**Db : MySql**

**Hibernate 4+**

**Install maven from http://download.eclipse.org/technology/m2e/releases/**

**Create web development project**

**Configure it to maven**

**Refer hibernate example diagram**

**1.create database onlinetutorialspoint;**

**2.use onlinetutorialspoint;**

**3.**CREATE TABLE `student` (

`id` INT(10) NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NULL DEFAULT NULL,

`rollnumber` INT(10) NULL DEFAULT NULL,

`gender` TINYINT(4) NULL DEFAULT NULL,

`class` VARCHAR(50) NULL DEFAULT NULL,

`lastupdated` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

PRIMARY KEY (`id`)

);

**4.open 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>CrudHibernate</groupId>

<artifactId>CrudHibernate</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>war</packaging>

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.0.1.Final</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-validator</artifactId>

<version>4.1.0.Final</version>

</dependency>

<dependency>

<groupId>org.hibernate.common</groupId>

<artifactId>hibernate-commons-annotations</artifactId>

<version>4.0.1.Final</version>

<classifier>tests</classifier>

</dependency>

<dependency>

<groupId>org.hibernate.javax.persistence</groupId>

<artifactId>hibernate-jpa-2.0-api</artifactId>

<version>1.0.1.Final</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>4.1.8.Final</version>

</dependency>

<dependency>

<groupId>javax.validation</groupId>

<artifactId>validation-api</artifactId>

<version>1.0.0.GA</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.6.4</version>

</dependency>

<dependency>

<groupId>org.jboss.logging</groupId>

<artifactId>jboss-logging</artifactId>

<version>3.1.0.CR2</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-log4j12</artifactId>

<version>1.6.4</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

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

<version>5.1.10</version>

</dependency>

</dependencies>

<build>

<sourceDirectory>src</sourceDirectory>

<plugins>

<plugin>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.5.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<artifactId>maven-war-plugin</artifactId>

<version>3.0.0</version>

<configuration>

<warSourceDirectory>WebContent</warSourceDirectory>

</configuration>

</plugin>

</plugins>

</build>

</project>

Note : try clean build after adding maven dependencies

Maven - update project

**5.create hibernate configuration file hibernate.cfg.xml (paste this file to source file)**

<?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.dialect">org.hibernate.dialect.MySQLDialect</property>

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

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/onlinetutorialspoint?zeroDateTimeBehavior=convertToNull</property>

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

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

<property name="hibernate.connection.autocommit">true</property>//else not saved

<mapping class="com.worldclock.pojo.Student" />

</session-factory>

</hibernate-configuration>

**6.HibernateConnector.java , loads hibernate.cfg.xml configuration**

package com.worldclock.config;

import org.hibernate.HibernateException;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.cfg.Configuration;

import org.hibernate.service.ServiceRegistry;

import org.hibernate.service.ServiceRegistryBuilder;

public class HibernateConnector {

private static HibernateConnector hc;

private SessionFactory sessionFactory;

private ServiceRegistry serviceRegistry;

Configuration configuration;

private HibernateConnector() throws HibernateException {

try {

configuration = new Configuration();

configuration.configure();

serviceRegistry = new ServiceRegistryBuilder().applySettings(

configuration.getProperties()).buildServiceRegistry();

sessionFactory = configuration.buildSessionFactory(serviceRegistry);

} catch (Throwable ex) {

System.err.println("Failed to create sessionFactory object." + ex);

throw new ExceptionInInitializerError(ex);

}

}

public static synchronized HibernateConnector getInstance() throws HibernateException {

if (hc == null) {

hc = new HibernateConnector();

}

return hc;

}

public Session getSession() throws HibernateException {

Session session = sessionFactory.openSession();

if (!session.isConnected()) {

this.reconnect();

}

return session;

}

private void reconnect() throws HibernateException {

this.sessionFactory = configuration.buildSessionFactory(serviceRegistry);;

}

}

**7.Student.java pojo**

package com.worldclock.pojo;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import static javax.persistence.GenerationType.IDENTITY;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "student", catalog = "onlinetutorialspoint"

)

public class Student implements java.io.Serializable {

private Integer id;

private String name;

private Integer rollnumber;

private Byte gender;

private String class\_;

public Student() {

}

public Student(String name, Integer rollnumber, Byte gender, String class\_) {

this.name = name;

this.rollnumber = rollnumber;

this.gender = gender;

this.class\_ = class\_;

}

@Id

@GeneratedValue(strategy = IDENTITY)

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

public Integer getId() {

return this.id;

}

public void setId(Integer id) {

this.id = id;

}

@Column(name = "name", length = 50)

public String getName() {

return this.name;

}

public void setName(String name) {

this.name = name;

}

@Column(name = "rollnumber")

public Integer getRollnumber() {

return this.rollnumber;

}

public void setRollnumber(Integer rollnumber) {

this.rollnumber = rollnumber;

}

@Column(name = "gender")

public Byte getGender() {

return this.gender;

}

public void setGender(Byte gender) {

this.gender = gender;

}

@Column(name = "class", length = 50)

public String getClass\_() {

return this.class\_;

}

public void setClass\_(String class\_) {

this.class\_ = class\_;

}

}

**8.StudentDAO.java data access object , which will interact with db using query interface methods**

package com.worldclock.dao;

import com.worldclock.config.HibernateConnector;

import com.worldclock.pojo.Student;

import java.util.List;

import org.hibernate.Query;

import org.hibernate.Session;

import org.hibernate.Transaction;

public class StudentDAO {

public List<Student> listStudent() {

Session session = null;

try {

session = HibernateConnector.getInstance().getSession();

Query query = session.createQuery("from Student s");

List queryList = query.list();

if (queryList != null && queryList.isEmpty()) {

return null;

} else {

System.out.println("list " + queryList);

return (List<Student>) queryList;

}

} catch (Exception e) {

e.printStackTrace();

return null;

} finally {

session.close();

}

}

public Student findStudentById(int id) {

Session session = null;

try {

session = HibernateConnector.getInstance().getSession();

Query query = session.createQuery("from Student s where s.id = :id");

query.setParameter("id", id);

List queryList = query.list();

if (queryList != null && queryList.isEmpty()) {

return null;

} else {

return (Student) queryList.get(0);

}

} catch (Exception e) {

e.printStackTrace();

return null;

} finally {

session.close();

}

}

public void updateStudent(Student student) {

Session session = null;

try {

session = HibernateConnector.getInstance().getSession();

session.saveOrUpdate(student);

session.flush();

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

}

}

public Student addStudent(Student student) {

Session session = null;

Transaction transaction = null;

try {

session = HibernateConnector.getInstance().getSession();

System.out.println("session : "+session);

transaction = session.beginTransaction();

session.save(student);

transaction.commit();

return student;

} catch (Exception e) {

e.printStackTrace();

return null;

}

}

public int deleteStudent(int id) {

Query createQuery = null;

Session session = null;

int deletedResult = -1;

try {

session = HibernateConnector.getInstance().getSession();

Transaction beginTransaction = session.beginTransaction();

createQuery = session.createQuery("delete from Student s where s.id =:id");

createQuery.setParameter("id", id);

deletedResult = createQuery.executeUpdate();

beginTransaction.commit();

return deletedResult;

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

return deletedResult;

}

}

}

**9.DbOperations.java Tester class, used for crud**

package com.worldclock.service;

import com.worldclock.dao.StudentDAO;

import com.worldclock.pojo.Student;

import java.util.List;

public class DbOperations {

StudentDAO studentDAO = new StudentDAO();

public static void main(String[] args) {

DbOperations dbOperations = new DbOperations();

//insert

//Student createStudent = dbOperations.createStudent();

//read

/\*List<Student> studentList = dbOperations.getStudentList();

if (studentList != null) {

for (Student student : studentList) {

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

}

}\*/

//update

/\*dbOperations.updateStudent(4);//input is id

Student student = dbOperations.getStudent(4);//input is 4

if (student != null) {

System.out.println("Student Details After Updation : " + student.getName());

}\*/

//delete

dbOperations.deleteStudent(4);

}

public Student createStudent() {

Student s = new Student();

s.setGender(new Byte("2"));

s.setName("sahan");

s.setClass\_("12");

s.setRollnumber(067);

studentDAO.addStudent(s);

return s;

}

public void updateStudent(Integer id) {

Student student = studentDAO.findStudentById(id);

student.setName("srujan");

studentDAO.updateStudent(student);

System.out.println("Student Updated Success");

}

public void deleteStudent(Integer id) {

int deletedResult = studentDAO.deleteStudent(id);

if(deletedResult == -1){

System.out.println("some error");

}

if(deletedResult == 0){

System.out.println("deletion failed");

}

if(deletedResult == 1){

System.out.println("deletion success");

}

public List<Student> getStudentList() {

return studentDAO.listStudent();

}

public Student getStudent(Integer id) {

return studentDAO.findStudentById(id);

}

}

Note : add --add-modules java.xml.bind Module while running Main method.

Add above module in run configuration -> Arguments -> VM arguments

**Hibernate flush vs commit**

**Flush :** compare previous data with updated data and synchronise. But data not saved to database.

**Commit :** compare previous data with updated data, synchronize and data saved to database.

**Commit -> flush() + commit()**

session.flush() vs transaction.commit();

Transaction begins with session.beginTransaction();

Flush will execute statements, but will not commit().

To save data call commit after flush.

Commit will save data into database, commit will call internally flush before committing.

flush() - can be rollback

commit() - can’t be rollback

**DAO : updateStudent() using transaction commit**

public void updateStudent(Student student) {

Session session = null;

Transaction transaction = null;

try {

session = HibernateConnector.getInstance().getSession();

transaction = session.beginTransaction();

session.saveOrUpdate(student);

//session.flush();

transaction.commit();

} catch (Exception e) {

e.printStackTrace();

} finally {

session.close();

}

}