**Hibernate**

Draw Backs of JDBC:

* In JDBC, if we open a database connection we need to write in try, and if any exceptions occurred catch block will takers about it, and finally used to close the connections.
* here as a programmer we must close the connection, or we may get a chance to get our of connections message…!
* Actually if we didn’t close the connection in the finally block, then jdbc doesn’t responsible to close that connection.
* In JDBC we need to write Sql commands in various places, after the program has created if the table structure is modified then the JDBC program doesn’t work, again we need to modify and compile and re-deploy required, which is tedious.
* JDBC used to generate database related error codes if an exception will occurs, but java programmers are unknown about this error codes right.
* In the Enterprise applications, the data flow with in an application from class to class will be in the form of objects, but while storing data finally in a database using JDBC then that object will be converted into text.  Because JDBC doesn’t transfer objects directly.

In order to overcome above problems,  Hibernate came into picture..!

# Prerequisites:

# Good understanding of the Java programming language. A basic understanding of relational databases, JDBC and SQL

# ****What is Hibernate?****

# ****Hibernate**** is a powerful and high-performance ****ORM**** service.

Hibernate maps Java classes to database tables and from Java data types to SQL data types.

Hibernate framework simplifies the development of java application to interact with the database.

Hibernate maps the Java classes to the database tables.

It also provides the data query and retrieval facilities that significantly reduces the development time.

# ****ORM****  "Object-relational Mapping".

# It is simply ****storing a Java object into a (relational) database****. When stored, Hibernate automatically maps (places) and stores each instance variable value of a Java object in a database column.

# For example, emp object of ****Employee**** class has three properties (instance variables) like ****emp.empid****, ****emp.name**** and ****emp.department****. When the object ****emp**** is stored in the database ****table**** EMPLOYEE, Hibernate stores these variables values in the columns ****EMPID****, ****NAME**** and ****DEPARTMENT**** of a database table EMPLOYEE. This reduces the code of JDBC to a maximum extent.

# ORM Overview

Object Oriented Programming use Classes whereas Relational Database use tables. In programming this will creates a gap. This gap is called impedance mismatch.

We can bridge the gap between Object oriented model and Relational model is known as Object Relational Mapping(ORM)

ORM is a mapping between Object oriented model and Relational model.

# ****The Object-Relational Impedance Mismatch:****

Object-Relational Impedance Mismatch' (sometimes called the 'paradigm mismatch') is just a fancy way of saying that object models and relational models do not work very well together. RDBMSs represent data in a tabular format , whereas object-oriented languages, such as Java, represent it as an interconnected graph of objects.

# ****Solution of Impedance Mismatch:****

Use an ORM mapping tool which will provide a simple API for storing and retrieving Java objects directly from database.  
There are several good ORM tools available in market that will do mapping between Objects and Database tables and solve the impedance mismatch.

Following are ORM tools:

1. Hibernate
2. OpenJPA
3. EJB Entity Bean
4. EclipseLink
5. TopLink

# Java ****variables vs table columns**** mapping is done in an ****XML**** file (like web.xml, in case of servlets, where an alias name is mapped to the actual servlet).

# **Hibernate** is Ob**ject/Relational persistence** for **Java** and **.NET** technologies. Here, **persistence means writing Java object to a permanent storage and making it to live (stay) long**.

# Always persistence storage comes with writing to a hard disk like to a text file or a RDBMS table.

# Remember DS like an ****ArrayList**** or ****Vector**** gives a temporary storage (available to program until program terminates. Once the program execution is over, the data is lost and when you restart the same program, the data is not available) of data as the data is written to the RAM.

# **Object/Relational means storing or writing a Java object to a database (to be more precisely, relational database) like Oracle**.

# Storing the variables of an object is known as storing the ****state**** of the object or persisting the object.

# Hibernate is an open source code that can be freely downloaded from ****www.hibernate.org**** and used.

# Hibernate stable versions

# Hibernate 4.2.21.Final,

# Hibernate 4.3.11.Final

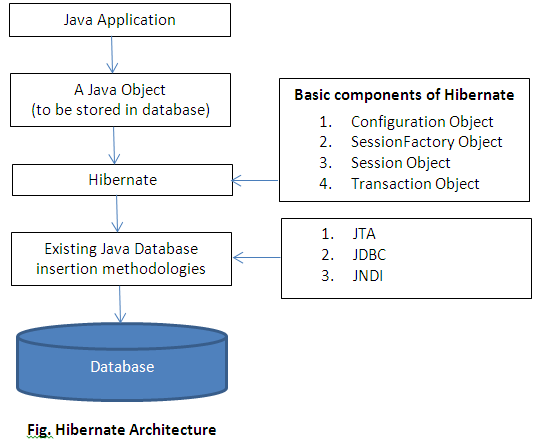
# Hibernate 5.0.12.Final

# Hibernate 5.1.4.Final

# Hibernate 5.2.7.Final

**ORM** is a supplement to **JDBC** and not a replacement. In fact, **Hibernate internally uses JDBC**. The aim of ORM is to reduce the writing of number of lines of DB programming code in Java.

**Hibernate Architecture**



Hibernate uses JDBC internally to insert a Java object into database. The main objects of Hibernate used in coding are **Configuration**, **SessionFactory**, **Session** and **Transaction**. The insertion of an object into database is known as **persisting** (literal meaning: preserving) a Java object.

**JDBC vs Hibernate – Features of Hibernate (advantages of Hibernate over JDBC)**

**1. Transparent Persistence**

It is a very complex task to write an object of an Object-Oriented programming language to a relational database with JDBC as there will be a mismatch between the **data types** (say, int) of a Java class and **column types** (say, number) of a database, and moreover, column types vary with each database. Hibernate takes care of this transparently depending on the database used (internally) without any botheration to the programmer.

Writing a Java object to a database and reading back is known as **transparent persistence**. Transparent means unknown to the programmer, to say, the internal implementation is unknown

**Another example:**

What is happening internally to store a Java object into a table is completely transparent (not known or abstracted) to the programmer. Moreover

1. How Java types like **int** and String are converted into database specific **number** and **varchar2** are not known.
2. How a Java **object is mapped to a database** table is not known to the programmer.
3. What **SQL query** is used by the Hibernate is not known (of course, this can be known with extra line of code with the option show\_aql).
4. Programmer is not aware of the internal mechanism of maintaining the **transactions**, **catches** and fetches etc.

All the above transparent mechanism amounts to less JDBC code to the programmer.

**2. Object relation management (Class – Class mapping)**

Imagine there are two Java classes **Student** and **Address** with database tables **Student** and **Addressbook**. The primary key of **addressbook** is a foreign key in **student**. When a **Student** object is inserted in **Student** table, automatically the address should also be inserted in **addressbook** table. When a **Student** record is deleted, its corresponding address in the **addressbook** table should also be deleted and so also for updations.

**3. Auto schema/Table generation**

When a Java object is tried to save to a database table, if the table does not exist, **Hibernate** creates a new table with appropriate types (specific to that database) and inserts the object. In Hibernate, inserting object means writing each variable of the Java object to each column of a table. That is, every Java object corresponds to one record in a database table.

**4. Automatic Primary key generations**

Hibernate has its own style of **primary key generation** for the records inserted without depending on the database native algorithm. Hibernate comes with its own data types (which are independent of any database). If required, programmer can suggest Hibernate to depend on the native database style or any **sequence algorithm** etc.

**5. Support for Query Language (database independent queries)**

JDBC uses **SQL** (Structured Query Language) which may vary with the database. Programmer should write an effective way of query, keeping in mind of the performance (like by decreasing the number of database hits to execute the query). Even though, Hibernate supports native **SQL**, it comes with its own query language known as **HQL (Hibernate Query Language)** which is independent of any database. **HQL** syntax is similar to SQL syntax and supports polymorphic queries. To say, **HQL** gives database independent code (SQL is database dependent code). Hibernate Query Language permits to write a query on class objects.

**6. Maintenance Cost**

JDBC programmer should write a lot of code to create Java persistent objects out of the **Resultset** object, if required. He may be required to write "emp.ename = res.getString(“empname”)" etc. where **emp** is an object of **Employee** class with variable ename. That is, the programmer should map manually himself the variables with database columns. Hibernate does this implicitly. It lessens the programmer code and this cuts short the development time and also maintenance cost.

**7. Connection pooling**

Hibernate supports connection pooling to increase the performance of database access (hits). Hibernate comes with three styles of pooling – **apache dbcp**, **proxoo**l and **C3P0**.

**8. Optimize Performance (Caching)**

If the client requires the same records again and again, Hibernate uses **cache memory**; places the records in **cache** and gives to the client. **Cache is nothing but retention (storing or accumulating) of data**. Hibernate comes with **Hibernate Dual Layer Caching Architecture (HDLCA)**. Caching is transparent to client and increases the performance. In JDBC, the programmer should hand code everything.

**9. Supports Transactions**

Hibernate comes with transaction facility. To look after, there is a **Transaction** class. Using Transaction class, a programmer can **begin** or **commit** or **rollback** a transaction.

**10. Criteria and Criterion/Conditions and Condition**

Hibernate comes with Criteria class to use with **HQL**. Criteria impose restrictions (constraints) on a database query.

**11. Open Source Code**

Hibernate is absolutely free either for trial use or commercial use. It is available under **LGPL (Lesser General Public License)**. Hibernate can be freely distributed and customized for both development and production deployments.

**12. Automatic Versioning and Time Stamping**

**Versioning** is a feature of a database where the user is guaranteed of changes done by him are not rolled back by another intentionally or unintentionally. **Automatic versioning is a technique of assuring data integrity**. Hibernate allows the programmer to specify the version field. On this field, the Hibernate updates the table each time relational tuple is modified through a Java class object. A row in a table is called **tuple** of the relation. When one user does an update operation, the other is not allowed. JDBC does not come with implicit check of updation of data.

**13. Enterprise Reliability and Scalability**

Hibernate can scale database hits from few hundreds of in-house Intranet to enterprise mission critical operations with lakhs of hits. In JDBC it is very difficult.

Disadvantages

 Hibernate increases complexity to enter data in a database with all mapping XML files etc. When database does not change frequently, I feel, JDBC is the best choice.

 In Web (Internet) environment, Hibernate API support is not adequate (acceptable/enough) .

 Hibernate does not allow to store or save two objects in a database with a single query.

# Hibernate Example with Eclipse IDE

1) Create the java project

2) Add jar files for hibernate and also add jdbc jar file to the project for jdbc Driver class “ojdbc14-1.0.jar”

3) Create the Persistent class

**package** com.visix.mnrao.emp;

**public** **class** Employee

{

**private** **int** empNum;

**private** String empName;

**private** **double** empSalary;

**private** **int** deptNum;

**public** **int** getEmpNum()

{

**return** empNum;

}

**public** **void** setEmpNum(**int** empNum)

{

**this**.empNum = empNum;

}

**public** String getEmpName()

{

**return** empName;

}

**public** **void** setEmpName(String empName)

{

**this**.empName = empName;

}

**public** **double** getEmpSalary()

{

**return** empSalary;

}

**public** **void** setEmpSalary(**double** empSalary)

{

**this**.empSalary = empSalary;

}

**public** **int** getDeptNum()

{

**return** deptNum;

}

**public** **void** setDeptNum(**int** deptNum)

{

**this**.deptNum = deptNum;

}

}

4) Create the Hibernate mapping file for Persistent class

To create the hibernate mapping file, Right click on src - new - file - specify the file name (e.g. employee.hbm.xml) - ok. It must be outside the package.

**Employee.hbm.xml**

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

<!DOCTYPE hibernate-mapping PUBLIC

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

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

<hibernate-mapping>

<class name=*"com.visix.mnrao.emp.Employee"* table=*"EMPLOYEE"*>

<id name=*"empNum"*>

<generator class=*"assigned"*></generator>

</id>

<property name=*"empName"*></property>

<property name=*"empSalary"*></property>

<property name=*"deptNum"*></property>

</class>

</hibernate-mapping>

5) Create the Hibernate Configuration file

To create the configuration file, right click on src - new - file. Now specify the configuration file name e.g. hibernate.cfg.xml.

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

<property name=*"dialect"*>org.hibernate.dialect.Oracle9Dialect</property>

<property name=*"connection.url"*>jdbc:oracle:thin:@localhost:1521:orcl</property>

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

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

<property name=*"connection.driver\_class"*>oracle.jdbc.driver.OracleDriver</property>

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

</session-factory>

</hibernate-configuration>

### 6) Create the class that retrieves or stores the persistent object

Storing the employee object to the database.

**package** com.visix.mnrao.emp;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

**public** **class** StoreData

{

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

{

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

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

SessionFactory factory=cfg.buildSessionFactory();

Session session=factory.openSession();

Transaction t=session.beginTransaction();

Employee e1=**new** Employee();

e1.setEmpNum(1002);

e1.setEmpName("xyz");

e1.setEmpSalary(6000);

e1.setDeptNum(20);

session.persist(e1);//persisting the object

t.commit();//transaction is commited

session.close();

System.***out***.println("successfully saved");

}

}

Right click on main method  Run AS  Java Application

O/p

**Successfully saved**

**Sql>** SELECT table\_name FROM user\_tables;

Now you can see **EMPLOYEE** table in the scott schema;

**Configuration Object:**

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate. The Configuration object provides two keys components:

* **Database Connection:**

This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.

* **Class Mapping Setup**

This component creates the connection between the Java classes and database tables..

**SessionFactory Object:**

Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So if you are using multiple databases then you would have to create multiple SessionFactory objects.

**Session Object:**

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

**Transaction Object:**

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

**Hibernate Configuration File**

Hibernate Configuration File(cfg file) is the file loaded into an hibernate application when working with hibernate. Hibernate uses this file to establish connection to the database server. It is an XML file which is used to define below information. Standard name for this file is hibernate.cfg.xml

There must be one configuration file for each database used in the application, suppose if we want to connect with 2 databases, like Oracle, MySql, then we must create 2 configuration files with different names, like oracle.cfg.xml for Oracle DB and mysql.cfg.xml for mysql database.

**Sample Configuration File :**

<hibernate-configuration>

<session-factory>

<! -- Related to the connection START -->

<property name=*"dialect"*>Database dialect class</property>

<property name=*"connection.driver\_class"*>Driver Class Name </property>

<property name=*"connection.url"*>URL </property>

<property name=*"connection.user"*>USER NAME </property>

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

<! -- Related to the connection END -->

<! -- Related to hibernate properties START -->

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

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

<property name=*"use\_sql\_comments"*>true/false</property>

<property name=*"hbm2ddl.auto"*>create/create-drop/update/validate</property>

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

<! -- Related to hibernate properties END-->

<! -- Related to mapping START-->

<mapping resource=*"hbm file 1 name .xml"* / >

<mapping resource=*"hbm file 2 name .xml"* / >

<! -- Related to the mapping END -->

</session-factory>

</hibernate-configuration>

**Hibernate Properties:**

**hibernate.dialect**

<property name=*"dialect"*>Database dialect class</property>

This property makes Hibernate generate the appropriate SQL for the chosen database.

**hibernate.connection.driver\_class**

<property name=*"connection.driver\_class"*>Driver Class Name </property>

The JDBC driver class.

**hibernate.connection.url**

<property name=*"connection.url"*>URL </property>

The JDBC URL to the database instance.

**hibernate.connection.username**

<property name=*"connection.user"*>USER NAME </property>

The database username.

**hibernate.connection.password**

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

The database password.

**hibernate.connection.pool\_size**

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

Limits the number of connections waiting in the Hibernate database connection pool.

**hibernate.show\_sql**

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

If the value is true, We can see generated sql statements in console

**hibernate.format\_sql :**

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

If the value is true, We can see generated sql statements in a readable format

**hibernate .use\_sql\_comments :**

<property name=*"use\_sql\_comments"*>true/false</property>

If the value is true, We can see comments in generated sql statements.

**hbm2ddl.auto**

<property name=*"hbm2ddl.auto"*>create/create-drop/update/validate</property>

 **create:**  Creates schema, destroys previous data.

 **create-drop:** Drops the schema at the end of a session.

 **update:**  Updates the schema.

 **validate:** Validates the schema. It makes no changes to database

**hibernate.connection.autocommit**

Allows autocommit mode to be used for the JDBC connection.

|  |  |
| --- | --- |
| **Database** | **Dialect Property** |
| DB2 | org.hibernate.dialect.DB2Dialect |
| HSQLDB | org.hibernate.dialect.HSQLDialect |
| HypersonicSQL | org.hibernate.dialect.HSQLDialect |
| Informix | org.hibernate.dialect.InformixDialect |
| Ingres | org.hibernate.dialect.IngresDialect |
| Interbase | org.hibernate.dialect.InterbaseDialect |
| Microsoft SQL Server 2000 | org.hibernate.dialect.SQLServerDialect |
| Microsoft SQL Server 2005 | org.hibernate.dialect.SQLServer2005Dialect |
| Microsoft SQL Server 2008 | org.hibernate.dialect.SQLServer2008Dialect |
| MySQL | org.hibernate.dialect.MySQLDialect |
| Oracle (any version) | org.hibernate.dialect.OracleDialect |
| Oracle 11g | org.hibernate.dialect.Oracle10gDialect |
| Oracle 10g | org.hibernate.dialect.Oracle10gDialect |
| Oracle 9i | org.hibernate.dialect.Oracle9iDialect |
| PostgreSQL | org.hibernate.dialect.PostgreSQLDialect |
| Progress | org.hibernate.dialect.ProgressDialect |
| SAP DB | org.hibernate.dialect.SAPDBDialect |
| Sybase | org.hibernate.dialect.SybaseDialect |
| Sybase Anywhere | org.hibernate.dialect.SybaseAnywhereDialect |

Object States in Hibernate –

1. Transient,
2. Persistent and
3. Detached

Object states in Hibernate plays a vital role in the execution of code in an application. Hibernate has provided three different states for an object of a **pojo (Plain Old Java Object** ) class. These three states are also called as life cycle states of an object.

**1. Transient Object State:**

An object which is not associated with hibernate session and does not represent a row in the database is considered as transient. It will be garbage collected if no other object refers to it.

An object that is created for the first time using the new() operator is in transient state. When the object is in transient sate then it will not contain any identifier (primary key value). You have to use save, persist or saveOrUpdate methods to persist the transient object.

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

emp.setName("Ravi Raj");

// here emp object is in a transient state

**2. Persistent Object State:**

An object that is associated with the hibernate session is called as Persistent object. When the object is in persistent state, then it represent one row of the database and consists of an identifier value. You can make a transient instance persistent by associating it with a Session.

Long id = (Long) session.save(emp);

//emp object is now in a persistent state

**3. Detached Object State:**

Object which is just removed from hibernate session is called as detached object. The state of the detached object is called as detached state.

When the object is in detached sate then it contain identity but you can’t do persistence operation with that identity.

Any changes made to the detached objects are not saved to the database. The detached object can be reattached to the new session and save to the database using update, saveOrUpdate and merge methods.

Example

**package** com.visix.mnrao.bean;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.jwt.hibernate.Student;

**public** **class** ObjectStatesDemo {

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

// Transient object state

Student student = **new** Student();

student.setId(101);

student.setName("MNRAO");

student.setRoll("10");

student.setDegree("B.Tech");

student.setPhone("9999");

// Transient object state

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

// Persistent object state

session.save(student);

t.commit();

// Persistent object state

session.close();

// Detached object state

}

}

**Hibernate Example With Annotation :**

This example is the same as the first example except that it uses annotations. In our first example we created *.hbm.xml* file for database and pojo class mapping, here there is no need to create hbm files, instead we will use annotations to do the object relational mapping.

**Steps to create the hibernate application with Annotation**

1. Add the jar file
2. Create the Persistent class
3. Add mapping of Persistent class in configuration file
4. Create the class that retrieves or stores the persistent object

Step 1: Add the jar file for annotation

In addition to the already existing jar files you need to add the following jar files to the classpath

1.hibernate-commons-annotations.jar

2.ejb3-persistence.jar

3.hibernate-annotations.jar –

Following jar files are enough for getting annotation support in hibernate.

1.antlr-2.7.6.jar

2.commons-collections-3.1.jar

3.dom4j-1.6.1.jar

4.hibernate-commons-annotations-3.2.0.Final.jar

5.hibernate-core-3.6.7.Final.jar

6.hibernate-jpa-2.0-api-1.0.1.Final.jar

7.javassist.jar

8.jms-1.1.jar 9.jsr250-api-1.0.jar

10.jta-1.1.jar

11.log4j-1.2.16.jar

12.mysql connector.jar

13.slf4j-api-1.6.1.jar

14.slf4j-log4j12-1.6.1.jar

**Step 2 : Create the Persistent class**

**package** com.visix.mnrao.bean;

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "STUDENT")

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

**private** **long** stusid;

**private** String name;

**private** String degree;

**private** String roll;

**private** String phone;

@Id

**public** **long** getStusid() {

**return** stusid;

}

**public** **void** setStusid(**long** stusid) {

**this**.stusid = stusid;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDegree() {

**return** degree;

}

**public** **void** setDegree(String degree) {

**this**.degree = degree;

}

**public** String getRoll() {

**return** roll;

}

**public** **void** setRoll(String roll) {

**this**.roll = roll;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

}

In the above class, we are creating the same persistent class which we have created in the previous example. But here, we are using annotation.  
  
**@Entity** annotation marks this class as an entity.  
  
**@Table** annotation specifies the table name where data of this entity is to be persisted. If you are not using @Table annotation in Entity class, hibernate will use the class name as the table name by default.  
  
**@Id** annotation marks the identifier for this entity.  
  
**@Column** annotation specifies the details of the column for this property or field. If @Column annotation is not specified, property name will be used as the column name bydefault.

**Step 3 : Mapping of Persistent class in configuration file**

The next change you need to do here is, instead of adding the .hbm.xml file to the hibernate.cfg.xml file, we add the fully qualified name of the annotated class to the mapping element.

**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=*"hibernate.dialect"*>org.hibernate.dialect.Oracle9Dialect</property>

<property name=*"hibernate.connection.driver\_class"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"hibernate.connection.url"*>jdbc:oracle:thin:@localhost:1521:ORCL</property>

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

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

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

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

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

<mapping class=*"com.visix.mnrao.bean.Student"* />

</session-factory>

</hibernate-configuration>

**Step 4 : Create Test class that retrieves or stores the persistent object**

**package** com.visix.mnrao.test;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.visix.mnrao.bean.Student;

**public** **class** Test {

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

Session session = **new** AnnotationConfiguration().configure(“**hibernate.cfg.xml**”)

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student1 = **new** Student();

student1.setStusid(1);

student1.setName("MNRAO");

student1.setRoll("101");

student1.setDegree("B.Tech");

student1.setPhone("99999");

Student student2 = **new** Student();

student2.setStusid(2);

student2.setName("Ravi");

student2.setRoll("102");

student2.setDegree("B.Tech");

student2.setPhone("934499");

session.persist(student1);

session.persist(student2);

t.commit();

session.close();

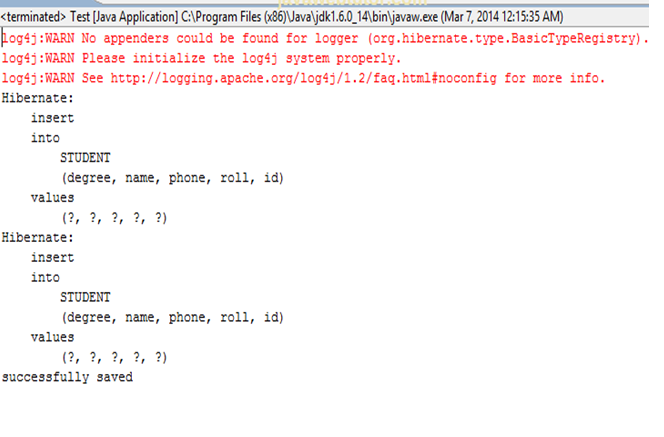
System.***out***.println("successfully saved");

}

}

**Output :**

If data is inserted into DB you can see generated sql in console as given below



**Difference between hibernate save(), saveOrUpdate() and persist() methods :**

Session interface in Hibernate provides a couple of methods to move an object from new or transient state to persistent state e.g. save(), saveOrUpdate() and persist() are used to store an object into the database, but there are some significant differences between them.

save()  generates a new identifier and INSERT record into database

save() method will fail if the primary key is already persistent i.e. object already exists in the database. This is why, you should only call save() with an absolutely new object which doesn't have any database identifier. Calling save() method on the detached object will throw exception.

saveOrUpdate  can either INSERT or UPDATE based upon existence of record.

The Session.save() method does an INSERT to store the object into the database and return the identifier generated by the database.

On the other hand, saveOrUpdate() can do INSERT or UPDATE depending upon whether object exists in database or not.

saveOrUpdate is more flexible in terms of use but it involves an extra processing to find out whether record already exists in table or not. saveOrUpdate does a select first to determine if it needs to do an insert or an update. It will insert data if primary key not exist otherwise it will update data.

Another key difference between save() and saveOrUpdate() method is that,

save() method is used to bring a transient object to persistent state

**(but)**

saveOurUpdate() can bring both transient (new) and detached (existing) object into persistent state. It is often used to re-attach a detached object into Session.

**Conclusion:**

save() method saves records into database by INSERT SQL query, Generates a new identifier and return the Serializable identifier back.

On the other hand saveOrUpdate() method either INSERT or UPDATE based upon existence of object in database. If persistence object already exists in database then UPDATE SQL will execute and if there is no corresponding object in database than INSERT will run.

**Example:**

Below is example of both save and saveOrUpdate() method. You can easily identify the difference in generated SQL statement in the console.

**Entity Class**

**import** javax.persistence.Entity;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "STUDENT")

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

**private** **long** id;

**private** String name;

**private** String degree;

**private** String roll;

**private** String phone;

@Id

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDegree() {

**return** degree;

}

**public** **void** setDegree(String degree) {

**this**.degree = degree;

}

**public** String getRoll() {

**return** roll;

}

**public** **void** setRoll(String roll) {

**this**.roll = roll;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

}

**Configuration File:**

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

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

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

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

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

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

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

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

<mapping class=*"com.visix.mnrai.bean.Student"* />

</session-factory>

</hibernate-configuration>

**Using save() method**

**package** com.visix.mnrao.student;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**import** com.visix.mnrao.bean.Student;

**public** **class** SaveTest {

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

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student = **new** Student();

student.setId(1);

student.setName("mnrao");

student.setRoll("101");

student.setDegree("B.E");

student.setPhone("99999");

Long studentId = (Long) session.save(student);

t.commit();

session.close();

System.***out***.println(" Identifier : " + studentId);

System.***out***.println("successfully saved");

}

}

Output:

log4j:WARN No appenders could be found for logger (org.hibernate.type.BasicTypeRegistry). log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Hibernate:

insert

into

STUDENT

(degree, name, phone, roll, id)

values (?, ?, ?, ?, ?)

Identifier : 1

successfully saved.

**Look at the above output carefully, save() method first generates identifier and then inserts data into database. We printed the value of the identifier after saving the object. You can clearly see the insert query in the console output**

**Using saveOrUpdate() method:**

**package** com.visix.mnrao.student;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** SaveOrUpdateTest {

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

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student = **new** Student();

student.setId(101);

student.setName("MNRAO");

student.setRoll("10");

student.setDegree("B.Tech");

student.setPhone("8888");

session.saveOrUpdate(student);

t.commit();

session.close();

System.***out***.println("successfully saved");

}

}

Output:

log4j:WARN No appenders could be found for logger (org.hibernate.type.BasicTypeRegistry). log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Hibernate:

select

student\_.id,

student\_.degree as degree0\_,

student\_.name as name0\_,

student\_.phone as phone0\_,

student\_.roll as roll0\_

from STUDENT student\_

where student\_.id=?

Hibernate:

insert

into

STUDENT

(degree, name, phone, roll, id)

values (?, ?, ?, ?, ?)

successfully saved

The above output shows that the entity with specified identifier does not exist in database by running select query. So it applied insert query to persist data.

Now check the output for such situation where entity for the given identifier is already exist in database. In the above example just change the name and phone number. Modified code is given below.

**package** com.visix.mnrao.student;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** SaveOrUpdateTest {

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

Session session = **new** AnnotationConfiguration().configure()

.buildSessionFactory().openSession();

Transaction t = session.beginTransaction();

Student student = **new** Student();

student.setId(101);

student.setName("MNRAO");

student.setRoll("10");

student.setDegree("B.Tech");

student.setPhone("9999");

session.saveOrUpdate(student);

t.commit();

session.close();

System.***out***.println("successfully saved");

}

}

**Output:**

log4j:WARN No appenders could be found for logger (org.hibernate.type.BasicTypeRegistry). log4j:WARN Please initialize the log4j system properly.

log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.

Hibernate:

select

student\_.id,

student\_.degree as degree0\_,

student\_.name as name0\_,

student\_.phone as phone0\_,

student\_.roll as roll0\_

from

STUDENT student\_

where

student\_.id=?

Hibernate:

update STUDENT

set

degree=?,

name=?,

phone=?,

roll=?

where id=?

successfully saved

In the above output, first select query runs to check existence of data for given identifier and since data is there, so update query runs to update data.

**Difference between save() and persist() methods**

* Main difference of save() and persist() is return type of the save() method is java.io.Serializable it returns the generated identity value whereas the return type of persist method is void i.e, it will not return any value. persist() is similar to Session.save() i.e. it is used to move a transient object to the persistent state by storing it into the database but it doesn't return the database identifier.

* Another key difference is persist() method can be used only within the boundary of a transaction whereas save method can be used inside or outside the transaction boundaries. persist() method guarantees that it will not execute an INSERT statement if it is called outside of transaction boundaries. But in the case of save() INSERT happens immediately, no matter if you are inside or outside of a transaction. This is not good in a long-running conversation with an extended Session/persistence context.

**Integrating JSP, Servlet and Hibernate in an MVC application**

**Project Description**

we are going to create a real time MVC application by using Servlet, JSP and Hibernate. This example is developed using MVC pattern, where JSP pages acts as view layer, Hibernate and Oracle database as the model and servlet as controller layer. It’s a very simple and basic application.

we are going to create a web application by using servlet, jsp and hibernate. When user will access the application ,login page will be displayed. For first time user there is a link for the registration. Once user will register his/her details will be collected in servlet and finally data will be saved into database by using hibernate.

Once user got registered , user can login to the application and use the website. If user enters wrong credential error page will be displayed.

**Look and Feel :**

We have used css and images to make pages attractive. After completion of this you can understand how css and images should be used in java based web application.

For creating the web application, we used JSP for presentation logic, Servlet class for controller layer and hibernate for database access codes.

**Tools and Technologies :**

* JDK 1.7
* Hibernate 5.2.7.Final
* Eclipse
* Oracle 11g.
* Tomcat 7.0

Create database table for storing data :

SQL query is given below. Execute the following query in ORACLE database.

**CREATE TABLE USER\_DETAILS (**

**unid DECIMAL(10,2) NOT NULL ,**

**firstName VARCHAR2(45) NOT NULL ,**

**middleName VARCHAR2(45) NOT NULL ,**

**lastName VARCHAR2(45) NOT NULL ,**

**email VARCHAR2(45) NOT NULL ,**

**userid VARCHAR2(45) NOT NULL ,**

**passwd VARCHAR2(45)NOT NULL**

**);**

ALTER TABLE USER\_DETAILS ADD (

CONSTRAINT unid\_pk PRIMARY KEY (UNID));

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

CREATE SEQUENCE hibernate\_sequence START WITH 1; ======================================

For creating the application in Eclipse, follow the steps mentioned below.

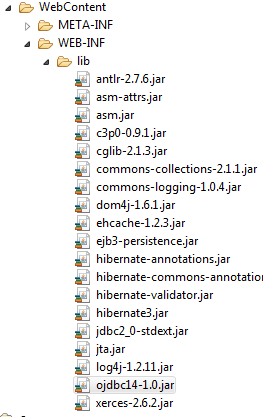
## Step 1 : Create Dynamic Web Project :

Open eclipse IDE, and go to**File -> New -> Project ->** and select **Dynamic Web Project**,specify the project name as "MVCApplication" and click on next -> finish .

Step 2 : Add Jar files for hibernate and ORACLE :

And also add

servlet-api.jar to the class path



**Directory structure of the project :**



## Step 3 : Creating web pages :

Now let us create jsp files inside Web-Content folder of your project.

**login.jsp:**

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns=*"http://www.w3.org/1999/xhtml"*>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=iso-8859-1"* />

<title>Login Page</title>

<link href=*"css/style.css"* rel=*"stylesheet"* type=*"text/css"* />

</head>

<body>

<form action=*"LoginServlet"* method=*"post"*>

<div style="padding: *100px 0 0 250px*;">

<div id=*"login-box"*>

<h2>Login Page</h2>

Please provide your credential to use this website

<br>

<br>

<div id=*"login-box-name"* style="margin-top:*20px*;">User Id:</div>

<div id=*"login-box-field"* style="margin-top:*20px*;">

<input name=*"userId"* class=*"form-login"* title=*"Username"* value=*""* size=*"30"* maxlength=*"50"* />

</div>

<div id=*"login-box-name"*>Password:</div>

<div id=*"login-box-field"*>

<input name=*"password"* type=*"password"* class=*"form-login"* title=*"Password"* value=*""* size=*"30"* maxlength=*"48"* />

</div>

<br />

<span class=*"login-box-options"*>

New User? <a href=*"register.jsp"* style="margin-left:*30px*;">Register Here</a>

</span>

<br />

<br />

<input style="margin-left:*100px*;" type=*"submit"* value=*"Login"* />

</div>

</div>

</form>

</body>

</html>

**register.jsp**

<html>

<head>

<title>Registration Form</title>

<style type=*"text/css"*>

**h3**{font-family: *Calibri*; font-size: *22pt*; font-style: *normal*; font-weight: *bold*; color:*SlateBlue*;

text-align: *center*; text-decoration: *underline* }

**table**{font-family: *Calibri*; color:*white*; font-size: *11pt*; font-style: *normal*;width: *50%*;

text-align:; background-color: *SlateBlue*; border-collapse: *collapse*; border: *2px solid navy*}

**table***.inner*{border: *0px*}

</style>

</head>

<body>

<h3>Student Registration Form</h3>

<form action=*"RegisterServlet"* method=*"POST"*>

<table align=*"center"* cellpadding = *"10"*>

<tr>

<td>First Name</td>

<td><input type=*"text"* name=*"firstName"* maxlength=*"30"*/>

(max 30 characters a-z and A-Z)

</td>

</tr>

<tr>

<td>Middle Name</td>

<td><input type=*"text"* name=*"middleName"* maxlength=*"30"*/>

(max 30 characters a-z and A-Z)

</td>

</tr>

<tr>

<td>Last Name</td>

<td><input type=*"text"* name=*"lastName"* maxlength=*"30"*/>

(max 30 characters a-z and A-Z)

</td>

</tr>

<tr>

<td>Email</td>

<td><input type=*"text"* name=*"email"* maxlength=*"100"* /></td>

</tr>

<tr>

<td>User ID</td>

<td><input type=*"text"* name=*"userId"* maxlength=*"100"* /></td>

</tr>

<tr>

<td>Password</td>

<td><input type=*"text"* name=*"password"* maxlength=*"100"* /></td>

</tr>

<tr>

<td colspan=*"2"* align=*"center"*>

<input type=*"submit"* value=*"Submit"*>

<input type=*"reset"* value=*"Reset"*>

</td>

</tr>

</table>

</form>

</body>

</html>

home.jsp:

<%@page import=*"java.util.List"*%>

<%@page import=*"com.visix.mnrao.service.LoginService"*%>

<%@page import=*"java.util.Date"*%>

<%@page import=*"com.visix.mnrao.model.User"*%>

<%@page contentType=*"text/html"* pageEncoding=*"UTF-8"*%>

<!DOCTYPE html>

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<link rel=*"stylesheet"* type=*"text/css"* href=*"css/style.css"*/>

<title>Result Page</title>

</head>

<body>

<center>

<div id=*"container"*>

<h1>Result Page</h1>

<b>This is Sample Result Page</b><br/>

<%=**new** Date()%></br>

<%

User user = (User) session.getAttribute("user");

%>

<b>Welcome <%= user.getFirstName() + " " + user.getLastName()%></b>

<br/>

<a href=*"logout.jsp"*>Logout</a>

</p>

<table>

<thead>

<tr>

<th>User ID</th>

<th>First Name</th>

<th>Middle Name</th>

<th>Last Name</th>

<th>email</th>

</tr>

</thead>

<tbody>

<%

LoginService loginService = **new** LoginService();

List<User> list = loginService.getListOfUsers();

**for** (User u : list) {

%>

<tr>

<td><%=u.getUserId()%></td>

<td><%=u.getFirstName()%></td>

<td><%=u.getMiddleName()%></td>

<td><%=u.getLastName()%></td>

<td><%=u.getEmail()%></td>

</tr>

<%}%>

<tbody>

</table>

<br/>

</div>

</center>

</body>

</html>

**logout.jsp:**

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<link rel=*"stylesheet"* type=*"text/css"* href=*"css/style.css"*>

<title>logout Page</title>

</head>

<body>

<%

session.removeAttribute("userId");

session.removeAttribute("password");

session.invalidate();

%>

<center>

<h1>You have successfully logged out</h1>

To login again <a href=*"login.jsp"*>click here</a>.

</center>

</body>

</html>

**error.jsp:**

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Error Page</title>

</head>

<body>

<center>

<h1>Your Login Was Unsuccessful - Please Try Again</h1>

To login again <a href=*"login.jsp"*>click here</a>.

</center>

</body>

</html>

Step 4 : Create css file

Create a folder css inside WebContent of the project. Now create a css file style.css inside this folder and add below code into this file.

**style.css:**

**body**{

font-family:*"Lucida Grande",* *"Lucida Sans Unicode",* *Verdana,* *Arial,* *Helvetica,* *sans-serif*;

font-size:*12px*;

}

**p,** **h1,** **form,** **button**{border:*0*; margin:*0*; padding:*0*;}

*.spacer*{clear:*both*; height:*1px*;}

*.mainform*{

margin:*0* *auto*;

width:*400px*;

padding:*14px*;

}

*#container*{

border:*solid* *2px* *burlywood*;

background: *silver*;

border-radius: *5px*;

}

*#container* **h1** {

font-size:*14px*;

font-weight:*bold*;

margin-bottom:*8px*;

}

*#container* **p**{

font-size:*11px*;

color:*#666666*;

margin-bottom:*20px*;

border-bottom:*solid* *1px* *burlywood*;

padding-bottom:*10px*;

}

*#container* **label**{

display:*block*;

font-weight:*bold*;

text-align:*right*;

width:*140px*;

float:*left*;

}

*#container* *.small*{

color:*#666666*;

display:*block*;

font-size:*11px*;

font-weight:*normal*;

text-align:*right*;

width:*140px*;

}

*#container* **input**{

float:*left*;

font-size:*12px*;

padding:*4px* *2px*;

border:*solid* *1px* *burlywood*;

width:*200px*;

margin:*2px* *0* *20px* *10px*;

border-radius: *5px*;

}

*#container* **button**{

clear:*both*;

margin-left:*150px*;

width:*125px*;

height:*31px*;

background:*#666666*;

text-align:*center*;

line-height:*31px*;

color:*#FFFFFF*;

font-size:*11px*;

font-weight:*bold*;

border-radius: *5px*;

}

**table,** **tr,** **td**{

border-radius: *5px*;

border:*solid* *1px*;

padding: *5px*;

width: *max-content*;

}

*#login-box* {

width:*333px*;

height: *352px*;

padding: *58px* *76px* *0* *76px*;

color: *#ebebeb*;

font: *12px* *Arial,* *Helvetica,* *sans-serif*;

margin-top: *-70px*;

background: *url(../images/login-box-backg.png)* *no-repeat* *left* *top*;

}

*#login-box* **img** {

border:*none*;

}

*#login-box* **h2** {

padding:*0*;

margin:*0*;

color: *#ebebeb*;

font: *bold* *44px* *"Calibri",* *Arial*;

}

*#login-box-name* {

float: *left*;

display:*inline*;

width:*80px*;

text-align: *right*;

padding: *14px* *10px* *0* *0*;

margin:*0* *0* *7px* *0*;

}

*#login-box-field* {

float: *left*;

display:*inline*;

width:*230px*;

margin:*0*;

margin:*0* *0* *7px* *0*;

}

*.form-login* {

width: *205px*;

padding: *10px* *4px* *6px* *3px*;

border: *1px* *solid* *#0d2c52*;

background-color:*#1e4f8a*;

font-size: *16px*;

color: *#ebebeb*;

}

*.login-box-options* {

clear:*both*;

padding-left:*87px*;

font-size: *11px*;

}

*.login-box-options* **a** {

color: *#ebebeb*;

font-size: *11px*;

}

**Step 5 : Creating hibernate persistence Class :**

Create a package **com.visix.mnrao.model** and create a java class User in this package and add following code in this class.

**User.java**

**package** com.visix.mnrao.model;

**import** java.io.Serializable;

**import** javax.persistence.Entity;

**import** javax.persistence.GeneratedValue;

**import** javax.persistence.Id;

**import** javax.persistence.Table;

@Entity

@Table(name = "USER\_DETAILS")

**public** **class** User **implements** Serializable {

@Id

@GeneratedValue

**private** Long unid;

**private** String firstName;

**private** String middleName;

**private** String lastName;

**private** String email;

**private** String userId;

**private** String passwd;

**public** User() {

}

**public** User(String firstName, String middleName, String lastName,

String email, String userId, String passwd) {

**this**.firstName = firstName;

**this**.middleName = middleName;

**this**.lastName = lastName;

**this**.email = email;

**this**.userId = userId;

**this**.passwd = passwd;

}

**public** Long getUnid() {

**return** unid;

}

**public** **void** setUnid(Long unid) {

**this**.unid = unid;

}

**public** String getFirstName() {

**return** firstName;

}

**public** **void** setFirstName(String firstName) {

**this**.firstName = firstName;

}

**public** String getMiddleName() {

**return** middleName;

}

**public** **void** setMiddleName(String middleName) {

**this**.middleName = middleName;

}

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getUserId() {

**return** userId;

}

**public** **void** setUserId(String userId) {

**this**.userId = userId;

}

**public** String getPasswd() {

**return** passwd;

}

**public** **void** setPasswd(String passwd) {

**this**.passwd = passwd;

}

}

**Step 5 : Create Util class for getting SeesionFactory Object**

Create a java class HibernateUtil. Method inside this class will return SessionFactory Object. SessionFactory is heavy weight object. For better performance only one SessionFactory should be created for entire application. So this class should be created as singleton.

HibernateUtil.java:

**package** com.visix.mnrao.util;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.cfg.AnnotationConfiguration;

**public** **class** HibernateUtil {

**private** **static** **final** SessionFactory ***sessionFactory***;

**static** {

**try** {

***sessionFactory*** = **new** AnnotationConfiguration().configure("hibernate.cfg.xml").buildSessionFactory();

} **catch** (Throwable ex) {

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

**throw** **new** ExceptionInInitializerError(ex);

}

}

**public** **static** Session openSession() {

**return** ***sessionFactory***.openSession();

}

}

**Step 6 : Create the Configuration file :**

The configuration file contains informations about the database and mapping file. Conventionally, its name should be hibernate.cfg.xml .Configuration file must be in classpath of your Project.Place this file in src of your project by default it will added to classpath of your project.

**hibernate.cfg.xml**

<?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.Oracle9Dialect</property>

<property name=*"hibernate.connection.driver\_class"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"hibernate.connection.url"*>jdbc:oracle:thin:@localhost:1521:ORCL</property>

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

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

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

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

<mapping class=*"com.visix.mnrao.model.User"* />

</session-factory>

</hibernate-configuration>

**Step 7 : Create Controller class**

Here we are going to create two classes LoginServlet and RegisterServlet which will help interaction between view layer, and business service classes.LoginSrvlet will control login request and RegisterServlet will control /new user registration process.

**LoginServlet.java:**

**package** com.visix.mnrao.ctrl;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** com.visix.mnrao.model.User;

**import** com.visix.mnrao.service.LoginService;

**public** **class** LoginServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

String userId = request.getParameter("userId");

String password = request.getParameter("password");

LoginService loginService = **new** LoginService();

**boolean** result = loginService.authenticateUser(userId, password);

User user = loginService.getUserByUserId(userId);

**if**(result == **true**){

request.getSession().setAttribute("user", user);

response.sendRedirect("home.jsp");

}

**else**{

response.sendRedirect("error.jsp");

}

}

}

**RegisterServlet.java**

**package** com.visix.mnrao.ctrl;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** com.visix.mnrao.model.User;

**import** com.visix.mnrao.service.RegisterService;

**public** **class** RegisterServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String firstName = request.getParameter("firstName");

String middleName = request.getParameter("middleName");

String lastName = request.getParameter("lastName");

String email = request.getParameter("email");

String userId = request.getParameter("userId");

String password = request.getParameter("password");

User user = **new** User(firstName,middleName,lastName, email,userId, password);

**try** {

RegisterService registerService = **new** RegisterService();

**boolean** result = registerService.register(user);

out.println("<html>");

out.println("<head>");

out.println("<title>Registration Successful</title>");

out.println("</head>");

out.println("<body>");

out.println("<center>");

**if**(result){

out.println("<h1>Thanks for Registering with us :</h1>");

out.println("To login with new UserId and Password<a href=login.jsp>Click here</a>");

}**else**{

out.println("<h1>Registration Failed</h1>");

out.println("To try again<a href=register.jsp>Click here</a>");

}

out.println("</center>");

out.println("</body>");

out.println("</html>");

} **finally** {

out.close();

}

}

}

**Step 8 : Create Service class for Business logic**

Create two classes LoginService and RegisterService. These classes will interact with model layer for business processing.

**LoginService.java:**

**package** com.visix.mnrao.service;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.hibernate.Query;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** com.visix.mnrao.util.HibernateUtil;

**import** com.visix.mnrao.model.User;

**public** **class** LoginService {

**public** **boolean** authenticateUser(String userId, String password) {

User user = getUserByUserId(userId);

**if**(user!=**null** && user.getUserId().equals(userId) && user.getPasswd().equals(password)){

**return** **true**;

}**else**{

**return** **false**;

}

}

**public** User getUserByUserId(String userId) {

Session session = HibernateUtil.*openSession*();

Transaction tx = **null**;

User user = **null**;

**try** {

tx = session.getTransaction();

tx.begin();

Query query = session.createQuery("from User where userId='"+userId+"'");

user = (User)query.uniqueResult();

tx.commit();

} **catch** (Exception e) {

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

tx.rollback();

}

e.printStackTrace();

} **finally** {

session.close();

}

**return** user;

}

**public** List<User> getListOfUsers(){

List<User> list = **new** ArrayList<User>();

Session session = HibernateUtil.*openSession*();

Transaction tx = **null**;

**try** {

tx = session.getTransaction();

tx.begin();

list = session.createQuery("from User").list();

tx.commit();

} **catch** (Exception e) {

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

tx.rollback();

}

e.printStackTrace();

} **finally** {

session.close();

}

**return** list;

}

}

RegisterService.java:

**package** com.visix.mnrao.service;

**import** org.hibernate.Query;

**import** org.hibernate.Session;

**import** org.hibernate.Transaction;

**import** com.visix.mnrao.util.HibernateUtil;

**import** com.visix.mnrao.model.User;

**public** **class** RegisterService {

**public** **boolean** register(User user){

Session session = HibernateUtil.*openSession*();

**if**(isUserExists(user)) **return** **false**;

Transaction tx = **null**;

**try** {

tx = session.getTransaction();

tx.begin();

session.saveOrUpdate(user);

tx.commit();

} **catch** (Exception e) {

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

tx.rollback();

}

e.printStackTrace();

} **finally** {

session.close();

}

**return** **true**;

}

**public** **boolean** isUserExists(User user){

Session session = HibernateUtil.*openSession*();

**boolean** result = **false**;

Transaction tx = **null**;

**try**{

tx = session.getTransaction();

tx.begin();

Query query = session.createQuery("from User where userId='"+user.getUserId()+"'");

User u = (User)query.uniqueResult();

tx.commit();

**if**(u!=**null**) result = **true**;

}**catch**(Exception ex){

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

tx.rollback();

}

}**finally**{

session.close();

}

**return** result;

}

}

**Step 9: Create web.xml file :**

Create web.xml file inside WEB-INF directory of project.

**web.xml:**

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"* id=*"WebApp\_ID"* version=*"2.5"*>

<display-name>MVCApplication</display-name>

<servlet>

<display-name>LoginServlet</display-name>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>com.visix.mnrao.ctrl.LoginServlet</servlet-class>

</servlet>

<servlet>

<display-name>RegisterServlet</display-name>

<servlet-name>RegisterServlet</servlet-name>

<servlet-class>com.visix.mnrao.ctrl.RegisterServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

<url-pattern>/LoginServlet</url-pattern>

</servlet-mapping>

<servlet-mapping>

<servlet-name>RegisterServlet</servlet-name>

<url-pattern>/RegisterServlet</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>login.jsp</welcome-file>

</welcome-file-list>

</web-app>

Hibernate integration with Servlet and JSP

**Hibernate integration with Servlet and JSP**

Developing Web Application by using hibernate.

We are going to insert the record of the user in the database. We will create Registration form for getting user data. These data we will collect in servlet and finally insert these data into Database by using hibernate.

For creating the web application, we are using

1. JSP for presentation logic,
2. Servlet class for controller layer and
3. DAO class for database access codes.

**Tools and Technologies:**

* JDK 1.7
* Hibernate library
* Eclipse
* Oracle11g.
* Tomcat 7.0

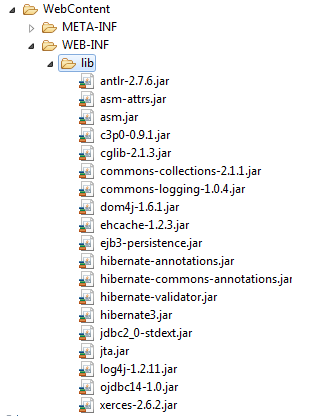
## Step 1 : Create Dynamic Web Project :

Open eclipse IDE,and go to**File -> New -> Project ->** and select **Dynamic Web Project**,specify the project name as HibernateWebApp and click on next -> finish .

# 

Step 2 : Add Jar files for hibernate and Oracle

Copy all the jar files as shown below inside lib folder of the project



**Step 3 : Creating web pages :**

Now let us create register.jsp file inside Web-Content folder of your project. This is a simple form where user can provide his/her detail.

Right Click on Web-Content then New -> JSP File and provide the name of JSP file as register.jsp and click Finish.

**register.jsp**

<%@page contentType=*"text/html"*%>

<%@page pageEncoding=*"UTF-8"*%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=UTF-8"*>

<title>Registration Form</title>

</head>

<body>

<h1>Registration Form</h1>

<form action=*"register"* method=*"post"*>

<table cellpadding=*"3pt"*>

<tr>

<td>User id :</td>

<td><input type=*"text"* name=*"userid"* size=*"30"* /></td>

</tr>

<tr>

<td>User Name :</td>

<td><input type=*"text"* name=*"userName"* size=*"30"* /></td>

</tr>

<tr>

<td>Password :</td>

<td><input type=*"password"* name=*"password1"* size=*"30"* /></td>

</tr>

<tr>

<td>Confirm Password :</td>

<td><input type=*"password"* name=*"password2"* size=*"30"* /></td>

</tr>

<tr>

<td>email :</td>

<td><input type=*"text"* name=*"email"* size=*"30"* /></td>

</tr>

<tr>

<td>Phone :</td>

<td><input type=*"text"* name=*"phone"* size=*"30"* /></td>

</tr>

<tr>

<td>City :</td>

<td><input type=*"text"* name=*"city"* size=*"30"* /></td>

</tr>

</table>

<p />

<input type=*"submit"* value=*"Register"* />

</form>

</body>

</html>

**Step 4 : Creating Java Classes :**

**User.java**

This is a simple bean class representing the Persistent class in hibernate.

Create a package com.stech.test.bean in this package create java class User.java and add following code in this class.

**package** com.stech.test.bean;

**public** **class** User {

**private** **int** id;

**private** String userName;

**private** String password1;

**private** String email;

**private** String phone;

**private** String city;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getUserName() {

**return** userName;

}

**public** **void** setUserName(String userName) {

**this**.userName = userName;

}

**public** String getPassword1() {

**return** password1;

}

**public** **void** setPassword1(String password1) {

**this**.password1 = password1;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getPhone() {

**return** phone;

}

**public** **void** setPhone(String phone) {

**this**.phone = phone;

}

**public** String getCity() {

**return** city;

}

**public** **void** setCity(String city) {

**this**.city = city;

}

}

**UserDAO.java**

This is DAO class. In this class we have implemented addUserDetails() method where we are adding the user details to database by using hibernate

Create a package com.stech.test.dao, in this package create java class UserDAO.java and add following code in this class.

**package com.stech.test.dao;**

**import org.hibernate.HibernateException;**

**import org.hibernate.Session;**

**import org.hibernate.SessionFactory;**

**import org.hibernate.Transaction;**

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

**import com.stech.test.bean.User;**

**public class UserDAO {**

**public void addUserDetails(int userId, String userName, String password, String email, String phone, String city) {**

**try {**

**// 1. configuring hibernate**

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

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

**// 2. create sessionfactory**

**SessionFactory sessionFactory = configuration.buildSessionFactory();**

**// 3. Get Session object**

**Session session = sessionFactory.openSession();**

**// 4. Starting Transaction**

**Transaction transaction = session.beginTransaction();**

**User user = new User();**

**user.setId(userId);**

**user.setUserName(userName);**

**user.setPassword1(password);**

**user.setEmail(email);**

**user.setCity(city);**

**user.setPhone(phone);**

**session.save(user);**

**transaction.commit();**

**System.out.println("\n\n Details Added \n");**

**} catch (HibernateException e) {**

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

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

**}**

**}**

**}**

Create a package com.stech.test.ctrl and create a java class UserControllerServlet in this package and add following code in this class.

**UserControllerServlet.java**

**package** com.stech.test.ctrl;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** javax.servlet.http.HttpSession;

**import** com.stech.test.dao.UserDAO;

**public** **class** UserControllerServlet **extends** HttpServlet {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doPost(HttpServletRequest request,

HttpServletResponse response) **throws** ServletException, IOException {

**int** userId =Integer.*parseInt*(request.getParameter("userid"));

String userName = request.getParameter("userName");

String password = request.getParameter("password1");

String email = request.getParameter("email");

String phone = request.getParameter("phone");

String city = request.getParameter("city");

HttpSession session = request.getSession(**true**);

**try** {

UserDAO userDAO = **new** UserDAO();

userDAO.addUserDetails(userId,userName, password, email, phone, city);

response.sendRedirect("Success");

} **catch** (Exception e) {

e.printStackTrace();

}

}

}

**Success.java:**

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

**package** com.stech.test.ctrl;

**import** java.io.IOException;

**import** java.io.PrintWriter;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServlet;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**public** **class** Success **extends** HttpServlet {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**protected** **void** doGet(HttpServletRequest request,

HttpServletResponse response) **throws** ServletException, IOException {

PrintWriter writer = response.getWriter();

writer.println("" + "" + "<center>"

+ "Details Added Successfully" + "</center>" + ""

+ "");

}

}

**Step 5 : Create the mapping file :**

Mapping file maps the User class with the table of the database.

Right click on **com.visix.mnrao.bean** then navigate to **New -> Other -> General -> Next** and provide the name of file as **user.hbm.xml** and click on finish. After creating user.hbm.xml add following code in this file.

**user.hbm.xml**:

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

<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"

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

<hibernate-mapping>

<class name=*"com.stech.test.bean.User"* table=*"USERDATA"*>

<id column=*"USER\_ID"* name=*"id"* type=*"java.lang.Integer"* />

<property column=*"USER\_NAME"* name=*"userName"* type=*"java.lang.String"* />

<property column=*"USER\_PASSWORD"* name=*"password1"* type=*"java.lang.String"* />

<property column=*"EMAIL"* name=*"email"* type=*"java.lang.String"* />

<property column=*"PHONE"* name=*"phone"* type=*"java.lang.String"* />

<property column=*"CITY"* name=*"city"* type=*"java.lang.String"* />

</class>

</hibernate-mapping>

## Step 6 : Create the Configuration file :

The configuration file contains informations about the database and mapping file. Conventionally, its name should be **hibernate.cfg.xml** .Configuration file must be in classpath of your Project. Place this file in src of your project by default it will added to classpath of your project.

<?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=*"hibernate.dialect"*>org.hibernate.dialect.Oracle9Dialect</property>

<property name=*"hibernate.connection.driver\_class"*>oracle.jdbc.driver.OracleDriver</property>

<property name=*"hibernate.connection.url"*>jdbc:oracle:thin:@localhost:1521:ORCL</property>

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

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

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

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

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

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

</session-factory>

</hibernate-configuration>

## Step 7: Create web.xml file :

Create web.xml file inside **WEB-INF** directory of project.

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

<web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns=*"http://java.sun.com/xml/ns/javaee"* xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd"* id=*"WebApp\_ID"* version=*"3.0"*>

<display-name>test</display-name>

<welcome-file-list>

<welcome-file>register.jsp</welcome-file>

</welcome-file-list>

<servlet>

<servlet-name>User</servlet-name>

<servlet-class>com.stech.test.ctrl.UserControllerServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>User</servlet-name>

<url-pattern>/register</url-pattern>

</servlet-mapping>

<servlet>

<display-name>Success</display-name>

<servlet-name>Success</servlet-name>

<servlet-class>com.stech.test.ctrl.Success</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Success</servlet-name>

<url-pattern>/Success</url-pattern>

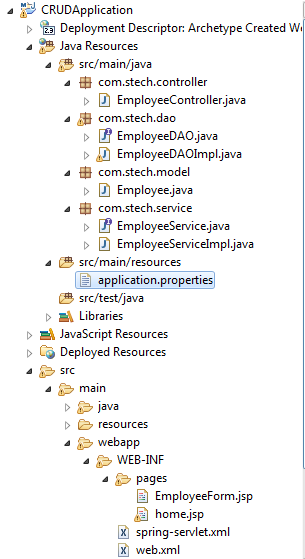
</servlet-mapping>

# </web-app>

# 

## Run the Application :

To run the hibernate web application, right click on the project then Run as -> Run On Server select Tomcat -> Next ->Finish.





Create database and table in Mysql:

Mysql> Create database spring;

Mysql> use spring;

Mysql> CREATE TABLE `EMP\_TBL` (

`id` int(11) NOT NULL AUTO\_INCREMENT,

`name` varchar(45) NOT NULL,

`email` varchar(45) NOT NULL,

`address` varchar(45) NOT NULL,

`telephone` varchar(45) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=1 DEFAULT CHARSET=utf8;

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/maven-v4\_0\_0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>com.sailotech</groupId>

<artifactId>MVCSPRINGHBNET</artifactId>

<packaging>war</packaging>

<version>0.0.1-SNAPSHOT</version>

<name>MVCSPRINGHBNET Maven Webapp</name>

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

<!-- Specifying the Versions of Spring, Hiberante, MySQL etc -->

<properties>

<spring.version>4.1.5.RELEASE</spring.version>

<hibernate.version>4.3.8.Final</hibernate.version>

<mysql.version>5.1.10</mysql.version>

<junit-version>4.11</junit-version>

<servlet-api-version>3.1.0</servlet-api-version>

<jsp-version>2.1</jsp-version>

<jstl-version>1.2</jstl-version>

<java.version>1.7</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-orm</artifactId>

<version>${spring.version}</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>${spring.version}</version>

<scope>test</scope>

</dependency>

<!-- Hibernate 4 dependencies -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>${hibernate.version}</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-c3p0</artifactId>

<version>${hibernate.version}</version>

</dependency>

<!--MYSQL Connector -->

<dependency>

<groupId>mysql</groupId>

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

<version>${mysql.version}</version>

</dependency>

<!-- Servlet and JSP -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>${servlet-api-version}</version>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>jsp-api</artifactId>

<version>${jsp-version}</version>

<scope>provided</scope>

</dependency>

<!-- JSTL dependency -->

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>${jstl-version}</version>

</dependency>

<!-- JUnit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit-version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>MVCSPRINGHBNET</finalName>

</build>

</project>

WEB-INF/Web.xml

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

<web-app>

<display-name>Archetype Created Web Application</display-name>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring-servlet.xml</param-value>

</context-param>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

Src/main/resources :

File : application.properties

#Database related properties

database.driver=com.mysql.jdbc.Driver

database.url=jdbc:mysql://localhost:3306/spring?characterEncoding=latin1&useConfigs=maxPerformance

database.user=root

database.password=root

#Hibernate related properties

hibernate.dialect=org.hibernate.dialect.MySQLDialect

hibernate.show\_sql=true

hibernate.format\_sql=true

hibernate.hbm2ddl.auto=update

WEB-INF/ spring-servlet.xml

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"* xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd*

*http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>

<!-- Specifying base package of the Components like Controller, Service,

DAO -->

<context:component-scan base-package=*"com.stech"* />

<!-- Getting Database properties -->

<context:property-placeholder location=*"classpath:application.properties"* />

<mvc:annotation-driven />

<!-- Specifying the Resource location to load JS, CSS, Images etc -->

<mvc:resources mapping=*"/resources/\*\*"* location=*"/resources/"* />

<!-- View Resolver -->

<bean

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/WEB-INF/pages/"* />

<property name=*"suffix"* value=*".jsp"* />

</bean>

<!-- DataSource -->

<bean class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*

id=*"dataSource"*>

<property name=*"driverClassName"* value=*"${database.driver}"*></property>

<property name=*"url"* value=*"${database.url}"*></property>

<property name=*"username"* value=*"${database.user}"*></property>

<property name=*"password"* value=*"${database.password}"*></property>

</bean>

<!-- Hibernate SessionFactory -->

<bean id=*"sessionFactory"*

class=*"org.springframework.orm.hibernate4.LocalSessionFactoryBean"*>

<property name=*"dataSource"* ref=*"dataSource"*></property>

<property name=*"hibernateProperties"*>

<props>

<prop key=*"hibernate.dialect"*>${hibernate.dialect}</prop>

<prop key=*"hibernate.hbm2ddl.auto"*>${hibernate.hbm2ddl.auto}</prop>

<prop key=*"hibernate.format\_sql"*>${hibernate.format\_sql}</prop>

<prop key=*"hibernate.show\_sql"*>${hibernate.show\_sql}</prop>

</props>

</property>

<property name=*"packagesToScan"* value=*"com.stech.model"*></property>

</bean>

<!-- Transaction -->

<bean id=*"transactionManager"*

class=*"org.springframework.orm.hibernate4.HibernateTransactionManager"*>

<property name=*"sessionFactory"* ref=*"sessionFactory"* />

</bean>

<tx:annotation-driven transaction-manager=*"transactionManager"* />

</beans>

Src/main/java

Package : com.stech.controller

**package** com.stech.controller;

**import** java.io.IOException;

**import** java.util.List;

**import** javax.servlet.http.HttpServletRequest;

**import** org.jboss.logging.Logger;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.ModelAttribute;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RequestMethod;

**import** org.springframework.web.servlet.ModelAndView;

**import** com.stech.model.Employee;

**import** com.stech.service.EmployeeService;

@Controller

**public** **class** EmployeeController {

@SuppressWarnings("unused")

**private** **static** **final** Logger ***logger*** = Logger.*getLogger*(EmployeeController.**class**);

**public** EmployeeController() {

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

}

@Autowired

**private** EmployeeService employeeService;

@RequestMapping(value = "/")

**public** ModelAndView listEmployee(ModelAndView model) **throws** IOException {

List<Employee> listEmployee = employeeService.getAllEmployees();

**for** (Employee employee : listEmployee) {

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

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

}

model.addObject("listEmployee", listEmployee);

model.setViewName("home");

**return** model;

}

@RequestMapping(value = "/newEmployee", method = RequestMethod.***GET***)

**public** ModelAndView newContact(ModelAndView model) {

Employee employee = **new** Employee();

model.addObject("employee", employee);

model.setViewName("EmployeeForm");

**return** model;

}

@RequestMapping(value = "/saveEmployee", method = RequestMethod.***POST***)

**public** ModelAndView saveEmployee(@ModelAttribute Employee employee) {

**if** (employee.getId() == 0) { // if employee id is 0 then creating the

// employee other updating the employee

employeeService.addEmployee(employee);

} **else** {

employeeService.updateEmployee(employee);

}

**return** **new** ModelAndView("redirect:/");

}

@RequestMapping(value = "/deleteEmployee", method = RequestMethod.***GET***)

**public** ModelAndView deleteEmployee(HttpServletRequest request) {

**int** employeeId = Integer.*parseInt*(request.getParameter("id"));

employeeService.deleteEmployee(employeeId);

**return** **new** ModelAndView("redirect:/");

}

@RequestMapping(value = "/editEmployee", method = RequestMethod.***GET***)

**public** ModelAndView editContact(HttpServletRequest request) {

System.***out***.println("ID ="+request.getParameter("id"));

**int** employeeId = Integer.*parseInt*(request.getParameter("id"));

Employee employee = employeeService.getEmployee(employeeId);

ModelAndView model = **new** ModelAndView("EmployeeForm");

model.addObject("employee", employee);

**return** model;

}

}

Package :

Src/java/main :

Package : com.stech.service

**package** com.stech.service;

**import** java.util.List;

**import** com.stech.model.Employee;

**public** **interface** EmployeeService {

**public** **void** addEmployee(Employee employee);

**public** List<Employee> getAllEmployees();

**public** **void** deleteEmployee(Integer employeeId);

**public** Employee getEmployee(**int** employeeid);

**public** Employee updateEmployee(Employee employee);

}

package com.stech.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.stech.dao.EmployeeDAO;

import com.stech.model.Employee;

@Service

@Transactional

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeDAO employeeDAO;

@Transactional

public void addEmployee(Employee employee) {

employeeDAO.addEmployee(employee);

}

@Transactional

public List<Employee> getAllEmployees() {

return employeeDAO.getAllEmployees();

}

@Transactional

public void deleteEmployee(Integer employeeId) {

employeeDAO.deleteEmployee(employeeId);

}

public Employee getEmployee(int empid) {

return employeeDAO.getEmployee(empid);

}

public Employee updateEmployee(Employee employee) {

// TODO Auto-generated method stub

return employeeDAO.updateEmployee(employee);

}

public void setEmployeeDAO(EmployeeDAO employeeDAO) {

this.employeeDAO = employeeDAO;

}

}

Src/main/java

Package : com.stech.model

**package** com.stech.model;

**import** java.io.Serializable;

**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 = "EMP\_TBL")

**public** **class** Employee **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = -3465813074586302847L;

@Id

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

**private** **int** id;

@Column

**private** String name;

@Column

**private** String email;

@Column

**private** String address;

@Column

**private** String telephone;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

**public** String getTelephone() {

**return** telephone;

}

**public** **void** setTelephone(String telephone) {

**this**.telephone = telephone;

}

}

Src/java/main

Package: com.stech.dao

**package** com.stech.dao;

**import** java.util.List;

**import** com.stech.model.Employee;

**public** **interface** EmployeeDAO {

**public** **void** addEmployee(Employee employee);

**public** List<Employee> getAllEmployees();

**public** **void** deleteEmployee(Integer employeeId);

**public** Employee updateEmployee(Employee employee);

**public** Employee getEmployee(**int** employeeid);

}

**package** com.stech.dao;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.hibernate.SessionFactory;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Repository;

**import** com.stech.model.Employee;

@Repository

**public** **class** EmployeeDAOImpl **implements** EmployeeDAO {

@Autowired

**private** SessionFactory sessionFactory;

**public** **void** addEmployee(Employee employee) {

sessionFactory.getCurrentSession().saveOrUpdate(employee);

}

@SuppressWarnings("unchecked")

**public** List<Employee> getAllEmployees() {

List <Employee >list = **new** ArrayList<Employee>();

list = sessionFactory.getCurrentSession().createQuery("from Employee")

.list();

**return** list;

}

**public** **void** deleteEmployee(Integer employeeId) {

Employee employee = (Employee) sessionFactory.getCurrentSession().load(

Employee.**class**, employeeId);

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

**this**.sessionFactory.getCurrentSession().delete(employee);

}

}

**public** Employee getEmployee(**int** empid) {

**return** (Employee) sessionFactory.getCurrentSession().get(

Employee.**class**, empid);

}

**public** Employee updateEmployee(Employee employee) {

sessionFactory.getCurrentSession().update(employee);

**return** employee;

}

}