Hibernate Example

In this chapter, we will create our first Hibernate Example.

**Question**: we will create Employee Object using Hibernate and insert into Database?

**Requirements:**

**Database :** Throughout this tutorial, we will use HSQLDb. HSQL is an in-memory database. We are using HSQL because we don’t need to install any database like MySql,Oracle. In memory, the database is such a database which stores data on main memory and taken action on them once the program has terminated it flushes data from main memory so data are not persistent unlike database like Mysql, Oracle. Mysql Oracle store data on disk storage.

So every time we will perform any CRUD operation we have to insert data first then do the operations on it.

**Maven** : Hibernate requires many dependencies and Each dependency has transitive dependencies so we will use Maven as build tools so it can take care of the dependencies and resolves it automatically.

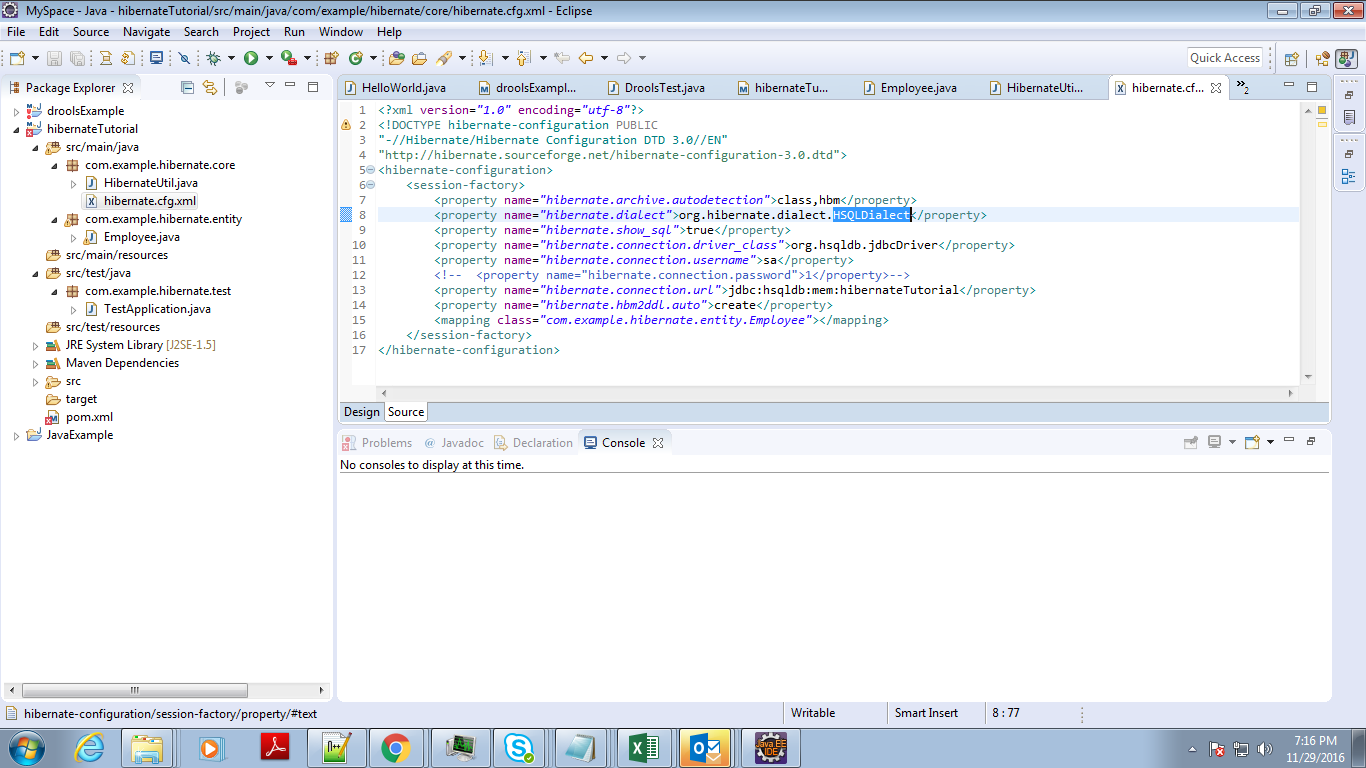
**Eclipse :** we will use **Eclipse Oxygen** as our editor.

**Project SetUp:**

Create a maven Project as per instruction given in previous chapter.

**Project Structure:**

Please find the project Structure



**Main Components :**

1. **hibernate.cfg.xml :** This is the configuration file. All Hibernate related configuration goes here.

**hibernate.dialect** : dialect property determine which underlying database the application uses. So it can change the hql to that db specific sql.

**hibernate.show\_sql** : If value is true, hibernate generated sql query are printed in console if false then it is not printed.

**hibernate.connection.driver\_class :** database specific driver class which will load by class loader on application startup**.**

**hibernate.connection.username :** database user name.

**hibernate.connection.password:** database password

**hibernate.connection.url** : Connection Url to connect the database.

**mapping class :** mapping all the Hibernate Entity classes here I map the Employee entity.

**Full hibernate.cfg.xml class :**

<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE hibernate-configuration PUBLIC

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

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.archive.autodetection">class,hbm</property>

<property name="hibernate.dialect">org.hibernate.dialect.HSQLDialect</property>

<property name="hibernate.show\_sql">true</property>

<property name="hibernate.connection.driver\_class">org.hsqldb.jdbcDriver</property>

<property name="hibernate.connection.username">sa</property>

<property name="hibernate.connection.url">jdbc:hsqldb:mem:hibernateTutorial</property>

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

<mapping class="com.example.hibernate.entity.Employee"></mapping>

</session-factory>

</hibernate-configuration>

2.  **Entity Object** : In Hibernate generally for each table, we create an Entity object where an entity object is nothing but a POJO. As we use Annotation here we also do the mapping here, In hibernate we map the table with the Entity object and it column with the properties of Entity Objects.

In later chapter, we will discuss about each annotation for now just look how an Entity Object looks like.

**Employee.java**

package com.example.hibernate.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.UniqueConstraint;

@Entity

@Table(name = "Employee", uniqueConstraints = {@UniqueConstraint(columnNames = "ID"), @UniqueConstraint(columnNames = "EMAIL")})

public class Employee

{

@Id

@Column(name = "ID", unique = true, nullable = false)

private Integer empId;

@Column(name = "EMAIL", unique = true, nullable = false, length = 100)

private String email;

@Column(name = "NAME", unique = false, nullable = false, length = 100)

private String name;

public Integer getEmpId()

{

return empId;

}

public void setEmpId(Integer empId)

{

this.empId = empId;

}

public String getEmail()

{

return email;

}

public void setEmail(String email)

{

this.email = email;

}

public String getName()

{

return name;

}

public void setName(String name)

{

this.name = name;

}

@Override

public String toString() {

return "Employee [empId=" + empId + ", email=" + email + ", name=" + name + "]";

}

}

**3. Hibernate Session and SessionFactory :**

Sessionfactory and Session are two main components of Hibernate. From hibernate.cfg.xml we configure the Sessionfactory. SessionFactory holds all the configuration details, mapping resources so it is very heavy. From Session factory we can create Session , Session is light weight and we perform all the operation upon session. IT is the heart of an application object, Using Session Object we can perform CRUD operation on the entity, it manages the transaction, session caching the entities so it is the pivot element in Hibernate.

**HibernateUtil.java**

package com.example.hibernate.core;

import org.hibernate.SessionFactory;

import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

public class HibernateUtil

{

private static SessionFactory sessionFactory = initFactory();

private static SessionFactory initFactory()

{

try

{

if (sessionFactory == null)

{

Configuration configuration = new Configuration().configure(HibernateUtil.class.getResource("hibernate.cfg.xml"));

StandardServiceRegistryBuilder serviceRegistryBuilder = new StandardServiceRegistryBuilder();

serviceRegistryBuilder.applySettings(configuration.getProperties());

ServiceRegistry serviceRegistry = serviceRegistryBuilder.build();

sessionFactory = configuration.buildSessionFactory(serviceRegistry);

}

return sessionFactory;

} catch (Throwable ex)

{

System.err.println("Initial SessionFactory creation failed." + ex);

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory()

{

return sessionFactory;

}

public static void shutdown()

{

getSessionFactory().close();

}

}

4. **Test The Application :** Hooray**,** We successfully configure our first example now we need to create some test data and insert that data into HSQL in-memory database.

**TestApplication.java**

package com.example.hibernate.test;

import org.hibernate.Session;

import com.example.hibernate.core.HibernateUtil;

import com.example.hibernate.entity.Employee;

public class TestApplication {

public static void main(String[] args) {

TestApplication app = new TestApplication();

app.insertEmployee();

app.findEmployeeById(1);

HibernateUtil.shutdown();

}

public void insertEmployee() {

Session session = HibernateUtil.getSessionFactory().openSession();

session.beginTransaction();

Employee emp = new Employee();

emp.setEmpId(1);

emp.setEmail("mitrashamik@mail.com");

emp.setName("Shamik Mitra");

session.save(emp);

session.getTransaction().commit();

}

public void findEmployeeById(Integer id) {

Session session = HibernateUtil.getSessionFactory().openSession();

Employee emp2 = (Employee) session.get(Employee.class, id);

System.out.println("Successfully Fetch " + emp2);

}

}

**Output:**

Hibernate: drop table Employee if exists

Hibernate: create table Employee (ID integer not null, EMAIL varchar(100) not null, NAME varchar(100) not null, primary key (ID))

Hibernate: alter table Employee add constraint UK\_ardf0f11mfa6tujs3hflthwdv unique (EMAIL)

Hibernate: insert into Employee (EMAIL, NAME, ID) values (?, ?, ?)

Hibernate: select employee0\_.ID as ID1\_0\_0\_, employee0\_.EMAIL as EMAIL2\_0\_0\_, employee0\_.NAME as NAME3\_0\_0\_ from Employee employee0\_ where employee0\_.ID=?

Successfully Fetch Employee [empId=1, email=mitrashamik@mail.com, name=Shamik Mitra]