1. Introduction

2. Pom.xml

3. Project Description.

4. Repository.

5. Dependency Management.

6. Build Configurations.

7.Build Profiles

8.Steps to prepare standalone project in Maven.

9. JDBC Application using maven and Eclipse IDE.

10. Develop Web Application with out DB using maven and Eclipse IDE.

11.Develop web Application with DB using maven and Eclipse IDE.

12. Develop Hybernet Application using maven.

**1.Introduction:-** Maven is java build tool. It simplifies the java project development, java project testing …etc. It does following.

1. Generates the source code.
2. Generates documents from source code.
3. Pack the compiled code into jar,war,ear and …etc.

.jar :: for standalone application.

.war:: for web application.

.ear:: for enterprise application.

1. Downloads the different APIs from maven repositories dynamically from internet and integrates to maven project.
2. This tool provides 2000+ ready made project directory structures(archetypes).

Maven-archetype-quickstart 🡪For standalone applications.

Maven-archetype-webapp 🡪 For web applications.

1. This readymade directory structure have following directories which is used to place source code and different files.

src/main/java 🡪 place packages and source code.

src/test/java 🡪 place source code and packages of unit testing.

src/main/webapp 🡪place the html files, js files , image files and ..etc.

src/main/resources 🡪 place the properties files and packages.

**2.Pom.xml:-**

Pom stands for project object model.

It is xml file. It contains the maven project information and configuration details.

In Maven 1.0 version, this file name was “project.xml” and in Maven 2,0 version, this file name was changed as “pom.xml”.

When maven project execution is started, automatically ,maven gets all configurations from “pom.xml” file.

This pom.xml contains following configurations:

1. Project Description
2. Repositatory.
3. Dependency Management.
4. Project Inheritance.
5. Build Profile.
6. Build Configurations.

**3. Project Description:-** In pom.xml, we specify maven project details using set of tags. The <project> tag is root tag in pom.xml.



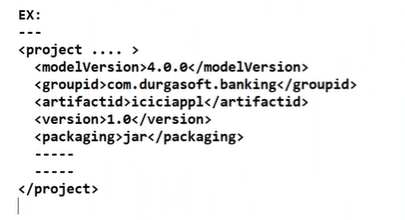
We declare the maven version which we are going to use by <modelVersion> tag. <modelVersion> will take 4.0.0 as value to support for maven2.x/3.x versions.

<groupid> tag will take id of project or base package name.

We declare the name of the project through the <artifactid> tag.

<version> tag will take project version.

<packaging> tag will take either jar/war/ear to deliver our application.



**4. Repository:-** Repository is place where several dependencies are stored. From the repository, maven download required dependencies and integrates to maven project. When we execute the maven project, maven will search for required dependencies in following 3 repositories.

1. Local Repository.

2. Central Repository.

3. Remote Repository.

**4.1.Local Repository:-**  It is location in our computer. It will be created by maven when we execute the maven command first time.

**4.2.Central Repository:-** It is a default repository for Maven, It is located at <http://repo1.maven.org/maven2>.

It is also possible to use some other repositories also in Maven, but we must configure that repositors in pom.xml.



Example:



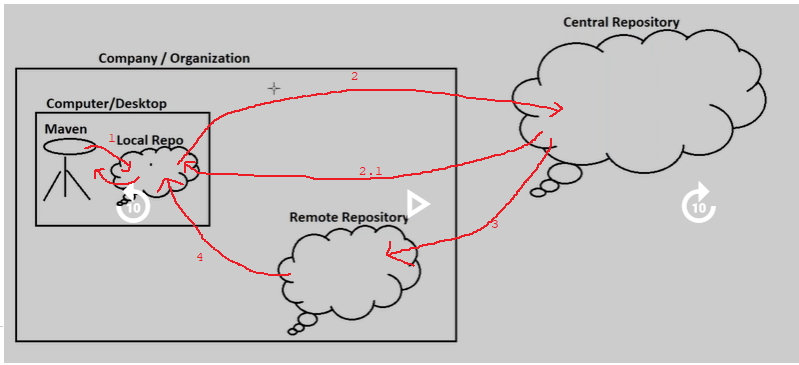
**4.3. Remote Repository:-** Every software company maintains its own repository. The company maintains their own repository in order to provide previous application support.

To configure remote repository in maven project we have use following configuration details in pom.xml file.

Example:

======





Step1: First Maven searches local repository for required dependencies. If required dependencies are in local repo then maven uses them in maven project execution otherwise maven does the second step.

Step2: Maven searches local repository for required dependencies. If required dependencies are present in central repository then maven down loads them into local repository and uses them. Otherwise

If remote repository details are specified in pom.xml, as per that configuration maven will searches remote repository otherwise maven will stop the execution of project and arises the exception.

If required dependencies does not be in remote computer then also maven stop the execution of project and arises the exception otherwise maven bring required dependencies into local repo and uses them at the time of project execution.

**5. Dependency Management:-** Dependency is nothing but a **jar file**.

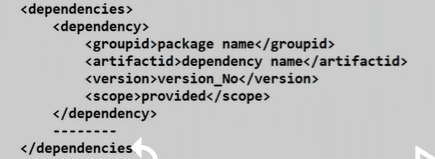
Problem:- We explicitly download jar files from the internet and keep them in Project/Application library. These jar file will be used in application compilation or application execution.

The maven given solution to this problem.

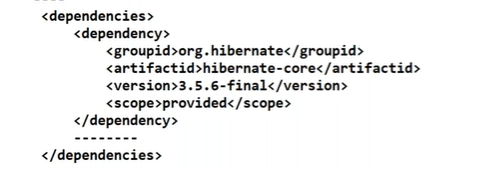
If we use maven for application development and executing then we will specify the required dependency details in pom.xml file instead of downloading them explicitly. In this context, maven takes responsibilities of downloading required libraries and attaching these libraries to application/project library.

To configure dependencies in pom.xml, we should use set of tags:

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



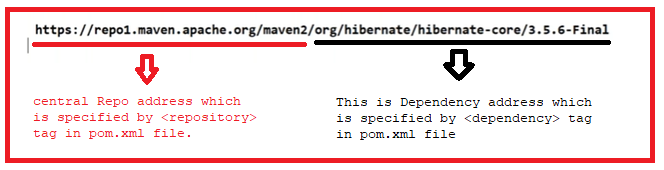
Example:-



Note:- The maven search for this dependency in specified address.

Central repository/dependency details.

Example:­



**5.1. Dependency Scope:-**  The maven gave six keyword related to dependency scope. The key word specifies that in which stage[s] of project life cycle , Dependency is available and in which stage[s] of project life cycle, Dependency is not available.

Key words are

1. Compile
2. Provided
3. Runtime
4. Test
5. System
6. Import

**a.Compile**:- It is default scope in maven project. This keyword make available the dependency in all life cycle stages/phases of maven project.

**b. Provided:-**  This keyword makes available the dependency only in compilation and test phase of maven project.

**c.RunTime:-** This keyword makes available the dependency only in execution phase of maven project.

**d.Test:-** This keyword makes available the dependency only in test phase of maven project.

Ex:- JUNIT , Mockito Dependencies.

**e. System:-**  when the required dependency is not in central repository, maven will not go to remote repository for searching. Instead of ,maven searches sub-directory of maven project for required jars/dependencies.

Example:



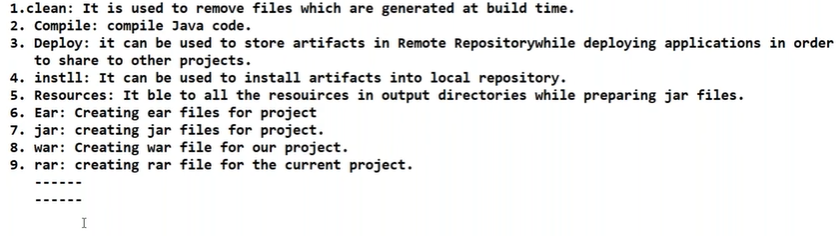
**6.Build Configurations:-**

The plug-in is small installable program . It extends/enhancement the functionality of software. There are two types of plug-in maven.

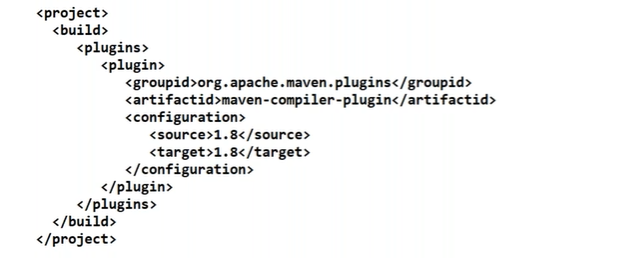
1. Build plugins
2. Remote plugins.

6.1. Build Plug-in:- The build plug-ins are configured with <build> tag in pom.xml file.

Examples:



Example configuration to compiler plug-in:



**6.2. Remote plugins:-**  They execute during the site generation process and they should be configured in the <reporting/> element of the pom.xml.

**7.Build Profiles:-** Build profile is a set of configuration values, which can be used to set or override default values of maven build.

**Types of Build Profile:**

1. **Per Project:** Defined in the project POM.xml.
2. **Per User:** Defined in Maven settings xml file.

(%USER\_HOME%/.m2/settings.xml).

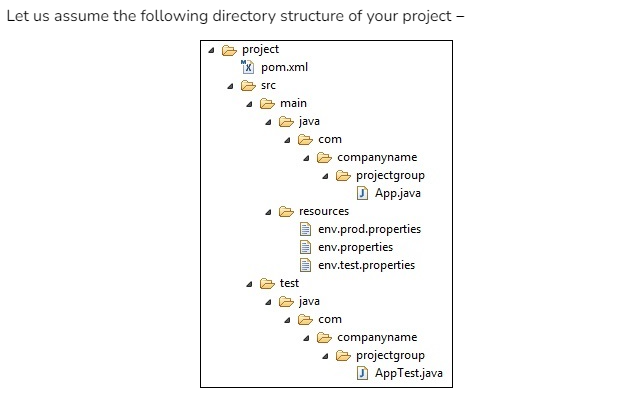
1. **Global:** Defined in Maven global setting xml file.

(%M2\_HOME%/conf/settings.xml).

**Profile Activation: A** Maven build profile can be activated in various ways.

* Explicitly using command console input.
* Through maven settings.
* Based on environment variables(User/System variables).
* OS settings.

**Profile Activation Examples:**

****

**There are 3 environment specific files-**

1. **Env.properties:** default configuration used if no profile is mentioned**.**
2. **Env.test.properties:** test configuration when test profile is used**.**
3. **Env.prod.properties:** production configuration when prod profile is used**.**

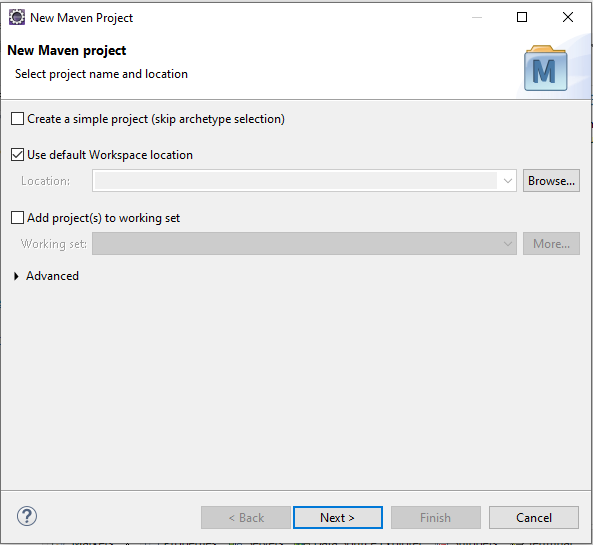
**This topic is pending.**

**8.Procedure to Develop the Standardalone Application using Eclipse + maven tool:**

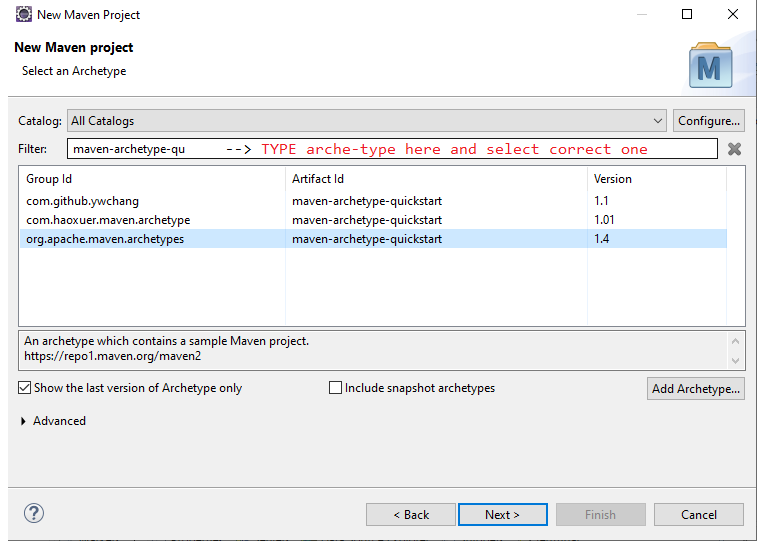
**Step1:** Launch the eclipse IDE.

**Step 2:** Create Maven project.

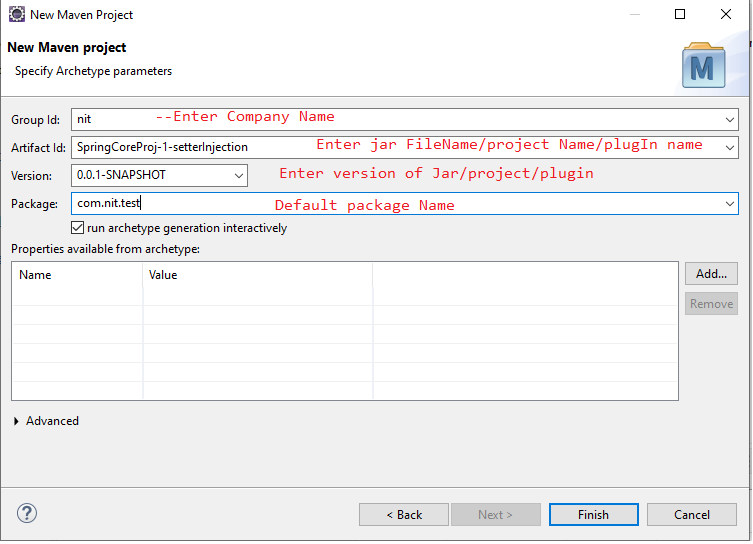
File🡪new 🡪New Maven project



2.1.Click on next button.

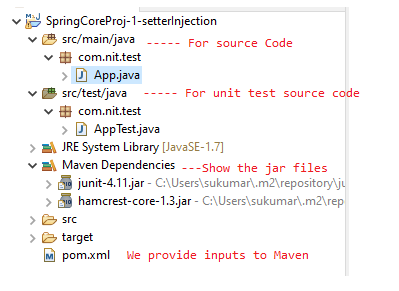


2.2 click the Next button.



2.3. click the Finish Button.

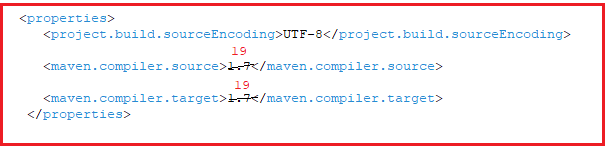
It gives following directory structure for SpringCoreProj-1setterInjection.



2.4. change JRE System Library version 1.7 to 19 or current java version.

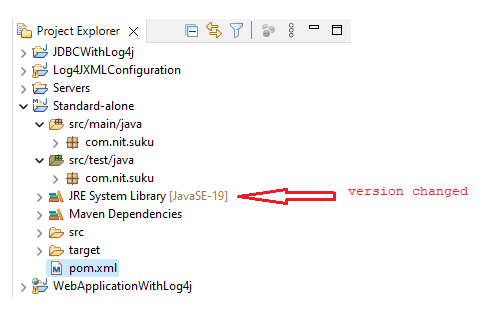
=> Open the pom.xml file

=> change maven.compiler.source and maven.compiler.target tag values as 19.



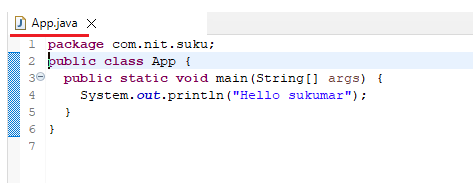
2.5. Update the maven project.

Select project 🡪Right click on it 🡪maven 🡪 update project.

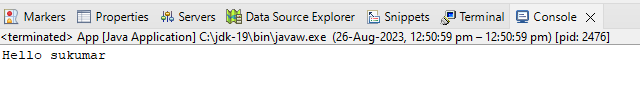


Step3: we should write application logic only in src/main/java package.

The com.nit.suku package has App.java by default. In that file , we may write logic or we may create another .java file for writing the application logic.



Output:

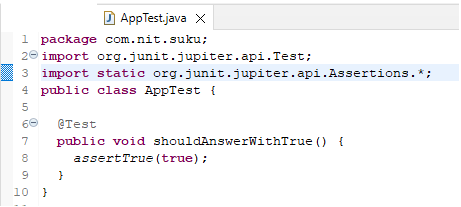


Execution Step to Run only Application logic of maven project:

a.select the project and press F11 button. (or)

b. select the project and press “execution” icon.

**Step4:** This is optional step. We do testing on application logic. That testing is unit testing. The unit testing logic should be written only in src/test/java. It has one sub package which com.nit.suku. This package has only AppTest.java file. In that file , we write unit testing logic.



Execution Steps to Run only Application logic of maven project:

Select project 🡪 right click on it 🡪 Run as 🡪 MavenTest

Output:



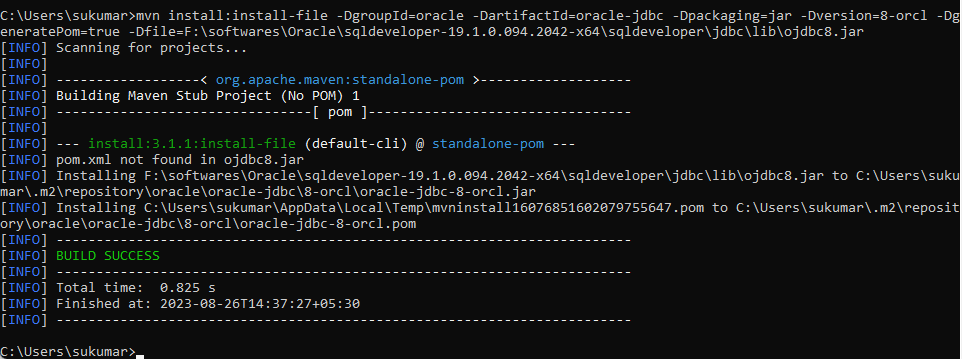
Drawback:-

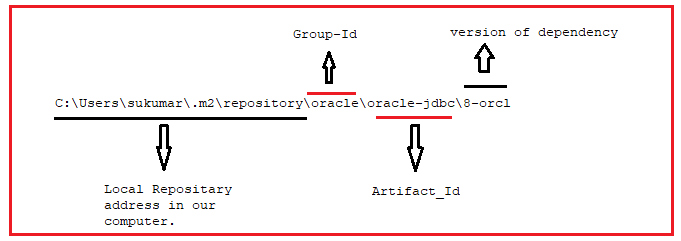
1. The maven displays the test report on console.
2. It does not give In-Detailed test reports.

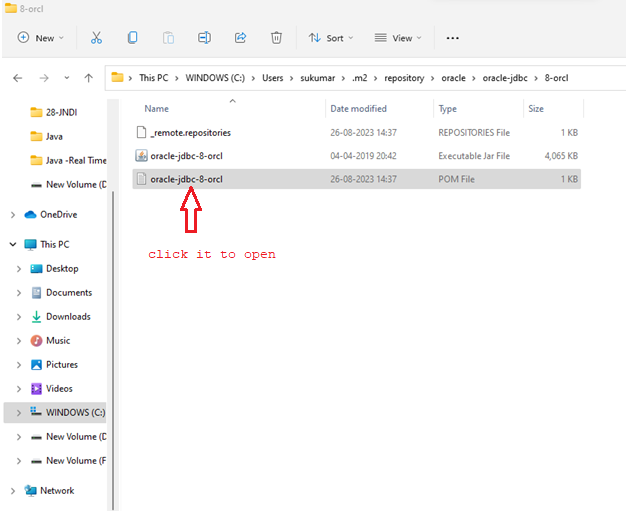
**9. JDBC Application using maven with Eclipse IDE:**

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

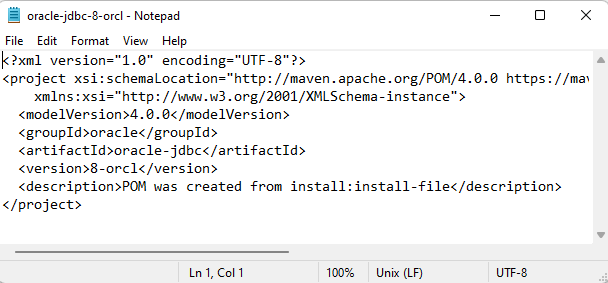
**Step1:**Prepare ojdbc.6/7/8… file dependency in local repository.

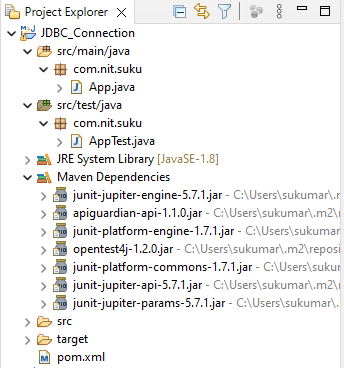




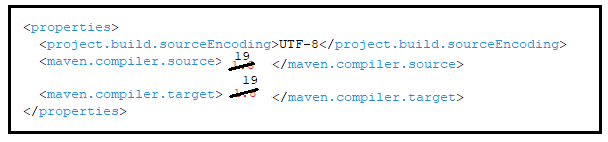


Maven prepares pom.xml file for this dependency:

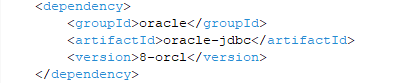
 Step2: prepare maven project in Eclipse IDE.



Change the maven compiler version.



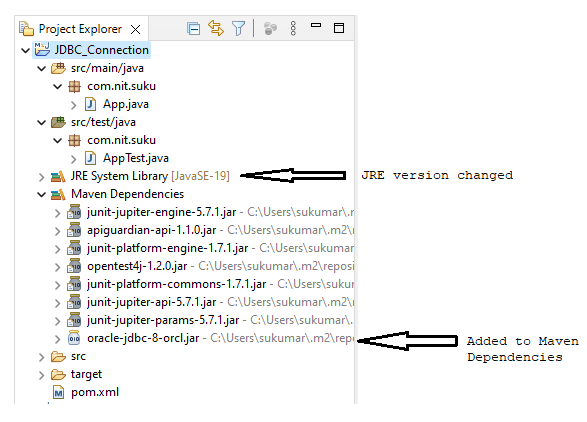
Add the ojdbc8.jar dependency to pom.xml. This dependency exists in local repository.



Update the project

Select projct🡪right click on it 🡪 maven🡪update project.

After updating project, you should observe the changes in project directory structure.



Step3: Write JDBC logic in Maven project.

package com.nit.suku;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.sql.Statement;

public class App {

public static void main(String[] args) {

Connection con=null;

Statement s1=null;

ResultSet r1=null;

try {

//Class.forName("oracle.jdbc.driver.OracleDriver");

con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","hms","tiger");

if(con!=null) {

s1=con.createStatement();

}

if(s1!=null)

r1=s1.executeQuery("Select \* from person");

if(r1!=null)

{

while(r1.next()) {

System.out.println(r1.getInt(1)+" "+r1.getString(2));

}

}

}

catch(SQLException e)

{

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

}

catch(Exception e)

{

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

}

finally {

try {

r1.close();

s1.close();

con.close();

} catch (SQLException e) {

e.getMessage();

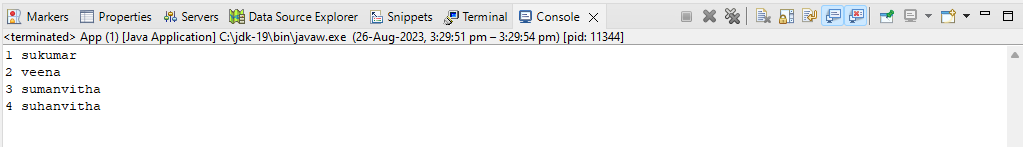
}

}

}

}

Step4: Execute project.



**10. Develop Web Application with out DB using maven and Eclipse IDE:**

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

**Step:1** Create web project in maven.

**Step:2** Provide configurations in pom.xml

Step:3 Update project.

Step:4 prepare web components(servlet components,jsp components,html components).

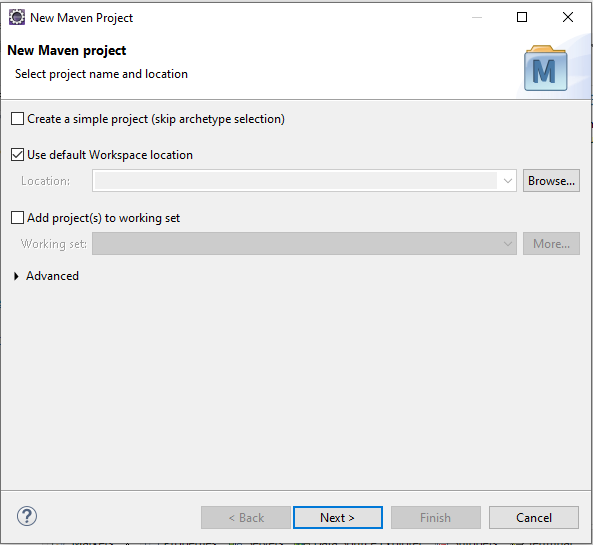
Step:5 Run Web Application.

**Step1:-** Create web project in maven.

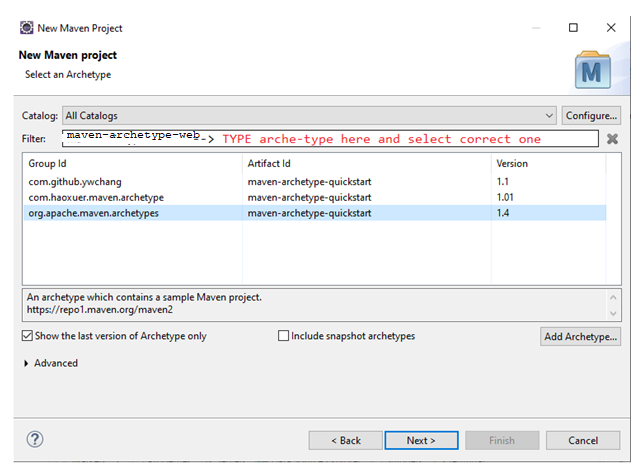
**a.** Launch the eclipse IDE.

**b.** Create Maven project.

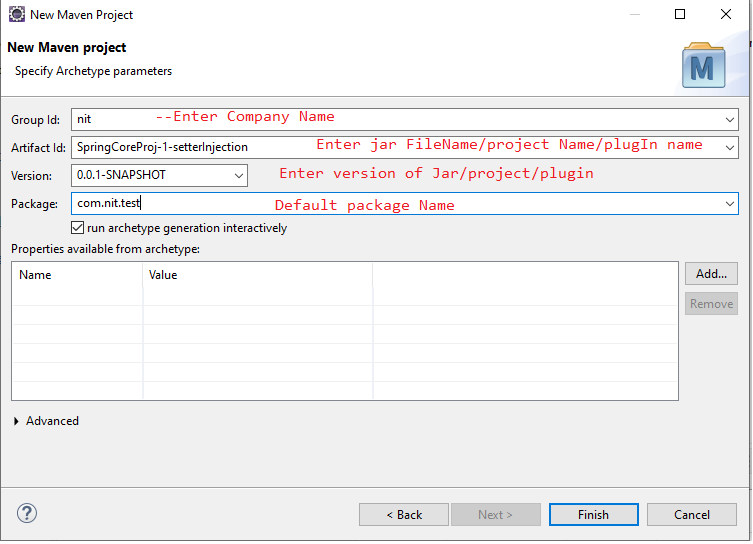
File🡪new 🡪New Maven project



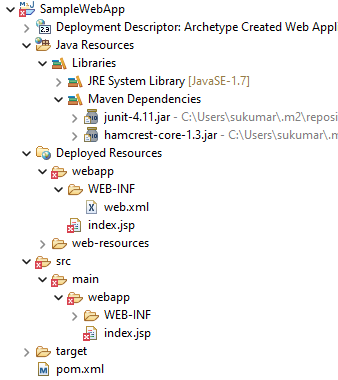
c..Click on next button.



2.2 click the Next button.

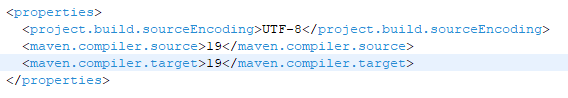


Click the finish Button. The project Structure will be as below:



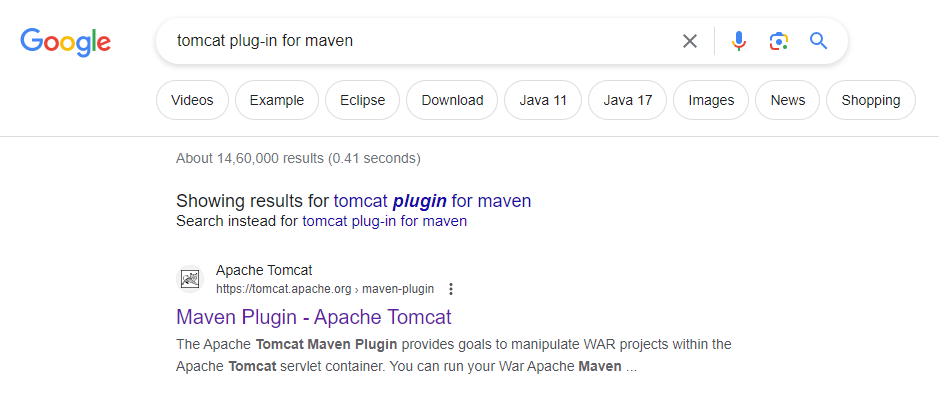
Step2:-

1. Change maven-compiler-source version number and maven-compiler-target version number in pom.xml file.

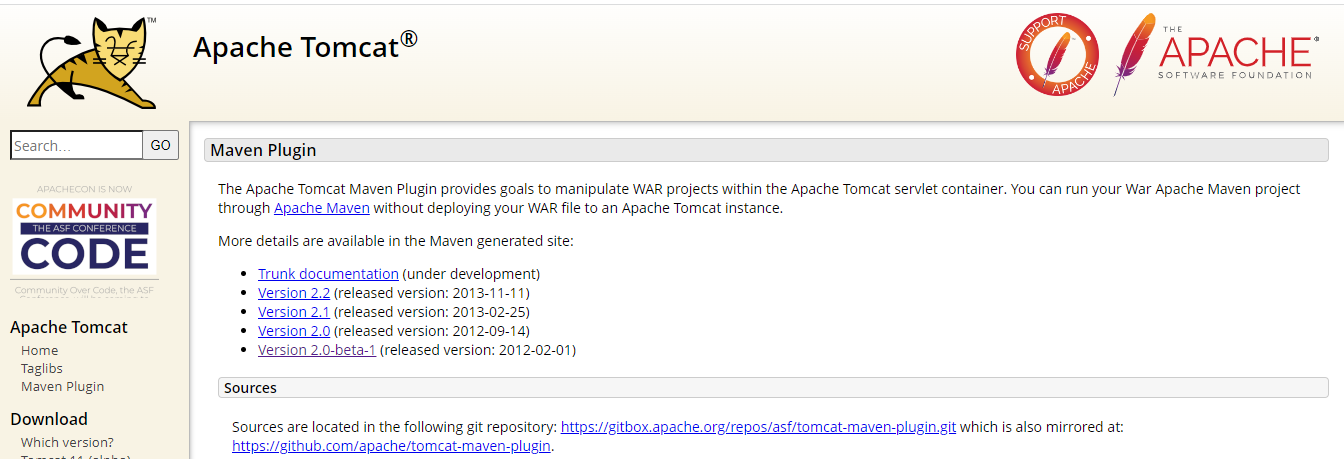


1. Add tomcat plug-in dependency or tomcat plug-in to pom.xml file.

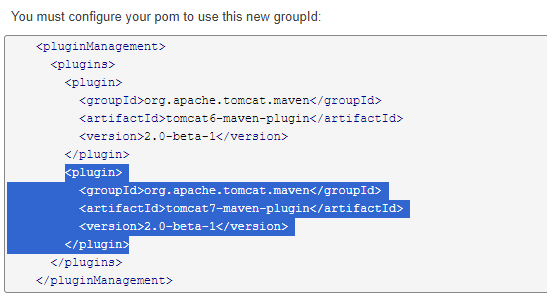
Bring plug-in or plug-in dependency from internet.



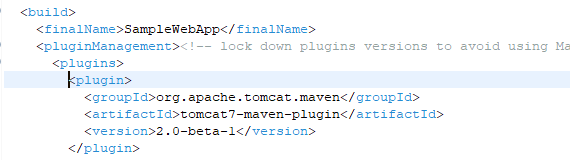
Click-on “Apache Tomcat”



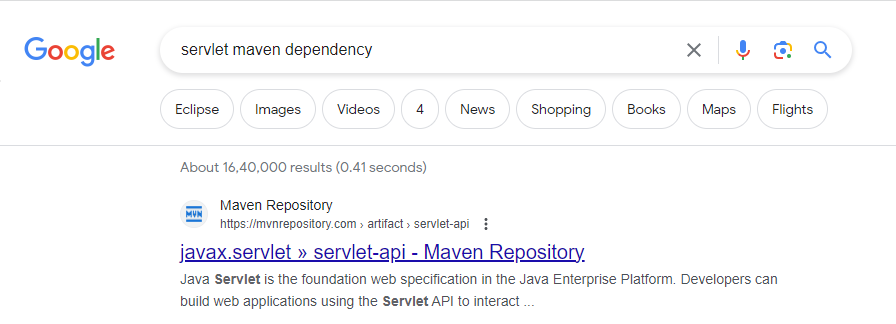
Click on “version2-0-beta-1



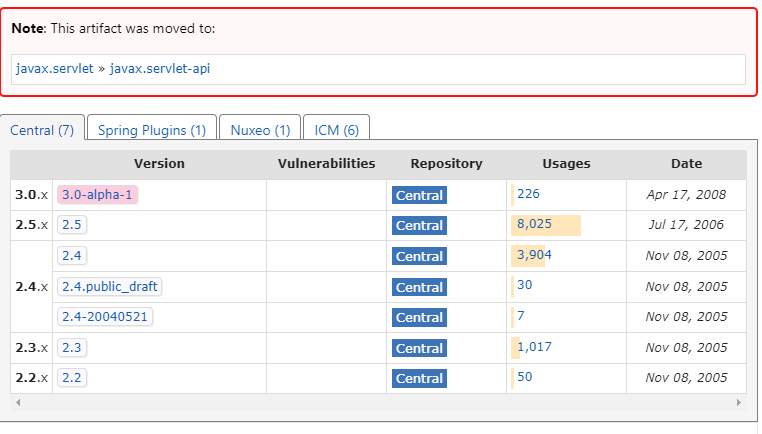
Copy the selected part and paste in <plugins> tag of pom.xml file



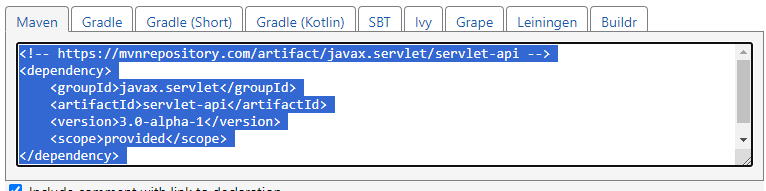
1. Add servlet API dependency to pom.xml file.



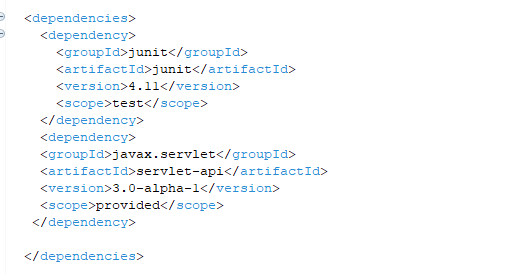
Click on “Maven Repository”.



Click on “3.0-alpha-1”



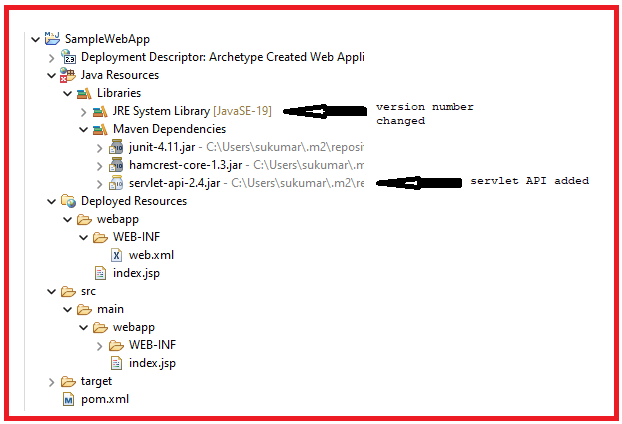
Paste the selected part in <dependencies> tag of pom.xml



**Step3: Update the maven project**

Select project🡪right click🡪maven🡪update project

After updating , observe the project directory structure.



**Step4: Develop the web components.**

Index.html

=========

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>Insert title here</title>

</head>

<body>

<h2>HelloWorld</h2>

<a href="./Display1"> Go To Servlet</a>

</body>

</html>

Display1.java

==========

**package** com.suku.serv;

**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** Display1 **extends** HttpServlet {

**protected** **void** doGet(HttpServletRequest req, HttpServletResponse res) **throws** ServletException, IOException {

PrintWriter pw=res.getWriter();

res.setContentType("text/html");

pw.write("<h2>Servlet Arrived</h2> ");

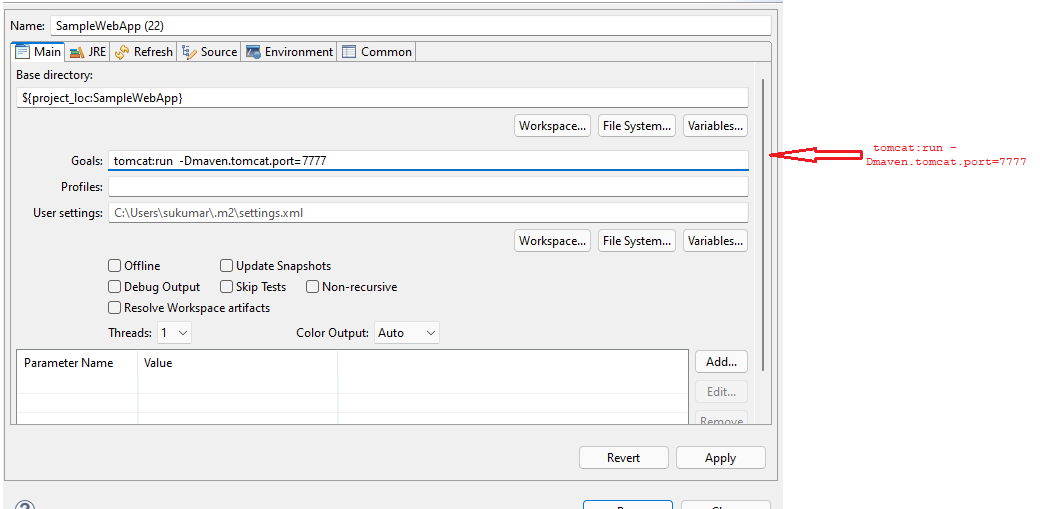
}

}

Update the maven project.

**Step5: Run the Maven project.**

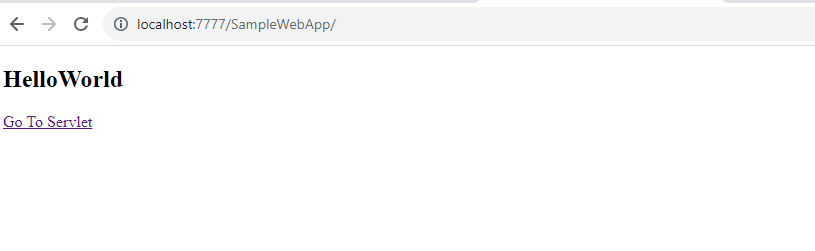
Select project🡪 run as 🡪 maven build🡪



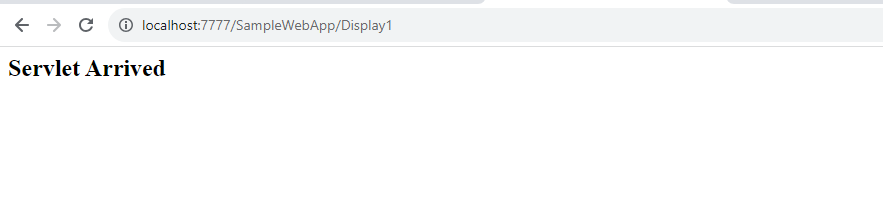
Press the run button.

Next Open the browser and enter the following url in address bar.

<http://localhost:7777/SampleWebapp/index.html>



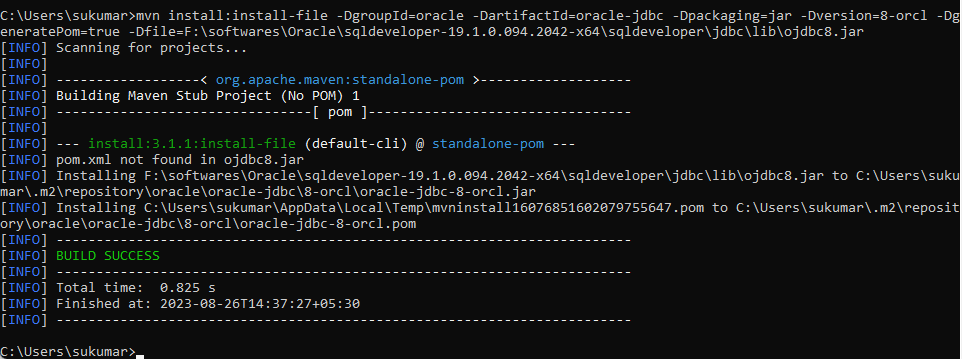
Click GoToServlet

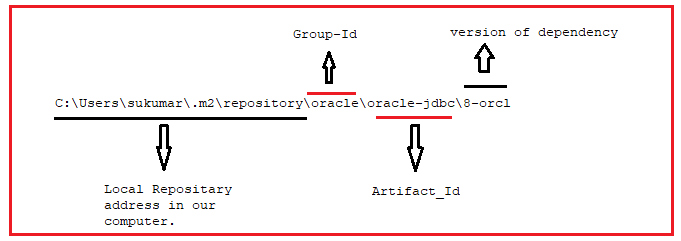


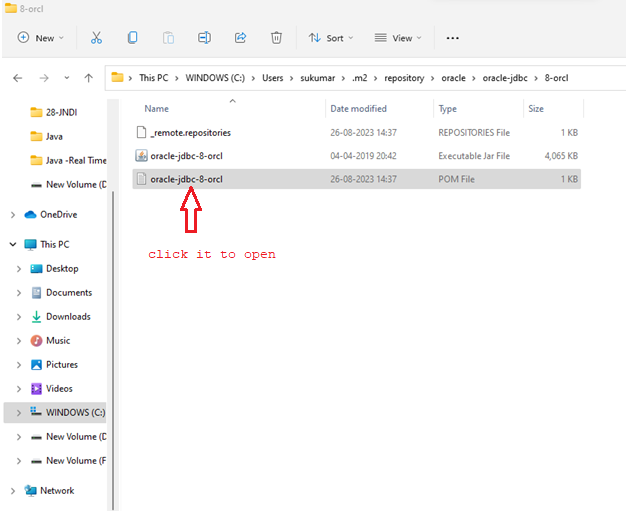
**11.Develop web Application with DB using maven and Eclipse IDE:**

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

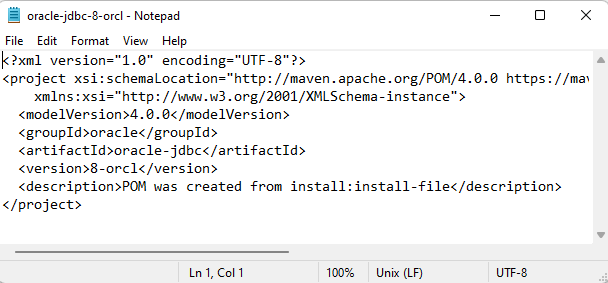
**Step1:**Prepare ojdbc.6/7/8… file dependency in local repository.







Maven prepares pom.xml file for this dependency:



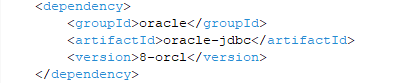
Step2: prepare maven web project in Eclipse IDE.

**Step:3** Provide configurations in pom.xml.

1. Change maven-compiler-source version number and maven-compiler-target version number in pom.xml file.
2. Add tomcat plug-in dependency or tomcat plug-in to pom.xml file.

Bring plug-in or plug-in dependency from internet.

1. Add servlet API dependency to pom.xml file.
2. Add the ojdbc8.jar dependency to pom.xml. This dependency exists in local repository.



**Step4: Update the maven project**

Select project🡪right click🡪maven🡪update project

After updating , observe the project directory structure.

Step:5 prepare web components(servlet components,jsp components,html components).

Step:6 Run Web Application.

**12. Develop Hybernet standalone Application using maven.**

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

**Note:- By default , hibernate does not support batch updates.** To enable, you must set the hibernate.jdbc.batch\_size property to value greater than zero. For example, in the hibernate.cfg.xml file:

**<property name=*"hibernate.jdbc.batch\_size"*>20</property>**

a. create Maven object.

b. Add following dependencies to pom.xml

**b.1** update maven compiler source and maven compiler target version number.

b.2 add oracle8.jar dependency

b.3 add hibernate dependencies.

b.4 add javassist dependencies.

c. Update the maven project.

d. Create POJO class.

e. Create Mapping File.

f. Create configuration File

g. Create the Client Application

h. Execute the Client Application.

Step2: Add following dependencies to pom.xml

**b.1** update maven compiler source and maven compiler target version number.

b.2 add oracle8.jar dependency

b.3 add hibernate dependencies.

b.4 add javassist dependencies.

b.5 add persistence-api dependencies.

<dependency>

<groupId>oracle</groupId>

<artifactId>oracle-jdbc</artifactId>

<version>8-orcl</version>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>3.6.10.Final</version>

</dependency>

<dependency>

<groupId>javassist</groupId>

<artifactId>javassist</artifactId>

<version>3.12.1.GA</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>2.0.7</version>

</dependency>

<dependency>

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

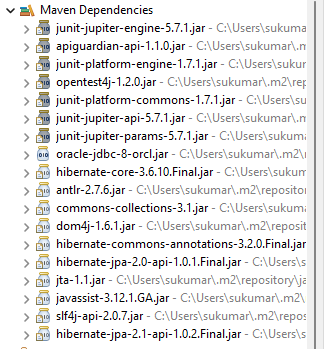
<artifactId>hibernate-jpa-2.1-api</artifactId>

<version>1.0.2.Final</version>

</dependency>

Step3: Update maven project.

Select project🡪right click on it 🡪 maven 🡪update project



Step4: Create POJO class/Domain class

**package** com.nit.suku;

**import** java.io.Serializable;

**public** **class** Emp **implements** Serializable {

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

**private** String ename;

**private** **float** sal;

**public** Emp() {

}

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

**return** eid;

}

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

**this**.eid = eid;

}

**public** String getEname() {

**return** ename;

}

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

**this**.ename = ename;

}

**public** **float** getSal() {

**return** sal;

}

**public** **void** setSal(**float** sal) {

**this**.sal = sal;

}

}

Step5: create Hibernate Mapping File.(Emp.hbm.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.connection.driver\_class"*>oracle.jdbc.OracleDriver</property>

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

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

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

<property name=*"hibernate.dialect"*>org.hibernate.dialect.Oracle10gDialect</property>

**<property name=*"hibernate.jdbc.batch\_size"*>20</property>**

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

</session-factory>

</hibernate-configuration>

Step6: : create hibernate Configuration File(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>

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

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

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

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

<property name=*"hibernate.dialect"*>org.hibernate.dialect.Oracle10gDialect</property>

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

</session-factory>

</hibernate-configuration>

**Step7:** Develop the client Application(Test.java).

**package** com.nit.suku;

**import** java.util.Scanner;

**import** org.hibernate.HibernateException;

**import** org.hibernate.MappingException;

**import** org.hibernate.Session;

**import** org.hibernate.SessionFactory;

**import** org.hibernate.Transaction;

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

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

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

**try** {

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

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

SessionFactory sf=cfg.buildSessionFactory();

Session sv=sf.openSession();

Scanner s1=**new** Scanner(System.***in***);

Transaction t=sv.beginTransaction();

Emp e1=**new** Emp();

System.***out***.print("Enter the eno: ");

e1.setEid(s1.nextInt());

s1.nextLine();

System.***out***.print("Enter the ename: ");

e1.setEname(s1.nextLine());

System.***out***.print("Enter the sal:");

e1.setSal(s1.nextFloat());

**int** pk=(Integer)sv.save(e1);

**if**(pk==e1.getEid()) {

System.***out***.println("record Inserted");

}

**else**

{

System.***out***.println("record not Inserted");

}

t.commit();

sv.close();

sf.close();

}

**catch**(MappingException e) {

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

}

**catch**(HibernateException e) {

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

}

}

}

Ouput:-

