class in single mapping file.    Upto Hibernate3.2.5 version Hibernate Mapping file is mandatory to provide mapping details, but, right from Hibernate3.2.5 version mapping file is optional, because, Hibernate 3.2.5 version has provided annotations as an alternative to mapping file.    To provide Basic mapping between POJO class and table in mapping file we have to use the following XML tags provided by Hibernate. |
| --- |

1. <!DOCTYPE >

2. <hibernate-mapping>

3. <class name="--" table="--">

4. <id name="--" column="--"/>

5. <property name="--" column="---"/>

ü Where <hibernate-mapping> tag is root tag in mapping file, it will include no of classes configuration.

ü Where <class> tag is able to provide single POJO class configuration inorder to provide mapping between POJO class name and the respective table name.

ü Where "name" attribute in <class> tag is able to provide fully qualified name of the POJO class.

ü Where "table" attribute in <class> tag is able to provide the respective table name.

ü Where <id> tag is able to provide ID property configuration inorder to provide mapping between ID property and the respective primary key column.

ü Where "name" attribute in <id> tag is able to provide id property name of the POJO class.

ü Where "column" attribute in <id> tag is able to provide primiry key column defined in table.

ü Where <property> tag is able to provide normal bean property configuration inorder to provide mapping between normal bean property and normal table column.

ü Where "name" attribute and "column" attribute will take bean property name and table column name respectively.

**Note:** If bean property names and table column names are same then it is optional to provide "column" attribute in mapping file.

**EX:**

1. <!DOCTYPE >

2. <hibernate-mapping>

3. <class name="com.newzen.Employee" table="emp1">

4. <id name="eno" column="eno"/>

5. <property name="ename" column=”ENAME”/>

6. <property name="esal" column="emp\_Sal"/>

7. <property name="eaddr" column="eaddr"/>

8. </class>

9. </hibernate-mapping>

**3. Prepare Hibernate Configuration File:**

The main purpose of hibernate configuration file is to provide all configuration details of hibernate application which includes Jdbc parameters to prepare connection , Transactions configurations, Cache mechanisms configurations, Connection pooling configurations,........

In Hibernate applications, the standard name of hibernate configuration file is "hibernate.cfg.xml", it is not fixed, we can provide any name but that name must be given to Hibernate software.

In Hibernate applications, we are able to provide more than one configuration file , but, for each and every database, that is, in hibernate applications if we use multiple databases then we are able to prepare multiple configuration files.

To prepare hibernate configuration file with basic configuration details we have to use the following XML tags.

1. <!DOCTYPE >

2. <hibernate-configuration>

3. <session-factory>

4. <property name="--"> value </property>

5. ------

6. ------

7. <mapping resource="---"/>

8. -----

9. </session-factory>

10. </hibernate-configuration>

ü Where <hibernate-configuration> is root tag.

ü Where <session-factory> tag is able to include hibernate properties configurations.

ü Where <property> tag is able to provide single property configuration.

ü Where "name" attribute in <property> tag is able to provide property name and body of the

<property> tag will take property value.

ü Where <mapping> tag is able to provide mapping file configuration.

ü Where "resource" attribute is able to provide mapping file name and location.

EX:

hibernate.cfg.xml:

1. <!DOCTYPE>

2. <hibernate-configuration>

3. <session-factory>

4. <property name="connection.driver\_Class">com.mysql.cj.jdbc.Driver</property>

5. <property name="connection.url">jdbc:mysql://localhost/adjava</property>

6. <property name="connection.user">root</property>

7. <property name="connection.password">root</property>

8. <property name="hibernate.dialect"> org.hibernate.dialect.mysqlDialect</property>

9. <mapping resource="Employee.hbm.xml"/>

10. </session-factory>

11. </hibernate-configuration>

In Hibernate configurations, "dialect" is a property, it able to provide dialect class name, it is providing the respective database native implementations which we are using in hibernate applications in order to prepare database dependent sql queries by Hibernate software.

Hibernate has provided separate predefined class in the form of org.hibernate.dialect.XXXDialect for each and every database.

**EX:** org.hibernate.dialect.Oracle9Dialect org.hibernate.dialect.Oracle10gDialect org.hibernate.dialect.MySQLDialect

**4.** **Prepare Client Application:**

The main intention of Hibernate Client application is to activate Hibernate Software, creating persistence objects and performing Persistence operations.

To prepare Client Application in hibernate applications we have to use the following steps as per Hiberante3.x version.

1. Create Configuration class object

2. Create Session Factory object

3. Create Session Object

4. Create Transaction object if it is required.

5. Perform Persistence operations

6. Close Session Factory and Session objects.

### **1. Create Configuration class object**

In Hibernate, the main intention of Configuration object is to store all the configuration details which we provided in hibernate configuration file.

To represent Configuration object Hibernate has provided a predefined class in the form of "org.hiberante.cfg.Configuration".

To create Configuration class object we have to use the following constructor from Configuration class.

public Configuration()

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

If we use the above instruction in Hibernate applications then we are able to get an empty Configuration object in heap memory, it will not include any Configuration details.

If we want to store Configuration details from Configuration file we have to use either of the following methods.

### **1.** **public Configuration configure()**

This method will get configuration details from the configuration file with the name hibernate.cfg.xml

### **2.** **public Configuration configure(String config\_file\_Name)**

This method can be used to get configuration details from hibernate configuration file with any name, it will be used when we change configuration file name from hibernate.cfg.xml file to some other name .

### **3.** **public Configuration configure(File file)**

This method can be used to get configuration details from a file which is represented in the form of java.io.File class object.

public Configuration configure(URL url)

This method can be used to get Configuration details from a file which is available in network represented in the form of java.net.URL .

**EX:** Configuration cfg = new Configuration(); cfg.configure();

When we use configure() method then Hibernate Software will search for hibernate.cfg.xml file, if it is available then Hibernate software will load the content of hibernate.cfg.xml file, parse it and read content from configuration file to Configuration object.

### **2.** **Create Session Factory object:**

In Hibernate, the main intention of Session Factory object is to manage Connections, Statements, Cache levels, and it able to provide no of Hibernate Session objects.

To represent Session Factory object Hibernate has provided a predefined interface in the form of "org.hibernate.Session Factory".

To get Session Factory object we have to use the following method from Configuration class. public Session Factory buildSessionFactory()

**EX:** SessionFactory sf = cfg.buildSessionFactory();

**Note:** The above approach to get Session Factory object is available upto Hibernate3.x version, buildSessionFactory() was deprecated in Hibernate4.x version.

In Hibernate applications, if we use multiple Databases then we have to prepare multiple Configuration files, multiple Configuration Object, w.r.t this, we have to prepare multiple Session Factory objects.

Session Factory object is heavy weight and it is thread safe upto a particular Database, because, it able to allow more than one thread at a time.

### **3.** **Create Session Object:**

In Hibernate, for each and every database interaction a separate Session will be created.

In Hibernate, Session is able to provide no of persistence methods in order to perform persistence operations.

To represent Session object, Hibernate has provided a predefined interface in the form of "org.hibernate.Session".

To get Session object, we have to use the following method from Session Factory. public Session openSession()

EX: Session s =sf.openSession();

In Hibernate, Session object is light weight and it is not thread safe, because, for each and every thread a separate Session object will be created.

### **4.** **Create Transaction Object:**

Transaction is a unit of work performed by Front End applications on Back end systems.

To represent Transactions, Hibernate has provided a predefined interface in the form of "org.hibernate.Transaction".

To get Transaction object we will use either of the following methods.

### **1.** **public Transaction getTransaction()**

It will return Transaction object with out begin, where to begin Transaction we have to use the following method.

public void begin()

### **2.** **public Transaction beginTransaction()**

It will return Transaction and begin Transaction.

In Hibernate applications, after performing persistence operations we must perform either commit or rollback operations inorder to complete Transactions, for this, we have to use the following methods from Transaction.

Public void commit()

Public void rollback()

**Note:** In Hibernate applications, Transaction is required for only Non Select operations, not required for Select operations.

### **5. Perform Persistence Operations:**

In Hibernate applications, to perform persistence operations Session has provided the following methods.

To insert an object or record into Database table we have to use the following methods.

1. save(--)

2. persist(--)

### **6.** **Close Session and SessionFactory :**

In Hibernate applications, it is convention to close Session and Sessionfactory objects at the end of client application in order to avoid security problems.

public void close()throws HibernateException.

**EX:** s.close();

sf.close();

Application-1: using Hibernate 3.X version

----------------------------------------------

Employee.java

**package** com.codegnan.beans;

**public** **class** Employee {

**private** **int** eid;

**private** String ename;

**private** **int** esal;

**public** **int** getEid() {

**return** eid;

}

**public** String getEname() {

**return** ename;

}

**public** **int** getEsal() {

**return** esal;

}

**public** **void** setEid(**int** eid) {

**this**.eid = eid;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **void** setEsal(**int** esal) {

**this**.esal = esal;

}

}

Employee.hbm.xml

----------------------------

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

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

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

<hibernate-mapping>

<class name=*"com.codegnan.beans.Employee"* table=*"employee"*>

<id name=*"eid"* column=*"eid"* type=*"int"* />

<property name=*"ename"* column=*"ename"* type=*"string"* />

<property name=*"esal"* column=*"esal"* type=*"int"* />

</class>

</hibernate-mapping>

Hibernate.cfg.xml

---------------------------

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

<!DOCTYPE hibernate-configuration PUBLIC

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

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

<hibernate-configuration>

<session-factory>

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

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

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

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

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

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

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

</session-factory>

</hibernate-configuration>

ClientApp.java

------------------------

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.beans.Employee;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.configure();

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

com.codegnan.beans.Employee employee = **new** Employee();

employee.setEid(1);

employee.setEname("bunny");

employee.setEsal(100000);

session.save(employee);

System.***out***.println("Data inserted successfully");

transaction.commit();

} **catch** (Exception e) {

**if** (transaction != **null**) {

transaction.rollback();

}

e.printStackTrace();

} **finally** {

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate\_01</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate\_01</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

</dependencies>

</project>

**Is it possible to run hibernate applicatins with out using mapping file?**

**ANS:**

Yes, it is possible to run hibernate applications with out using mapping file, but, we have to use Annotations.

To provide mapping details in Hibernate applications we have to use the following annotations provided by JPA in the form of "javax.persistence" package.

**1.** **@Entity:**

It is a class level annotation, it can be used to declare a class as an Entity class.

**2.** **@Table(name="--")**

This annotation is a class level annotation, it can be used to configure table name for the entity class. Where "name" member is able to provide the respective table name.

**3.** **@Id :**

This annotation is Field/Method[getXXX()] level annotation, it can be used to declare a property as ID property.

**4.** **@Column(name="--")**

This annotation is Field/Method[getXXX()] level annotation, it can be used to provide a database table column name inorder to provide mapping .

1) In Hibernate Applications, to use annotations we have to use the following steps.

2) Remove mapping file and provide annotations ion POJO class.

3) In Configuration File , provide <mapping> tag with "class" attribute inplace of "resource" attribute."class" attribute will take fully qualified name of the respective POJO class.

4) In Client Application, Create Either AnnotationConfiguration object or Configuration object to process Annotations.

**Note:** AnnotationConfiguration is deprecated class, so it is better to use Configuration class object to process annotations.

Ex: Employee.java

@Entity

@Table(name="emp1")

public class Employee{

@Id

@Column(name="ENO")

private int eno;

@Column(name="ENAME")

private String ename;

private float esal;

@Column(name="EADDR")

private String eaddr;

setXXX() and getXXX()

}

**Is it possible to run Hibernate Applications without using configuration file?**

**ANS:**

Yes, it is possible to run Hibernate applications without using configuration file.

In Hibernate Applications, we are able to provide configuration details in the following two approaches.

1) Programmatic Approach

**1) Programmatic Approach**

In Programmatic approach, to provide configuration details, first we have to create Configuration class object then we have to set all hibernate properties to Configuration class object explicitly by using the following method.

public void setProperty(String prop\_Name, String prop\_Val)

To add mapping file name and location to Configuration file we have to use the following method. public void addResource(String mapping\_File\_Name)

**Note:** If we are using annotations in place of mapping file then we have to use the following method to add annotated class.

public void addAnnotatedClass(Class class)

**Is it possible to prepare hibernate applications without using mapping file and configuration file**

**==========================================================================**

**Yes possible**

**Insert data example**

**—--------------------------**

**Ex:**

**====**

**Employee.java**

**—---------------------**

**package** com.codegnan.beans;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "employee")

**public** **class** Employee {

@Id

@Column(name = "eno")

**private** **int** eno;

@Column(name = "ename")

**private** String ename;

@Column(name = "esal")

**private** **float** esal;

@Column(name = "eaddr")

**private** String eaddr;

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

@Override

**public** String toString() {

**return** "Employee [eno=" + eno + ", ename=" + ename + ", esal=" + esal + ", eaddr=" + eaddr + "]";

}

}

**ClientApp.java**

**=================**

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.beans.Employee;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

Employee emp = **new** Employee();

emp.setEno(111);

emp.setEname("AAA");

emp.setEsal(60000);

emp.setEaddr("hyderabad");

session.save(emp);

transaction.commit();

System.***out***.println("Employee records are inserted succesfully");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

Application -3

Delete records example

======================

**package** com.codegnan.beans;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "employee")

**public** **class** Employee {

@Id

@Column(name = "eno")

**private** **int** eno;

@Column(name = "ename")

**private** String ename;

@Column(name = "esal")

**private** **float** esal;

@Column(name = "eaddr")

**private** String eaddr;

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

@Override

**public** String toString() {

**return** "Employee [eno=" + eno + ", ename=" + ename + ", esal=" + esal + ", eaddr=" + eaddr + "]";

}

}

**ClientApp.java**

**=================**

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.beans.Employee;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

Employee emp = **new** Employee();

emp.setEno(111);

session.delete(emp);

transaction.commit();

System.***out***.println("Employee records are deleted succesfully");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

-------------------Application-4-----------------------

Example to retrieve records from DataBase

-------------------------------------------------------------------

**Employee.java**

**—---------------------**

**package** com.codegnan.beans;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "employee")

**public** **class** Employee {

@Id

@Column(name = "eno")

**private** **int** eno;

@Column(name = "ename")

**private** String ename;

@Column(name = "esal")

**private** **float** esal;

@Column(name = "eaddr")

**private** String eaddr;

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

@Override

**public** String toString() {

**return** "Employee [eno=" + eno + ", ename=" + ename + ", esal=" + esal + ", eaddr=" + eaddr + "]";

}

}

**ClientApp.java**

**=================**

**package** com.codegna.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

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

**import** com.codegnan.beans.Employee;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.addAnnotatedClass(Employee.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

Employee emp = (Employee) session.get(Employee.**class**, 111);

**if** (emp == **null**) {

System.***out***.println("Employees doesn't existed");

} **else** {

System.***out***.println("Employee Details");

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

System.***out***.println("Employeee Number : " + emp.getEno());

System.***out***.println("Employeee Name : " + emp.getEname());

System.***out***.println("Employeee Salary : " + emp.getEsal());

System.***out***.println("Employeee Address " + emp.getEaddr());

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

# **Primary Key Generation Algorithms in Hibernate**

· Primary key is single column or Collection of columns in a table to recognize the records individually.

· In Database applications, to perform the database operations like retriving a record, updating a record, deleting a record,we need primary key and its value.

· In Database applications, we are unable to give option to the users to enter primary key values , because, there is no guarantee for the data entered by the users whether it is unique data or not, but, in Database tables only unique values are accepted by primary key columns.

· To give guarantee for uniqueness in primary key values we have to use Primary key generation algorithms.

· Almost all the Persistence mechanisms like Hibernate, JPA, Open JPA, Toplink,......................................................................................... are

having their own implementation for primary key generation algorithms.

**Annotations Support for Primary Key Generation Alg:**

Hibernate has provided annotations for almost all the primary key generation alg in the form of javax.persistence package and org.hibernate.annotation package.

To provide annotations for primary key generation alg javax.persistence package has provided the following annotations.

§ @GeneratedValue(--,--)

§ @SequenceGenerator(--)

@TableGenerator(--)

---

---

To provide a particular primary key generation alg in hibernate applications, we have to use "strategy" member in @GeneratedValue(-) annotation, it will take either of the following constants from "GenerationType" Enum.

1. IDENTITY

2. SEQUENCE

3. TABLE

4. AUTO

**1. IDENTITY:** This value of GenerationType enum will represent IdentityGenerator or "identity" primary key generation algorithm to generate primary key value on the basis of the underlying database provided identity column.

**EX: Employee.java**

@Entity

@Table(name="emp1")

**public** **class** Employee {

@Id

@Column(name="ENO")

@GeneratedValue(strategy=GenerationType.IDENTITY)

**private** **int** eno;

@Column(name="ENAME")

**private** String ename;

@Column(name="ESAL")

**private** **float** esal;

@Column(name="EADDR")

**private** String eaddr;

setXXX() and getXXX()

}

Example program on identity generator//s

----------------------------------------

//Create table emp2(eno int primary key auto\_increment,ename varchar(20),esal int,eaddr varchar(20));

Employee.java

----------------------

**package** com.codegnan.beans;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.GenerationType;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "emp2")

**public** **class** Employee {

@Id

@Column(name = "eno")

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

**private** **int** eno;

@Column(name = "ename")

**private** String ename;

@Column(name = "esal")

**private** **float** esal;

@Column(name = "eaddr")

**private** String eaddr;

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

}

ClientApp.java

—--------------------

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.beans.Employee;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

//configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

Employee emp = **new** Employee();

emp.setEno(222);

emp.setEname("malli");

emp.setEsal(2000);

emp.setEaddr("hyd");

session.save(emp);

transaction.commit();

System.***out***.println("Employee records are inserted succesfully");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

Example program on table generator//s

---------------------------------------------------------------

create table my\_table(id int primary key,next\_hi int);

create table emp1(eno int,ename varchar(10),esal int,eaddr varchar(10));

Employee.java

--------------------------

**package** com.codegnan.beans;

**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;

**import** javax.persistence.TableGenerator;

@Entity

@Table(name = "emp1")

**public** **class** Employee {

@Id

@Column(name = "eno")

@TableGenerator(name = "tableGenerator", table = "my\_table", pkColumnName = "id", pkColumnValue = "10", valueColumnName = "next\_hi")

@GeneratedValue(strategy = GenerationType.***TABLE***, generator = "tableGenerator")

**private** **int** eno;

@Column(name = "ename")

**private** String ename;

@Column(name = "esal")

**private** **float** esal;

@Column(name = "eaddr")

**private** String eaddr;

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

}

ClientApp.java

================

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.beans.Employee;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

//configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

Employee emp = **new** Employee();

// emp.setEno(111);

emp.setEname("BBB");

emp.setEsal(4000);

emp.setEaddr("VJY");

**int** pk\_val = (**int**) session.save(emp);

transaction.commit();

System.***out***.println(pk\_val + " Employee Inserted Succesfully");

} **catch** (Exception e) {

**if** (transaction != **null**) {

transaction.rollback();

}

} **finally** {

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

**4. Association Mapping:**

In General, in Enterprise applications we will use both Object Oriented Data Model and Relational Data Model to represent data. In Enterprise applications we will use Object Oriented Data Model in Front-End applications toreprepsent data and we will used Relational data Model in Back-End Applications to represent Data.

In the above context, both the data models are having their own approaches to represent data, the differences between data models may provide mismatches between data models, it may reduce data persistency in Enterprise Applications.

In the above context, to improve data persistency we must use ORM implementations, Hibernate is an ORM implementation , it able to provide solutions for mismatchs between Data models.

**In Enterprise applications, we are able to get Associations mismatch while achieving associations in Object Oriented Data Model and in Relational Data Model.**

**IN Enterprise Applications, the main intention of Associations is to to improve data navigation between entities and to improve communication between entities.**

**In Object Oriented Data Model, we are able to achieve associations by declaring one or Collection of reference variables of an entity class in another entity class.**

**EX:**

**class Account{**

**}**

**class Address{**

**}**

**class Employee{**

**Account acc;// one-to-one association**

**Collection<Address> addr; //one-to-many**

**}**

**In relational Data Model, we are able to achieve associations in the following three ways.**

**1) By defining Primary Key-Foreign Key relations between tables**

**2) By Providing Join column between tables**

**3) By Defining Join table between tables.**

**To resolve the above associations mismatch between both the data models Hibernate has provided "Association Mappings".**

**Hibernate has provided the following four types of Associations mappings inorder to resolve Associations mismatch.**

**1. One-To-One Association Mapping**

**2. One-To-Many Association Mapping**

**3. Many-To-One Association Mapping**

**4. Many-To-Many Association Mapping**

**1. One-To-One Association Mapping**

**It is a relation between entities, where one instance of an entity should be mapped with exactly one instance of another entity.**

**EX:** Every Employee must have exactly one individual Account.

**Annotations support for one-to-one Assosiation**

**---------------------------------------------------**

**NOTE: if we perfom retrival operation through annotation it is mandatory. First insertion operation is required. After insertion operation you perfom the retrieve operation. Because through annotation it create the relation.**

Account.java

—---------------------

**package** com.codegnan.entity;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "account")

**public** **class** Account {

@Id

@Column(name = "ACCNO", length = 10)

**private** String accNo;

@Column(name = "ACCNAME", length = 10)

**private** String accName;

@Column(name = "ACCTYPE", length = 10)

**private** String accType;

@Column(name = "BALANCE", length = 10)

**private** **int** balance;

// constructors

// setters and getters

**public** Account() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Account(String accNo, String accName, String accType, **int** balance) {

**super**();

**this**.accNo = accNo;

**this**.accName = accName;

**this**.accType = accType;

**this**.balance = balance;

}

**public** String getAccNo() {

**return** accNo;

}

**public** **void** setAccNo(String accNo) {

**this**.accNo = accNo;

}

**public** String getAccName() {

**return** accName;

}

**public** **void** setAccName(String accName) {

**this**.accName = accName;

}

**public** String getAccType() {

**return** accType;

}

**public** **void** setAccType(String accType) {

**this**.accType = accType;

}

**public** **int** getBalance() {

**return** balance;

}

**public** **void** setBalance(**int** balance) {

**this**.balance = balance;

}

@Override

**public** String toString() {

**return** "Account [accNo=" + accNo + ", accName=" + accName + ", accType=" + accType + ", balance=" + balance

+ "]";

}

}

Employee.java

=============

**package** com.codegnan.entity;

**import** javax.persistence.CascadeType;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.OneToOne;

**import** javax.persistence.Table;

@Entity

@Table(name = "emp")

**public** **class** Employee {

@Id

@Column(name = "ENO", length = 10)

**private** **int** eno;

@Column(name = "ENAME", length = 10)

**private** String ename;

@Column(name = "ESAL", length = 10)

**private** **float** esal;

@Column(name = "EADDR", length = 10)

**private** String eaddr;

@OneToOne(cascade = CascadeType.***ALL***)

**private** Account acc;

// constructors getters and setters

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Employee(**int** eno, String ename, **float** esal, String eaddr, Account acc) {

**super**();

**this**.eno = eno;

**this**.ename = ename;

**this**.esal = esal;

**this**.eaddr = eaddr;

**this**.acc = acc;

}

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

**public** Account getAcc() {

**return** acc;

}

**public** **void** setAcc(Account acc) {

**this**.acc = acc;

}

@Override

**public** String toString() {

**return** "Employee [eno=" + eno + ", ename=" + ename + ", esal=" + esal + ", eaddr=" + eaddr + ", acc=" + acc

+ "]";

}

}

ClientApp.java

===========

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.entity.Account;

**import** com.codegnan.entity.Employee;

/\*public class ClientApp {

public static void main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

// configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

configuration.addAnnotatedClass(Account.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

try {

// Open session

session = sessionFactory.openSession();

// Begin transaction

transaction = session.beginTransaction();

// Create and set up Account

Account acc = new Account();

acc.setAccNo("abc123");

acc.setAccName("sunny");

acc.setAccType("Savings");

acc.setBalance(10000);

// Create and set up Employee

Employee emp = new Employee();

emp.setEno(111);

emp.setEname("sunny");

emp.setEsal(25000);

emp.setEaddr("Hyd");

emp.setAcc(acc); // Set the account for the employee

// Save Employee object, which will also save the Account due to cascading

int pk\_Val = (Integer) session.save(emp);

// Commit transaction

transaction.commit();

System.out.println(pk\_Val + " Employee inserted successfully");

} catch (Exception e) {

// Rollback transaction if an exception occurs

if (transaction != null) {

transaction.rollback();

System.out.println("Transaction rolled back due to an error.");

}

e.printStackTrace();

} finally {

// Close session and session factory

if (session != null) {

session.close();

}

if (sessionFactory != null) {

sessionFactory.close();

}

}

}

}\*/

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

// configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

configuration.addAnnotatedClass(Account.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

// Open session

session = sessionFactory.openSession();

// Begin transaction

transaction = session.beginTransaction();

// Retrieve Employee by primary key

Employee emp = (Employee) session.get(Employee.**class**, 111);

**if** (emp != **null**) {

// Print Employee details

System.***out***.println("Employee Details");

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

System.***out***.println("Employee Id : " + emp.getEno());

System.***out***.println("Employee Name : " + emp.getEname());

System.***out***.println("Employee Salary : " + emp.getEsal());

System.***out***.println("Employee Address : " + emp.getEaddr());

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

// Retrieve associated Account

Account acc = emp.getAcc();

**if** (acc != **null**) {

// Print Account details

System.***out***.println("Account Details");

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

System.***out***.println("Account Number : " + acc.getAccNo());

System.***out***.println("Account Name : " + acc.getAccName());

System.***out***.println("Account Type : " + acc.getAccType());

System.***out***.println("Account Balance : " + acc.getBalance());

} **else** {

System.***out***.println("No associated account found for Employee.");

}

} **else** {

System.***out***.println("Employee not found with ID: 111");

}

// Commit transaction

transaction.commit();

} **catch** (Exception e) {

// Rollback transaction if an exception occurs

**if** (transaction != **null**) {

transaction.rollback();

System.***out***.println("Transaction rolled back due to an error.");

}

e.printStackTrace();

} **finally** {

// Close session and session factory

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

**2. One-To-Many Association:**

**=====================================**

**It is a relation between entities where one instance of an entity should be mapped with multiple instances of another entity.**

**EX: Single Department has Multiple Employees**

**In Hibernate Applications, to represent One-To-Many association , JPA has provided the following annotation.**

**@OneToMany(cascade=CascadeType.ALL)**

**NOTE: if we are using annotations to perfoming the retrieve operation first we have to save the objects by using annotations and retring by annotations.**

**Normal saving object is different and hibernate annotations saving is different.**

**If we are perfoming the retrieve operations by using annotations in associations mapping firstly we save the objects by using annotations after we retrieving the operation by annotations**

**-------**

**Employee.java**

**----------------**

**package** com.codegnan.entity;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "emp1")

**public** **class** Employee {

@Id

@Column(name = "eno", length = 5)

**private** **int** eno;

@Column(name = "ename", length = 10)

**private** String ename;

@Column(name = "esal", length = 8)

**private** **float** esal;

@Column(name = "eaddr", length = 10)

**private** String eaddr;

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Employee(**int** eno, String ename, **float** esal, String eaddr) {

**super**();

**this**.eno = eno;

**this**.ename = ename;

**this**.esal = esal;

**this**.eaddr = eaddr;

}

**public** **int** getEno() {

**return** eno;

}

**public** **void** setEno(**int** eno) {

**this**.eno = eno;

}

**public** String getEname() {

**return** ename;

}

**public** **void** setEname(String ename) {

**this**.ename = ename;

}

**public** **float** getEsal() {

**return** esal;

}

**public** **void** setEsal(**float** esal) {

**this**.esal = esal;

}

**public** String getEaddr() {

**return** eaddr;

}

**public** **void** setEaddr(String eaddr) {

**this**.eaddr = eaddr;

}

@Override

**public** String toString() {

**return** "Employee [eno=" + eno + ", ename=" + ename + ", esal=" + esal + ", eaddr=" + eaddr + "]";

}

}

Department.java

=================

**package** com.codegnan.entity;

**import** java.util.Set;

**import** javax.persistence.CascadeType;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.OneToMany;

**import** javax.persistence.Table;

@Entity

@Table(name = "department")

**public** **class** Department {

@Id

@Column(name = "did", length = 5)

**private** String did;

@Column(name = "dname", length = 10)

**private** String dname;

@OneToMany(cascade = CascadeType.***ALL***)

**private** Set<Employee> emps;

**public** Department() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Department(String did, String dname, Set<Employee> emps) {

**super**();

**this**.did = did;

**this**.dname = dname;

**this**.emps = emps;

}

**public** String getDid() {

**return** did;

}

**public** **void** setDid(String did) {

**this**.did = did;

}

**public** String getDname() {

**return** dname;

}

**public** **void** setDname(String dname) {

**this**.dname = dname;

}

**public** Set<Employee> getEmps() {

**return** emps;

}

**public** **void** setEmps(Set<Employee> emps) {

**this**.emps = emps;

}

@Override

**public** String toString() {

**return** "Department [did=" + did + ", dname=" + dname + ", emps=" + emps + "]";

}

}

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.entity.Department;

**import** com.codegnan.entity.Employee;

**import** java.util.HashSet;

**import** java.util.Set;

/\*public class ClientApp {

public static void main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

configuration.addAnnotatedClass(Account.**class**);

// Build SessionFactory

SessionFactory sessionFactory = configuration.buildSessionFactory();

Session session = null;

Transaction transaction = null;

try {

// Open session

session = sessionFactory.openSession();

// Begin transaction

transaction = session.beginTransaction();

// Create Employee objects

Employee e1 = new Employee(111, "AAA", 5000, "Hyd");

Employee e2 = new Employee(222, "BBB", 6000, "Hyd");

Employee e3 = new Employee(333, "CCC", 7000, "Hyd");

Employee e4 = new Employee(444, "DDD", 8000, "Hyd");

// Create a Set of Employees

Set<Employee> employees = new HashSet<>();

employees.add(e1);

employees.add(e2);

employees.add(e3);

employees.add(e4);

// Create a Department object and set its properties

Department dept = new Department();

dept.setDid("D-111");

dept.setDname("ADMIN");

dept.setEmps(employees);

// Save the Department object, which will also save the associated Employees

session.save(dept);

transaction.commit();

System.out.println("Department Inserted Successfully");

} catch (Exception e) {

// Rollback transaction if an exception occurs

if (transaction != null) {

transaction.rollback();

System.out.println("Transaction rolled back due to an error.");

}

e.printStackTrace();

} finally {

// Close session and session factory

if (session != null) {

session.close();

}

if (sessionFactory != null) {

sessionFactory.close();

}

}

}

}\*/

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

// Create Configuration instance and configure settings from hibernate.cfg.xml

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

// configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Employee.**class**);

configuration.addAnnotatedClass(Account.**class**);

// Build SessionFactory

SessionFactory sessionFactory = configuration.buildSessionFactory();

Session session = **null**;

**try** {

// Open session

session = sessionFactory.openSession();

// Retrieve Department by primary key

Department dept = (Department) session.get(Department.**class**, "D-111");

**if** (dept != **null**) {

System.***out***.println("Department Details");

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

System.***out***.println("Department Id : " + dept.getDid());

System.***out***.println("Department Name : " + dept.getDname());

Set<Employee> emps = dept.getEmps();

System.***out***.println("EID\tENAME\tESAL\tEADDR");

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

**for** (Employee emp : emps) {

System.***out***.print(emp.getEno() + "\t");

System.***out***.print(emp.getEname() + "\t");

System.***out***.print(emp.getEsal() + "\t");

System.***out***.print(emp.getEaddr() + "\n");

}

System.***out***.println("Employees retrieved successfully");

} **else** {

System.***out***.println("No department found with ID: D-111");

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// Close session and session factory

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

**3.** **Many-To-One Association:**

**------------------------------------------**

**It is a relation between entity classes, where multiple instances of an entity should be mapped with exactly single instance of another entity.**

**EX: Multiple Students have joined with single branch:**

**Annotations support for Many-To-One Assosiation**

**------------------------------------------------------**

**To represent many-to-one association, JPA has provided a seperate annotation like@ManyToOne(calcade=CascadeType.ALL)**

**Student.java**

**—----------------**

**package com.codegnan.entity;**

**import javax.persistence.CascadeType;**

**import javax.persistence.Column;**

**import javax.persistence.Entity;**

**import javax.persistence.Id;**

**import javax.persistence.ManyToOne;**

**import javax.persistence.Table;**

**@Entity**

**@Table(name = "student")**

**public class Student {**

**@Id**

**@Column(name = "sid", length = 5)**

**private String sid;**

**@Column(name = "sname", length = 10)**

**private String sname;**

**@Column(name = "saddr", length = 10)**

**private String saddr;**

**@ManyToOne(cascade = CascadeType.*ALL*)**

**private Branch branch;**

**public Student() {**

**super();**

**// TODO Auto-generated constructor stub**

**}**

**public Student(String sid, String sname, String saddr, Branch branch) {**

**super();**

**this.sid = sid;**

**this.sname = sname;**

**this.saddr = saddr;**

**this.branch = branch;**

**}**

**public String getSid() {**

**return sid;**

**}**

**public void setSid(String sid) {**

**this.sid = sid;**

**}**

**public String getSname() {**

**return sname;**

**}**

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

**this.sname = sname;**

**}**

**public String getSaddr() {**

**return saddr;**

**}**

**public void setSaddr(String saddr) {**

**this.saddr = saddr;**

**}**

**public Branch getBranch() {**

**return branch;**

**}**

**public void setBranch(Branch branch) {**

**this.branch = branch;**

**}**

**@Override**

**public String toString() {**

**return "Student [sid=" + sid + ", sname=" + sname + ", saddr=" + saddr + ", branch=" + branch + "]";**

**}**

**}**

**Branch.java**

**----------------**

**package** com.codegnan.entity;

**import** javax.persistence.Column;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "branch")

**public** **class** Branch {

@Id

@Column(name = "bid", length = 7)

**private** String bid;

@Column(name = "bname", length = 10)

**private** String bname;

**public** Branch() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** Branch(String bid, String bname) {

**super**();

**this**.bid = bid;

**this**.bname = bname;

}

**public** String getBid() {

**return** bid;

}

**public** **void** setBid(String bid) {

**this**.bid = bid;

}

**public** String getBname() {

**return** bname;

}

**public** **void** setBname(String bname) {

**this**.bname = bname;

}

@Override

**public** String toString() {

**return** "Branch [bid=" + bid + ", bname=" + bname + "]";

}

}

clientApp.java

=============

**package** com.codegnan.test;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.entity.Branch;

**import** com.codegnan.entity.Student;

/\*

public class ClientApp {

public static void main(String[] args) {

// Create Configuration instance and configure settings from hibernate.cfg.xml

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Student.**class**);

configuration.addAnnotatedClass(Branch.**class**);

// Build SessionFactory

SessionFactory sessionFactory = configuration.buildSessionFactory();

Session session = null;

try {

// Open session

session = sessionFactory.openSession();

// Create Branch object

Branch branch = new Branch();

branch.setBid("B-111");

branch.setBname("CS");

// Create Student objects

Student std1 = new Student();

std1.setSid("S-111");

std1.setSname("AAA");

std1.setSaddr("Hyd");

std1.setBranch(branch);

Student std2 = new Student();

std2.setSid("S-222");

std2.setSname("BBB");

std2.setSaddr("Hyd");

std2.setBranch(branch);

Student std3 = new Student();

std3.setSid("S-333");

std3.setSname("CCC"); // Changed name to avoid duplication

std3.setSaddr("Hyd");

std3.setBranch(branch);

// Begin transaction

Transaction tx = session.beginTransaction();

session.save(branch); // Save the branch first to maintain referential integrity

session.save(std1);

session.save(std2);

session.save(std3);

tx.commit();

System.out.println("Students are stored successfully");

} catch (Exception e) {

// Rollback transaction if an exception occurs

if (session.getTransaction() != null) {

session.getTransaction().rollback();

}

e.printStackTrace();

} finally {

// Close session and session factory

if (session != null) {

session.close();

}

if (sessionFactory != null) {

sessionFactory.close();

}

}

}

}\*/

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

// Create Configuration instance and configure settings from hibernate.cfg.xml

// Create Configuration instance and configure settings from hibernate.cfg.xml

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

// configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Student.**class**);

configuration.addAnnotatedClass(Branch.**class**);

// Build SessionFactory

SessionFactory sessionFactory = configuration.buildSessionFactory();

Session session = **null**;

**try** {

// Open session

session = sessionFactory.openSession();

// Retrieve and display student details

**for** (String studentId : **new** String[]{"S-111", "S-222", "S-333"}) {

Student student = (Student) session.get(Student.**class**, studentId);

**if** (student != **null**) {

Branch branch = student.getBranch();

System.***out***.println("Student Details");

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

System.***out***.println("Student Id :" + student.getSid());

System.***out***.println("Student Name :" + student.getSname());

System.***out***.println("Student Address :" + student.getSaddr());

System.***out***.println("Branch Id :" + branch.getBid());

System.***out***.println("Branch Name :" + branch.getBname());

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

} **else** {

System.***out***.println("Student with ID " + studentId + " not found.");

}

}

System.***out***.println("Records retrieved successfully");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

// Close session and session factory

**if** (session != **null**) {

session.close();

}

**if** (sessionFactory != **null**) {

sessionFactory.close();

}

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-05</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-05</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>

Annotation Support for many-to-many association

=======================================

Student.java

==============

**package** com.codegnan.entity;

**import** javax.persistence.\*;

**import** java.util.Set;

@Entity

@Table(name = "student")

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

@Id

@Column(name = "SID", length = 5)

**private** String sid;

@Column(name = "SNAME", length = 10)

**private** String sname;

@Column(name = "SADDR", length = 10)

**private** String saddr;

@ManyToMany(cascade = CascadeType.***ALL***)

@JoinTable(name = "student\_course", joinColumns = @JoinColumn(name = "sid"), inverseJoinColumns = @JoinColumn(name = "cid"))

**private** Set<Course> courses;

**public** Student() {

// Default constructor

}

**public** Student(String sid, String sname, String saddr, Set<Course> courses) {

**this**.sid = sid;

**this**.sname = sname;

**this**.saddr = saddr;

**this**.courses = courses;

}

// Getters and setters

**public** String getSid() {

**return** sid;

}

**public** **void** setSid(String sid) {

**this**.sid = sid;

}

**public** String getSname() {

**return** sname;

}

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

**this**.sname = sname;

}

**public** String getSaddr() {

**return** saddr;

}

**public** **void** setSaddr(String saddr) {

**this**.saddr = saddr;

}

**public** Set<Course> getCourses() {

**return** courses;

}

**public** **void** setCourses(Set<Course> courses) {

**this**.courses = courses;

}

@Override

**public** String toString() {

**return** "Student [sid=" + sid + ", sname=" + sname + ", saddr=" + saddr + ", courses=" + courses + "]";

}

}

Course.java

=============

**package** com.codegnan.entity;

**import** javax.persistence.\*;

**import** java.util.Set;

@Entity

@Table(name = "course")

**public** **class** Course {

@Id

@Column(name = "CID", length = 5)

**private** String cid;

@Column(name = "CNAME", length = 10)

**private** String cname;

@Column(name = "CCOST", length = 5)

**private** **int** ccost;

@ManyToMany(mappedBy = "courses", cascade = CascadeType.***ALL***)

**private** Set<Student> students;

**public** Course() {

// Default constructor

}

**public** Course(String cid, String cname, **int** ccost) {

**this**.cid = cid;

**this**.cname = cname;

**this**.ccost = ccost;

}

// Getters and setters

**public** String getCid() {

**return** cid;

}

**public** **void** setCid(String cid) {

**this**.cid = cid;

}

**public** String getCname() {

**return** cname;

}

**public** **void** setCname(String cname) {

**this**.cname = cname;

}

**public** **int** getCcost() {

**return** ccost;

}

**public** **void** setCcost(**int** ccost) {

**this**.ccost = ccost;

}

**public** Set<Student> getStudents() {

**return** students;

}

**public** **void** setStudents(Set<Student> students) {

**this**.students = students;

}

@Override

**public** String toString() {

**return** "Course [cid=" + cid + ", cname=" + cname + ", ccost=" + ccost + "]";

}

}

ClientApp.java

===================

**package** com.codegnan.test;

**import** java.util.HashSet;

**import** java.util.Set;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**import** com.codegnan.beans.Course;

**import** com.codegnan.beans.Student;

**public** **class** ClientApp {

**public** **static** **void** main(String[] args) {

Configuration configuration = **new** Configuration();

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost:3306/jfs");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.setProperty("hibernate.hbm2ddl.auto", "create");

configuration.addAnnotatedClass(Student.**class**);

configuration.addAnnotatedClass(Course.**class**);

SessionFactory sessionFactory = **null**;

Session session = **null**;

Transaction transaction = **null**;

**try** {

sessionFactory = configuration.buildSessionFactory();

session = sessionFactory.openSession();

transaction = session.beginTransaction();

Course c1 = **new** Course("c-111", "c", 500);

Course c2 = **new** Course("c-222", "c++", 700);

Course c3 = **new** Course("c-333", "java", 1000);

Set<Course> courses = **new** HashSet<>();

courses.add(c1);

courses.add(c2);

courses.add(c3);

Student std1 = **new** Student("s-111", "AAA", "hyd", courses);

Student std2 = **new** Student("s-222", "BBB", "Vjy", courses);

Student std3 = **new** Student("s-333", "CCC", "bng", courses);

session.save(std1);

session.save(std2);

session.save(std3);

transaction.commit();

System.***out***.println("Students inserted succesfully");

} **catch** (Exception e) {

e.printStackTrace();

}**finally** {

**if**(session!=**null**) {

session.close();

}**if**(sessionFactory!=**null**) {

sessionFactory.close();

}

}

}

}

**Hibernate Employee Curd Operations**

**=============================================**

**Employee.java**

**======================**

**package com.codegnan.hibernateapp.withoutxml;**

**import javax.persistence.Column;**

**import javax.persistence.Entity;**

**import javax.persistence.Id;**

**import javax.persistence.Table;**

**//@Entity is for mapping the class with DB table**

**@Entity**

**/\***

**\* @Table is optional, its useful when Table name and classname are different**

**\*/**

**@Table(name = "employee1")**

**public class Employee {**

**@Id**

**@Column(name = "eid")**

**private int eid;**

**@Column(name = "ename")**

**private String ename;**

**@Column(name = "esal")**

**private float esal;**

**@Column(name = "eaddr")**

**private String eaddr;**

**public int getEid() {**

**return eid;**

**}**

**public void setEid(int eid) {**

**this.eid = eid;**

**}**

**public String getEname() {**

**return ename;**

**}**

**public void setEname(String ename) {**

**this.ename = ename;**

**}**

**public float getEsal() {**

**return esal;**

**}**

**public void setEsal(float esal) {**

**this.esal = esal;**

**}**

**public String getEaddr() {**

**return eaddr;**

**}**

**public void setEaddr(String eaddr) {**

**this.eaddr = eaddr;**

**}**

**@Override**

**public String toString() {**

**return "Employee [eid=" + eid + ", ename=" + ename + ", esal=" + esal + ", eaddr=" + eaddr + "]";**

**}**

**}**

EmployeeCRUD.java

========================

**package** com.codegnan.test;

**import** java.util.Scanner;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

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

**import** org.hibernate.service.ServiceRegistry;

**import** com.codegnan.beans.Employee;

**public** **class** EmployeeCRUD {

**private** **static** SessionFactory *sessionFactory*;// declare sessionFactory

**public** **static** **void** main(String[] args) {

// initilize hibernate sessionFactory

*sessionFactory* = *createSessionFactory*();

// start the CRUD Operations

EmployeeCRUD employeeCRUD = **new** EmployeeCRUD();

employeeCRUD.run();

*sessionFactory*.close();

}

**public** **static** SessionFactory createSessionFactory() {

// create configuration Object

Configuration configuration = **new** Configuration();

// set hibernate properties.

configuration.setProperty("hibernate.dialect", "org.hibernate.dialect.MySQL5Dialect");

configuration.setProperty("hibernate.connection.driver\_class", "com.mysql.cj.jdbc.Driver");

configuration.setProperty("hibernate.connection.url", "jdbc:mysql://localhost/hibernatedb");

configuration.setProperty("hibernate.connection.username", "root");

configuration.setProperty("hibernate.connection.password", "root");

configuration.setProperty("hibernate.show\_sql", "true");

configuration.setProperty("hibernate.hbm2ddl.auto", "update");

configuration.addAnnotatedClass(Employee.**class**);

ServiceRegistry serviceRegistry = **new** StandardServiceRegistryBuilder()

.applySettings(configuration.getProperties()).build();

**return** configuration.buildSessionFactory(serviceRegistry);

}

**public** **void** run() {

Scanner scanner = **new** Scanner(System.***in***);

**int** choice;

**do** {

System.***out***.println("Choose An Operation ");

System.***out***.println("1. create Employee");

System.***out***.println("2. Read Employee");

System.***out***.println("3. Update Employee");

System.***out***.println("4. Delete Employee");

System.***out***.println("5. Exit");

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

System.***out***.println("Enter your Choice");

choice = scanner.nextInt();

**switch** (choice) {

**case** 1:

createEmployee(scanner);// call method create an employee

**break**;

**case** 2:

readEmployee(scanner);// call method to read an employee

**break**;

**case** 3:

updateEmployee(scanner);// call method to update an employee

**break**;

**case** 4:

deleteEmployee(scanner);// call method to delee an employee

**break**;

**case** 5:

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

**break**;

**default**:

System.***out***.println("Invalid Choice! Please try again");

}

} **while** (choice != 5);

scanner.close();

}

**public** **void** createEmployee(Scanner scanner) {

Session session = *sessionFactory*.openSession();

Transaction transaction = **null**;

**try** {

transaction = session.beginTransaction();

Employee employee = **new** Employee();

System.***out***.print("Enter Employee Id : ");

employee.setEid(scanner.nextInt());

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

employee.setEname(scanner.next());

System.***out***.println("Enter Employee Salary : ");

employee.setEsal(scanner.nextFloat());

System.***out***.println("Enter Employee Address : ");

employee.setEaddr(scanner.next());

session.save(employee);

transaction.commit();

System.***out***.println("Employee created succesfully");

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **void** readEmployee(Scanner scanner) {

Session session = *sessionFactory*.openSession();

**try** {

System.***out***.print("Enter employee Id to Read : ");

**int** eid = scanner.nextInt();

Employee employee = session.get(Employee.**class**, eid);

**if** (employee != **null**) {

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

} **else** {

System.***out***.println("Employee not found with ID :" + eid);

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **void** updateEmployee(Scanner scanner) {

Session session = *sessionFactory*.openSession();

**try** {

Transaction transaction = session.beginTransaction();

System.***out***.print("Enter Employee id to Update : ");

**int** eid = scanner.nextInt();

Employee employee = session.get(Employee.**class**, eid);

**if** (employee != **null**) {

scanner.nextLine();

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

employee.setEname(scanner.next());

System.***out***.println("Enter new Employee salary : ");

employee.setEsal(scanner.nextFloat());

System.***out***.println("Enter new Employee Address : ");

employee.setEaddr(scanner.next());

session.update(employee);

transaction.commit();

System.***out***.println("Employee Updated Succesfully ");

} **else** {

System.***out***.println("Employee Not Found With Id : " + eid);

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

session.close();

}

}

**public** **void** deleteEmployee(Scanner scanner) {

Session session = *sessionFactory*.openSession();

**try** {

Transaction transaction = session.beginTransaction();

System.***out***.print("Enter Employee Id To Delete : ");

**int** eid = scanner.nextInt();

Employee employee = session.get(Employee.**class**, eid);

**if** (employee != **null**) {

session.delete(employee);

transaction.commit();

System.***out***.println("Employee record deleted Successfully....");

} **else** {

System.***out***.println("Employee Not Found With id : " + eid);

}

} **catch** (Exception e) {

e.printStackTrace();

} **finally** {

session.close();

}

}

}

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>com.codegnan</groupId>

<artifactId>hibernate-08</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>hibernate-08</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<version>8.0.33</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.36.Final</version>

</dependency>

</dependencies>

</project>