Download and configure Mongo DB 3.2

Login as root

move to /etc/yum.repos.d using the cd command
in it create mongodb.repo file using the vi edit as follows

mongodb.repo

```
[MongoDB]
name=MongoDB Repository
baseurl=http://repo.mongodb.org/yum/redhat/$releasever/mongodb-org/3.2/x86_64/
gpgcheck=0
enabled=1
```

Then type

```
yum   install   mongodb-org
```

and wait for a long time, in between you may be asked about size , with a question Is it OK[y/n]:
type y and press enter and things will be downloaded,

In the end you will get the messge as Complete!

Now create folder named as data on / by typing

```
mkdir  /data
```

then in data create a folder named as db by typing

```
mkdir  /data/db
```

now type mongod
and the server should start
press ctrl c
Now open /etc/

done done
Now let us learn how to manage multiple terminals. We are in text mode

right now we are logged in as root user. Now lets say that we want to keep logged in as root and switch to another instance of the terminal and login as student and keep switching as per requirement
It is very simple
press CTRL + ALT + F2
The linux login prompt will appear, login as student
Now
press CTRL + ALT + F1
and to switch back to the second console
press CTRL + ALT + F2

use can use F3 or F4 or F5 or F6 (1-6) options. It is sufficient of all our future need.

Now let us learn to start and stop server.

Now let us run the mongo db server from our student login on 1st instance
Exit from root
login as student
first of all we will create the following structure in student logins home folder mongodb/data/db
for that type
mkdir mongodb
mkdir mongodb/data
mkdir mongodb/data/db

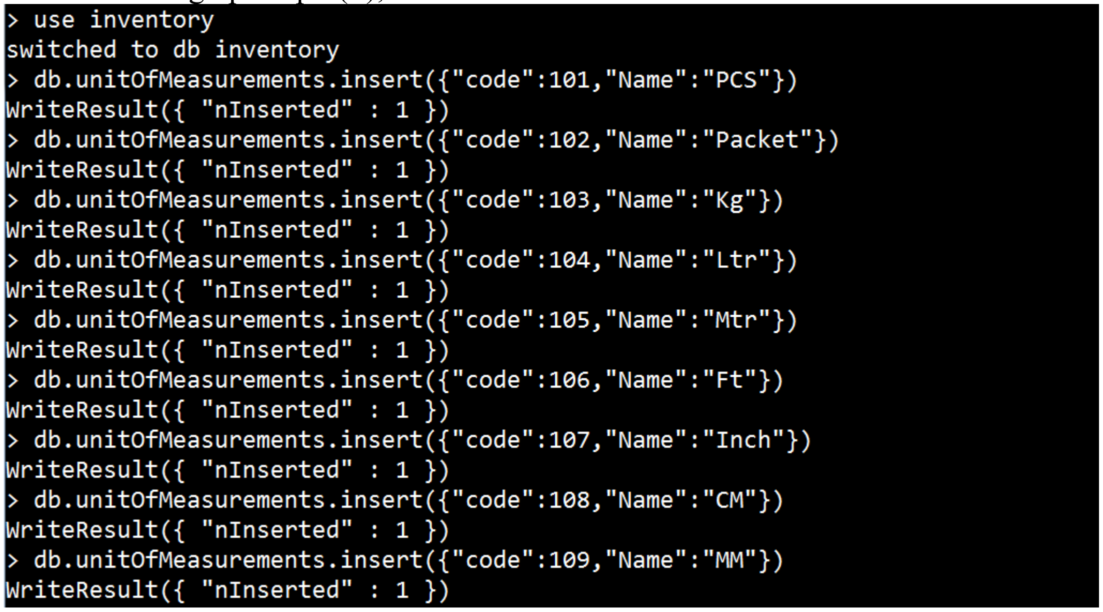
now to start server and use the create folder structure as the data path type
mongod   --dbpath   mongodb/data/db
the server will start listening on port 27017

Now we will switch to 2nd instance of console. For that press CTRL + ALT + F2
login as student
How to stop the server
mongod   –dbpath   mongodb/data/db --shutdown

Now you can switch to the 1st console and you will see that the mongo db server has been shut down.

Now let us learn to perform CRUD operations using the mongo db client.
First of all start the server as taught earlier
mongod   --dbpath   mongodb/data/db

Now switch to another console instance, and type
mongo
you should see the mongo prompt. (>), then see what I did to insert unitOfMeasurements



```
> use inventory
switched to db inventory
> db.unitOfMeasurements.insert({"code":101,"Name":"PCS"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":102,"Name":"Packet"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":103,"Name":"Kg"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":104,"Name":"Ltr"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":105,"Name":"Mtr"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":106,"Name":"Ft"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":107,"Name":"Inch"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":108,"Name":"CM"})
WriteResult({ "nInserted" : 1 })
> db.unitOfMeasurements.insert({"code":109,"Name":"MM"})
WriteResult({ "nInserted" : 1 })
```

to insert :   db.unitOfMeasurements.insert({“code”:101,”name”:”MM”})

To fetch added unitOfMeasurements type

```
db.unitOfMeasurements.find()
```

or

```
db.unitOfMeasurements.find({“code”:103})
```

to insert multiple you can do

```
db.unitOfMeasurements.insert({              },{               },{                },
{               },{              },
{               },{               })
```

to add relational operators in query do the following

```
db.unitOfMeasurements.find({“code”: { $gt : 102}})
```

other query operators are

$gte $lt $lte $eq $ne $in $nin

Logical operators

```
db.unitOfMeasurements.find({$or: [{"code":103},{"code":104},{"code":{$gt:107}}]})
```

other logical operator are $and $not $nor

type

quit() to get out of mongo client

Now again start mongo client and type

```
use inventory
db.unitOfMeasurements.remove({})
```

and quit()

we just emptied the whole unitOfMeasurements collection.

Now in /home/student create a folder named as mongodb-json-files

now move into that folder and create a file named as unit_of_measurement.json with following contents.

```
[
{  "code" : 101, "name" : "PCS" },
{  "code" : 102, "name" : "PACKET" },
{  "code" : 103, "name" : "KG" },
{  "code" : 104, "name" : "LTR" },
{  "code" : 105, "name" : "MTR" },
{  "code" : 106, "name" : "FT" },
{  "code" : 107, "name" : "INCH" },
{  "code" : 108, "name" : "CM" },
{  "code" : 109, "name" : "MM" }
]
```

Now on the linux prompt while staying in the mongodb-json-files folder type the following
mongoimport --db inventory --collection unitOfMeasurements --type json --file unit_of_measurement.json –jsonArray

Note : Server is running and we used a utility named as mongoimport to import json data from our file.

Now start mongo client and verify the existence of the record using use inventory followed by db.unitOfMeasurements.find()

Note down the values of _id fields along with the name of the Unit Of Measurement
Values at my end are as follows.

```
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d6"), "code" : 101, "name" : "PCS" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d7"), "code" : 103, "name" : "KG" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d8"), "code" : 104, "name" : "LTR" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d9"), "code" : 105, "name" : "MTR" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45da"), "code" : 106, "name" : "FT" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45db"), "code" : 107, "name" : "INCH" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45dc"), "code" : 108, "name" : "CM" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45dd"), "code" : 109, "name" : "MM" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45de"), "code" : 102, "name" : "PACKET" }
```

Now create another file by the name of item.json in the same folder with following json array

category.json

```
[
{ "code" : 1001,
"category" : "Thread"
},
{ "code" : 1002,
"category" : "Trunk"
},
{ "code" : 1003,
"category" : "Umbrella"
},
{ "code" : 1004,
"category" : "Cloth"
}
]
```

Now use mongoimport to import the documents in a collection named as categories using the following

```
mongoimport --db   inventory   –collection categories   --type  json   --file   category.json   –jsonArray
```

Use mongo client a verify that the documents exist.
The values at my end are as follows

```
{ "_id" : ObjectId("57ea7520a5dcadad46ef45e3"), "code" : 1001, "category" : "Thread" }
{ "_id" : ObjectId("57ea7520a5dcadad46ef45e4"), "code" : 1003, "category" : "Umbrella" }
{ "_id" : ObjectId("57ea7520a5dcadad46ef45e5"), "code" : 1004, "category" : "Cloth" }
{ "_id" : ObjectId("57ea7520a5dcadad46ef45e6"), "code" : 1002, "category" : "Trunk" }
```

Now create brand.json

brand.json

```
[
{ "code" : 1,
"category" : "Picasso"
},
{ "code" : 2,
"category" : "Flora"
},
{ "code" : 3,
"category" : "Novelty"
},
{ "code" : 1004,
"category" : "Raymonds"
},
{ "code" : 1005,
"category" : "Carbon"
},
{ "code" : 1006,
"category" : "Sharp"
}
]
```

Now import the documents from brand.json using mongoimport and verify it, values at my end are as follows

```
{ "_id" : ObjectId("57ea824da5dcadad46ef45ed"), "code" : 1, "category" : "Picasso" }
{ "_id" : ObjectId("57ea824da5dcadad46ef45ee"), "code" : 2, "category" : "Flora" }
{ "_id" : ObjectId("57ea824da5dcadad46ef45ef"), "code" : 3, "category" : "Novelty" }
{ "_id" : ObjectId("57ea824da5dcadad46ef45f0"), "code" : 1004, "category" : "Raymonds" }
{ "_id" : ObjectId("57ea824da5dcadad46ef45f1"), "code" : 1005, "category" : "Carbon" }
{ "_id" : ObjectId("57ea824da5dcadad46ef45f2"), "code" : 1006, "category" : "Sharp" }
```

Now create item.json as per the criteria discussed in the classroom session.

Now let us write java code to insert a UnitOfMeasurement in Mongo DB database
login as root and move to /ourlib
download the mongodb.jar file using **wget "http://tm-certificates.com/mongodb.jar"**

Now move into the javaeg folder and create mongo1.java as follows

```
mongo1.java
import com.mongodb.*;
class mongo1
{
public static void main(String data[])
{
try
{
int code=Integer.parseInt(data[0]);
String name=data[1];
Mongo mongoClient=new Mongo("localhost",27017);
DB db=mongoClient.getDB("inventory");
DBCollection unitOfMeasurements=db.getCollection("unitOfMeasurements");
BasicDBObject basicObject=new BasicDBObject();
basicObject.append("code",code);
basicObject.append("name",name);
unitOfMeasurements.insert(basicObject);
System.out.println("Inserted document");
}catch(Exception e)
{
System.out.println(e);
}
}
}
```

---

compile using

```
javac  -classpath   /ourlib/mongodb.jar:.    mongo1.java
```

for execution

```
java    -classpath    /ourlib/mongodb.jar:.    mongo1   110    ML
```

Verify using mongo client, following is the output on my side.

```
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d6"), "code" : 101, "name" : "PCS" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d7"), "code" : 103, "name" : "KG" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d8"), "code" : 104, "name" : "LTR" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45d9"), "code" : 105, "name" : "MTR" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45da"), "code" : 106, "name" : "FT" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45db"), "code" : 107, "name" : "INCH" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45dc"), "code" : 108, "name" : "CM" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45dd"), "code" : 109, "name" : "MM" }
{ "_id" : ObjectId("57ea6474a5dcadad46ef45de"), "code" : 102, "name" : "PACKET" }
{ "_id" : ObjectId("57ebc58ee4b0f4700b4a1b59"), "code" : 110, "name" : "ML" }
```

Now let us create mongo2.java to extract records from unitOfMeasurements collection

```
import com.mongodb.*;
class mongo2
{
public static void main(String data[])
{
try
{
Mongo mongoClient=new Mongo("localhost",27017);
DB db=mongoClient.getDB("inventory");
DBCollection unitOfMeasurements=db.getCollection("unitOfMeasurements");
DBCursor cursor=unitOfMeasurements.find();
DBObject dbObject;
while(cursor.hasNext())
{
dbObject=cursor.next();
System.out.println(dbObject);
}
}catch(Exception e)
{
System.out.println(e);
}
}
}
```

Compile and run as we did for mongo1.java

Maven

Login as root
move to /root/Downloads folder
download the zip file using the following

```
wget "http://mirror.fibergrid.in/apache/maven/maven-3/3.3.9/binaries/apache-maven-3.3.9-bin.tar.gz"
```

After downloading, unzip it using

```
tar    xvf    apache-maven-3.3.9-bin.tar.gz
```

Now a folder by the name of apache-maven-3.3.9 must have got created, let us move that folder to /usr/local and we will name it as apache-maven, for that do the following

right now my current foder is Downloads, so I typed

```
mv    apache-maven-3.3.9     /usr/local/apache-maven
```

Now we will add required environment variables to .bashrc file of student login, we will not do that for

root as we will be doing all the necessary practicals as student
for that edit the /home/student/,bashrc file and add the following before the line export path

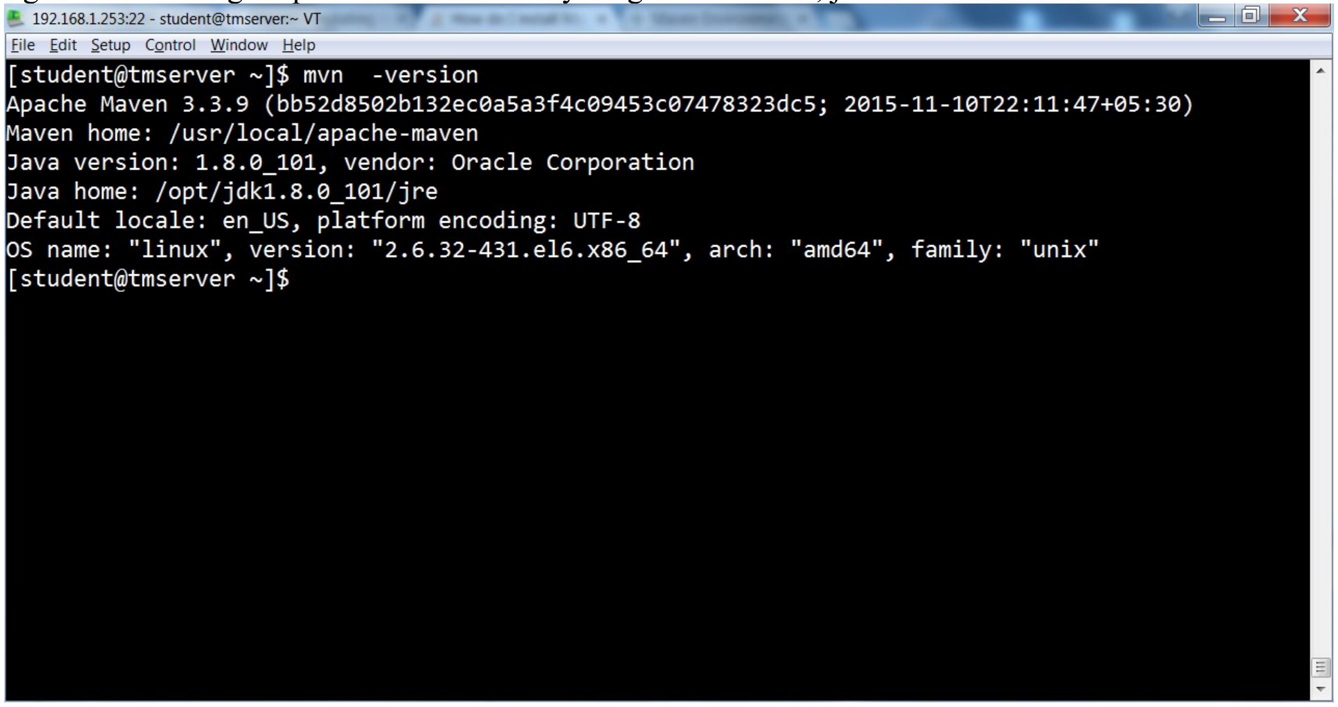
Finally my .bashrc file under /home/student/ folder looks as follows

```
# .bashrc

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi

# User specific aliases and functions
export JAVA_HOME=/opt/jdk1.8.0_101
export JRE_HOME=/opt/jdk1.8.0_101/jre
PATH=$PATH:/opt/jdk1.8.0_101/bin:/opt/jdk1.8.0_101/jre/bin
export ORACLE_HOME=/u01/app/oracle/product/11.2.0/xe
export ORACLE_BASE=/u01/app/oracle
export ORACLE_SID=XE
PATH=$PATH:/u01/app/oracle/product/11.2.0/xe/bin
export M2_HOME=/usr/local/apache-maven
export M2=$M2_HOME/bin
PATH=$M2:$PATH
```

Now exit from root login and login as root ant type

```
mvn  -version
```

I got the following output which states everything about maven, java and linux


192.168.1.253:22 - student@tmserver:~ VT
File Edit Setup Control Window Help

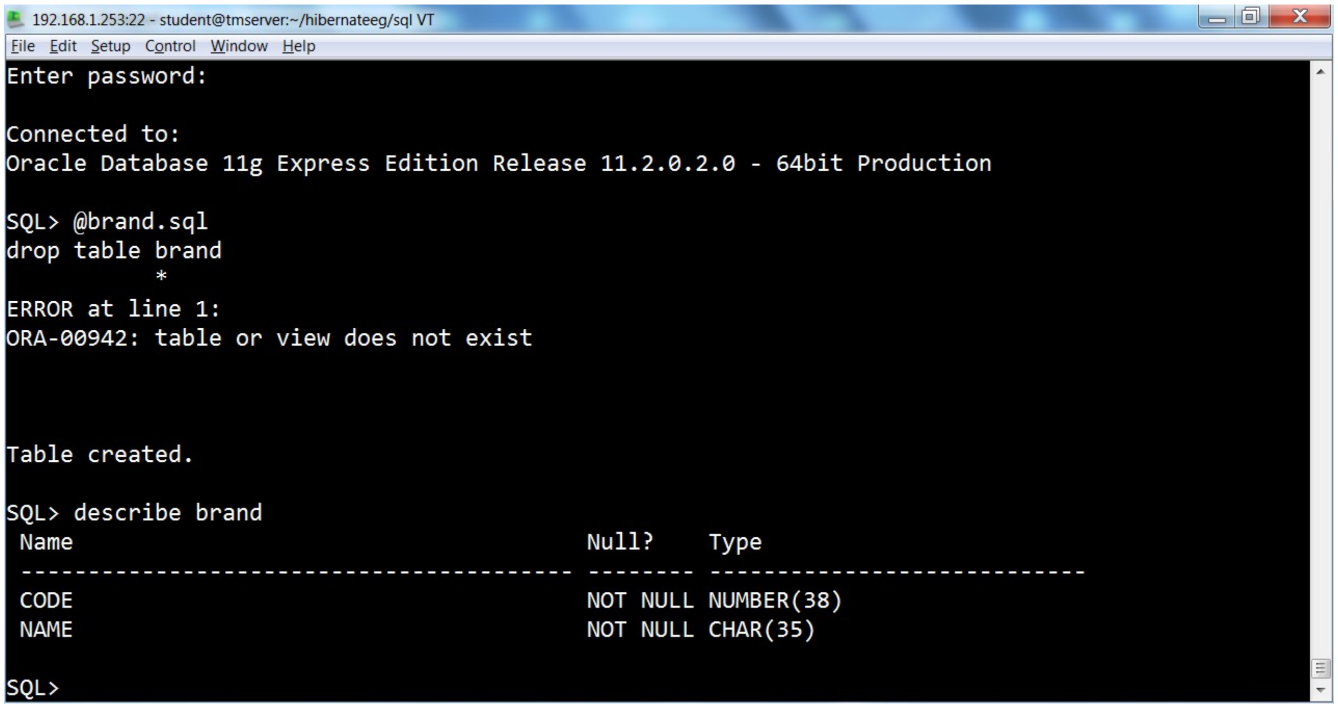
```
[student@tmserver ~]$ mvn  -version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T22:11:47+05:30)
Maven home: /usr/local/apache-maven
Java version: 1.8.0_101, vendor: Oracle Corporation
Java home: /opt/jdk1.8.0_101/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "2.6.32-431.el6.x86_64", arch: "amd64", family: "unix"
[student@tmserver ~]$
```

Now let us create our first hibernate example. First of all create folder named as hibernateeg in the /home/student/ folder
Now move into the hibernateeg folder
create a folder named as sql
in sql folder create a file named as brand.sql with following content

brand.sql

```
drop table brand;
create table brand
(
code int primary key,
name char(35) not null unique
);
```

Now while staying in the sql folder, start sqlplus using
sqlplus student
Provide password and then on the sql prompt type
@brand.sql
The sql statements from the file will be executed, since initially there is no table by the name of brand, the drop table brand won't work, don't worry, the next statement to create the brand table will still work.
You can see by typing
describe brand
My UI is as follows


192.168.1.253:22 - student@tmserver:~/hibernateeg/sql VT
File Edit Setup Control Window Help

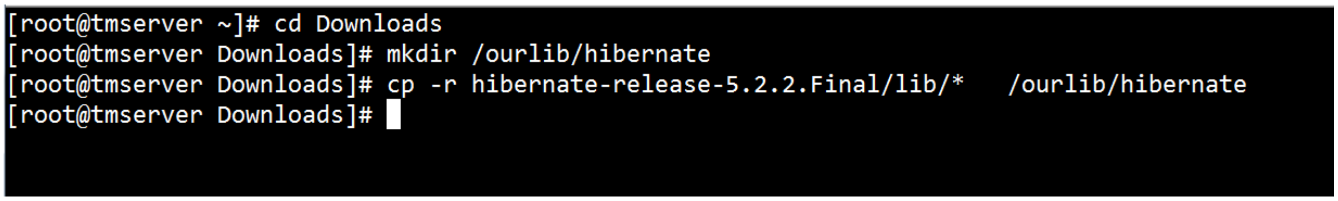
```
Enter password:

Connected to:
Oracle Database 11g Express Edition Release 11.2.0.2.0 - 64bit Production

SQL> @brand.sql
drop table brand
           *
ERROR at line 1:
ORA-00942: table or view does not exist



Table created.

SQL> describe brand
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 CODE                                      NOT NULL NUMBER(38)
 NAME                                      NOT NULL CHAR(35)

SQL>
```

Now quit from sqlplus

Now logout from student account and login as root
Start GUI mode using startx, use opera to download Hibernate latest release, it will be a zip file.
When asked, save it to Downloads folder. After the file gets downloaded, right click and select extract here. Finally a folder will be created by the name of
hibernate-release-5.2.2.Final
Now swith back to the text mode by logging out of the GUI mode.
Now create a folder named as hibernate in /ourlib folder by typing
mkdir   /ourlib/hibernate
now while sitting in Downloads folder, type the following

```
cp   -r    hibernate-release-5.2.2.Final/lib/*   /ourlib/hibernate
```

refer the following UI, if you have a problem



```
[root@tmserver ~]# cd Downloads
[root@tmserver Downloads]# mkdir /ourlib/hibernate
[root@tmserver Downloads]# cp -r hibernate-release-5.2.2.Final/lib/*   /ourlib/hibernate
[root@tmserver Downloads]# 
```

Now let us create our first example to perform CRUD Operations on the brand table.

Exit from root and login as student.
Move into the hibernateeg folder, in the hibernateeg folder create folder named as example1
move into the example1 folder.

```
package com.thinking.machines.hibernate.utility;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtility
{
private static final SessionFactory sessionFactory = buildSessionFactory();
private static SessionFactory buildSessionFactory()
{
try
{
return new Configuration().configure().buildSessionFactory();
}catch(Throwable throwable)
{
System.err.println("Initial SessionFactory creation failed." + throwable);
throw new ExceptionInInitializerError(throwable);
}
}
public static SessionFactory getSessionFactory()
{
return sessionFactory;
}
public static void shutdown()
{
getSessionFactory().close();
}
}
```

```
package com.thinking.machines.inventory;
public class Brand implements java.io.Serializable,Comparable<Brand>
{
private int code;
private String name;
public Brand()
{
code=0;
name="";
}
public void setCode(int code)
{
this.code=code;
}
public int getCode()
{
return this.code;
}
public void setName(String name)
{
this.name=name;
}
public String getName()
{
return this.name;
}
public boolean equals(Object object)
{
if(!(object instanceof Brand)) return false;
return this.code==((Brand)object).code;
}
public int compareTo(Brand brand)
{
return this.name.compareToIgnoreCase(brand.name);
}
public int hashCode()
{
return this.code;
}
}
```

Create the following files in testcases folder

brand.hbm.xml

```
<?xml version="1.0"?>
<!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.thinking.machines.inventory.Brand" table="brand">
<id name="code" type="int">
<column name="code" precision="10" scale="0" />
<generator class="assigned" />
</id>
<property name="name" type="string">
<column name="name" length="35" not-null="true" unique="true"/>
</property>
</class>
</hibernate-mapping>
```

---

hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
<property name="hibernate.connection.username">student</property>
<property name="hibernate.connection.password">student</property>
<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<property name="show_sql">true</property>
<mapping resource="brand.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

```
import java.util.Date;
import org.hibernate.Session;
import com.thinking.machines.hibernate.utility.*;
import com.thinking.machines.inventory.*;
public class AddBrand
{
public static void main(String[] data)
{
Session session=HibernateUtility.getSessionFactory().openSession();
try
{
int code=Integer.parseInt(data[0]);
String name=data[1];
session.beginTransaction();
Brand brand=new Brand();
brand.setCode(code);
brand.setName(name);
session.save(brand);
session.getTransaction().commit();
System.out.println("***************");
System.out.println("Brand added");
System.out.println("***************");
}catch(Throwable t)
{
System.out.println("*************************");
System.out.println("Problem : "+t);
System.out.println("*************************");
}
HibernateUtility.shutdown();
}
}
```

```
import java.util.Date;
import org.hibernate.Session;
import com.thinking.machines.hibernate.utility.*;
import com.thinking.machines.inventory.*;
public class UpdateBrand
{
public static void main(String[] data)
{
Session session=HibernateUtility.getSessionFactory().openSession();
try
{
int code=Integer.parseInt(data[0]);
String name=data[1];
session.beginTransaction();
Object object=session.load(Brand.class,code);
if(object!=null)
{
Brand brand=(Brand)object;
System.out.println("*********** Existing Data *************");
System.out.println(brand.getCode()+","+brand.getName());
System.out.println("*********** Updating data  *************");
brand.setName(name);
session.update(brand);
System.out.println("***************");
System.out.println("Brand updated");
System.out.println("***************");
}
else
{
System.out.println("Unable to load brand with code : "+code);
}
session.getTransaction().commit();
}catch(Throwable t)
{
System.out.println("*************************");
System.out.println("Problem : "+t);
System.out.println("*************************");
}
HibernateUtility.shutdown();
}
}
```

```
import java.util.Date;
import org.hibernate.Session;
import com.thinking.machines.hibernate.utility.*;
import com.thinking.machines.inventory.*;
public class DeleteBrand
{
public static void main(String[] data)
{
Session session=HibernateUtility.getSessionFactory().openSession();
try
{
int code=Integer.parseInt(data[0]);
session.beginTransaction();
Object object=session.load(Brand.class,code);
if(object!=null)
{
Brand brand=(Brand)object;
System.out.println("*********** Existing Data *************");
System.out.println(brand.getCode()+","+brand.getName());
System.out.println("*********** deleting  data  *************");
session.delete(brand);
System.out.println("****************");
System.out.println("Brand deleted");
System.out.println("****************");
}
else
{
System.out.println("Unable to load brand with code : "+code);
}
session.getTransaction().commit();
}catch(Throwable t)
{
System.out.println("*************************");
System.out.println("Problem : "+t);
System.out.println("**************************");
}
HibernateUtility.shutdown();
}
}
```

```
import java.util.Date;
import org.hibernate.Session;
import com.thinking.machines.hibernate.utility.*;
import java.util.*;
import com.thinking.machines.inventory.*;
public class GetAllBrands
{
public static void main(String[] data)
{
Session session=HibernateUtility.getSessionFactory().openSession();
try
{
session.beginTransaction();
Brand brand;
List brands=session.createQuery("from Brand").list();
for(int x=0;x<brands.size();x++)
{
brand=(Brand)brands.get(x);
System.out.println(brand.getCode()+","+brand.getName());
}
session.getTransaction().commit();
System.out.println("****************");
System.out.println("Brand listing done");
System.out.println("****************");
}catch(Throwable t)
{
System.out.println("**************************");
System.out.println("Problem : "+t);
System.out.println("**************************");
}
HibernateUtility.shutdown();
}
}
```

```
import java.util.Date;
import org.hibernate.Session;
import com.thinking.machines.hibernate.utility.*;
import com.thinking.machines.inventory.*;
public class GetBrand
{
public static void main(String[] data)
{
Session session=HibernateUtility.getSessionFactory().openSession();
try
{
int code=Integer.parseInt(data[0]);
session.beginTransaction();
Object object=session.load(Brand.class,code);
if(object!=null)
{
Brand brand=(Brand)object;
System.out.println("*********** Existing Data ************");
System.out.println(brand.getCode()+","+brand.getName());
}
else
{
System.out.println("Unable to load brand with code : "+code);
}
session.getTransaction().commit();
}catch(Throwable t)
{
System.out.println("**************************");
System.out.println("Problem : "+t);
System.out.println("**************************");
}
HibernateUtility.shutdown();
}
}
```

```
import java.util.Date;
import org.hibernate.Session;
import com.thinking.machines.hibernate.utility.*;
import com.thinking.machines.inventory.*;
import org.hibernate.Query;
public class DeleteAllBrands
{
public static void main(String[] data)
{
Session session=HibernateUtility.getSessionFactory().openSession();
try
{
session.beginTransaction();
System.out.println("*********** deleting  all data  ************");
Query hql=session.createQuery("delete from Brand");
hql.executeUpdate();
System.out.println("***************");
System.out.println("All Brands deleted");
System.out.println("***************");
session.getTransaction().commit();
}catch(Throwable t)
{
System.out.println("*************************");
System.out.println("Problem : "+t);
System.out.println("*************************");
}
HibernateUtility.shutdown();
}
}
```

---

in example1 folder create the following shell script file : cmpapp.sh

cmpapp.sh

```
cd classes
rm -r com
cd ..
cd src
javac -classpath  /ourlib/hibernate/required/*:.  -d  ../classes
com/thinking/machines/hibernate/utility/*.java
javac -classpath  /ourlib/hibernate/required/*:.  -d  ../classes  com/thinking/machines/inventory/*.java
cd ..
cd testcases
javac -classpath  /ourlib/hibernate/required/*:../classes:. *.java
cd ..
```

To compile (while staying in example1 folder type sh cmpapp.sh)

Now move to testcases folder and for execution type

```
java    -classpath   /ourlib/oracle.jar:/ourlib/hibernate/required/*:../classes:.  AddBrand arg1  arg2
```

Note arg1 means code and arg2 means name

Run the rest of the test cases and after every testcase execution login to oracle and check the records in brand table.

For every testcase , check what needs to be passed as arguments

---

Hibernate - Example 2 (This time we will repeat the example 1, but will use maven)
Login as student
First of all you need to be connected to internet.

Now move into the /home/student/hibernateeg folder and type the following. (Note : before every dash I have given a space)

```
mvn   archetype:generate    -DgroupId=com.thinking.machines.inventory    -DartifactId=example2
-DarchetypeArtifactId=maven-archetype-quickstart   -DinteractiveMode=false
```

If everything works well, then a folder by the name of example2 will get created. That folder will contain the pom.xml file and src folder. Move into the src folder and remove the test folder by typing

```
rm  -r   test
```

now move back to the example2 folder.
Now edit the pom.xml file and make it as follows

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.thinking.machines.inventory</groupId>
  <artifactId>HibernateExample2</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>HibernateExample2</name>
  <url>http://maven.apache.org</url>

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

<repositories>
<repository>
<id>JBoss repository</id>
<url>http://repository.jboss.org/nexus/content/groups/public/</url>
</repository>
</repositories>


  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

   <dependency>
     <groupId>org.hibernate</groupId>
     <artifactId>hibernate-core</artifactId>
     <version>5.2.2.Final</version>
   </dependency>

   <dependency>
    <groupId>com.oracle</groupId>
    <artifactId>ojdbc6</artifactId>
    <version>11.2.0</version>
   </dependency>

  </dependencies>
</project>
```

Now in example2 create a folder named as lib using mkdir lib
Then copy the oracle.jar from /ourlib to lib type staying in example2 folder and typing

```
cp   /ourlib/oracle.jar   lib
```

Now we need to install oracle.jar in maven repository. For that type

```
mvn    install:install-file    -Dfile=lib/oracle.jar  -DgroupId=com.oracle   -DartifactId=ojdbc6
-Dversion=11.2.0 -Dpackaging=jar
```

Now move to src/main/java/com/thinking/machines folder and create hibernate folder and in hibernate create utility folder.

Create the following java file (HibernateUtility) in the utility folder.

```
package com.thinking.machines.hibernate.utility;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
public class HibernateUtility
{
private static final SessionFactory sessionFactory = buildSessionFactory();
private static SessionFactory buildSessionFactory()
{
try
{
return new Configuration().configure().buildSessionFactory();
}catch(Throwable throwable)
{
System.err.println("Initial SessionFactory creation failed." + throwable);
throw new ExceptionInInitializerError(throwable);
}
}
public static SessionFactory getSessionFactory()
{
return sessionFactory;
}
public static void shutdown()
{
getSessionFactory().close();
}
}
```

Now create the following java files (see class names) in inventory folder

```
package com.thinking.machines.inventory;
import java.util.*;
public class BusinessLayerException extends Exception
{
private HashMap<String,String> exceptions=new HashMap<String,String>();
public BusinessLayerException()
{
super("");
}
public BusinessLayerException(String message)
{
super(message);
}
public void put(String key,String value)
{
exceptions.put(key,value);
}
public String get(String key)
{
return exceptions.get(key);
}
public boolean contains(String key)
{
return exceptions.get(key)!=null;
}
public int getSize()
{
return exceptions.size();
}
public boolean hasGenericException()
{
return getMessage()!=null && getMessage().trim().length()>0;
}
public boolean hasPropertyException()
{
return exceptions.size()>0;
}
}
```

```
package com.thinking.machines.inventory;
public class BrandException extends BusinessLayerException
{
public BrandException()
{
}
public BrandException(String message)
{
super(message);
}
}
```

```
package com.thinking.machines.inventory;
public class Brand implements java.io.Serializable,Comparable<Brand>
{
private int code;
private String name;
public Brand()
{
code=0;
name="";
}
public void setCode(int code)
{
this.code=code;
}
public int getCode()
{
return this.code;
}
public void setName(String name)
{
this.name=name;
}
public String getName()
{
return this.name;
}
public boolean equals(Object object)
{
if(!(object instanceof Brand)) return false;
return this.code==((Brand)object).code;
}
public int compareTo(Brand brand)
{
return this.name.compareToIgnoreCase(brand.name);
}
public int hashCode()
{
return this.code;
}
}
```

```
package com.thinking.machines.inventory;
import com.thinking.machines.hibernate.utility.*;
import org.hibernate.*;
import java.util.*;
public class BrandManager
{
public void addBrand(Brand brand) throws BrandException
{
if(brand==null) throw new BrandException("Brand not provided to add.");
BrandException brandException=new BrandException();
if(brand.getCode()<=0) brandException.put("code","Brand code missing.");
if(brand.getName()==null || brand.getName().trim().length()==0) brandException.put("name","Brand
name missing.");
if(brandException.getSize()>0) throw brandException;
Session session=null;
try
{
session=HibernateUtility.getSessionFactory().openSession();
session.beginTransaction();
session.save(brand);
session.getTransaction().commit();
}catch(Throwable t)
{
throw new BrandException(t.getMessage());
}
finally
{
if(session!=null)
{
session.close();
}
}
}
public List getAllBrands() throws BrandException
{
List brands=null;
Session session=null;
try
{
session=HibernateUtility.getSessionFactory().openSession();
session.beginTransaction();
brands=session.createQuery("from Brand").list();
session.getTransaction().commit();
}catch(Throwable t)
{

}
finally
{
if(session!=null)
{
session.close();
}
}
if(brands==null)
{
throw new BrandException("No brands.");
}
return brands;
}
}
```

From inventory folder delete the App.java file (this was created by MAVEN)

Now move to /home/student/hibernateeg/example2 folder and type

mvn clean compile

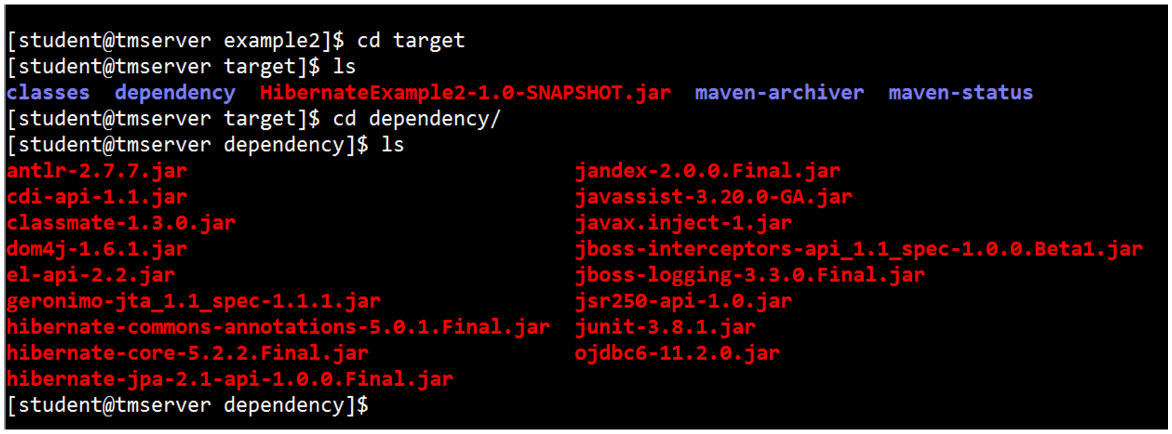
The build should be successful. Now to create the jar file type

mvn package

Now we will make maven install dependencies in target folder, for that type the following

mvn install dependency:copy-dependencies

Now move into the target folder and you should see the jar file, I am providing my directory contents as follows



```
[student@tmserver example2]$ cd target
[student@tmserver target]$ ls
classes  dependency  HibernateExample2-1.0-SNAPSHOT.jar  maven-archiver  maven-status
[student@tmserver target]$ cd dependency/
[student@tmserver dependency]$ ls
antlr-2.7.7.jar                               jandex-2.0.0.Final.jar
cdi-api-1.1.jar                               javassist-3.20.0-GA.jar
classmate-1.3.0.jar                           javax.inject-1.jar
dom4j-1.6.1.jar                               jboss-interceptors-api_1.1_spec-1.0.0.Beta1.jar
el-api-2.2.jar                                jboss-logging-3.3.0.Final.jar
geronimo-jta_1.1_spec-1.1.1.jar               jsr250-api-1.0.jar
hibernate-commons-annotations-5.0.1.Final.jar junit-3.8.1.jar
hibernate-core-5.2.2.Final.jar                ojdbc6-11.2.0.jar
hibernate-jpa-2.1-api-1.0.0.Final.jar
[student@tmserver dependency]$
```

Note (Henceforth if you require to recompile, then type mvn compile only. If you write mvn clean compile, then remember to again install dependencies as done above)

Now move back to the example2 folder and create a folder named as testcases in the example2 folder move into the testcases folder and create the following java files (See the class names)

```
import com.thinking.machines.inventory.*;
public class AddBrandTestCase
{
public static void main(String data[])
{
int code=Integer.parseInt(data[0]);
String name=data[1];
Brand brand=new Brand();
brand.setCode(code);
brand.setName(name);
try
{
new BrandManager().addBrand(brand);
System.out.println("Brand added.");
}catch(BrandException brandException)
{
System.out.println(brandException);
}
}
}
```

```
import com.thinking.machines.inventory.*;
import java.util.*;
public class GetAllBrandsTestCase
{
public static void main(String data[])
{
Brand brand;
try
{
List brands=new BrandManager().getAllBrands();
for(int x=0;x<brands.size();x++)
{
brand=(Brand)brands.get(x);
System.out.printf("Brand : Code %d, Name %s\n",brand.getCode(),brand.getName());
}
}catch(BrandException brandException)
{
System.out.println(brandException);
}
}
}
```

Now while staying in the testcases folder, type the following to compile the testcases

```
javac   -classpath   ../target/*:../target/dependency/*:.   *.java
```

Now move to the example2 folder and create the following xml files

brand.hbm.xml

```
<?xml version="1.0"?>
<!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.thinking.machines.inventory.Brand" table="brand">
<id name="code" type="int">
<column name="code" precision="10" scale="0" />
<generator class="assigned" />
</id>
<property name="name" type="string">
<column name="name" length="35" not-null="true" unique="true"/>
</property>
</class>
</hibernate-mapping>
```

hibernate.cfg.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>
<property name="hibernate.connection.driver_class">oracle.jdbc.driver.OracleDriver</property>
<property name="hibernate.connection.url">jdbc:oracle:thin:@localhost:1521:XE</property>
<property name="hibernate.connection.username">student</property>
<property name="hibernate.connection.password">student</property>
<property name="hibernate.dialect">org.hibernate.dialect.Oracle10gDialect</property>
<property name="show_sql">false</property>
<mapping resource="brand.hbm.xml"></mapping>
</session-factory>
</hibernate-configuration>
```

Now while staying in the example2 folder type the following to run the test cases

```
java   -classpath   target/*:target/dependency/*:testcases:.    AddBrandTestCase 106 SomeBrand
```

and

```
java   -classpath   target/*:target/dependency/*:testcases:.    AddBrandTestCase 106 SomeBrand
```

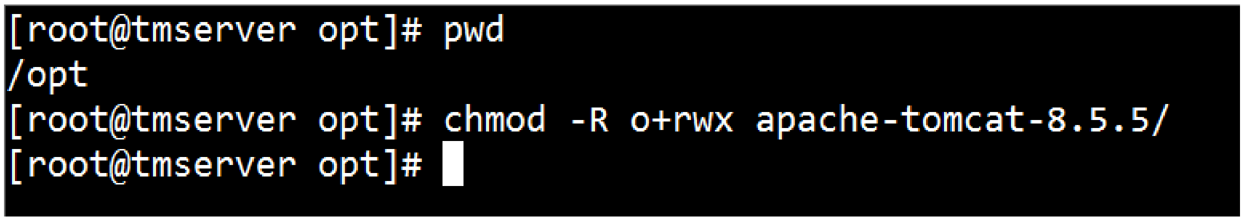
Now let us install Tomcat 8
First of all login as root user.
Now move to /opt folder
Then type the following to download the tomcat gz file
wget "http://mirror.fibergrid.in/apache/tomcat/tomcat-8/v8.5.5/bin/apache-tomcat-8.5.5.tar.gz"

Now unzip the gz file using the following command

tar -xvf apache-tomcat-8.5.5.tar.gz

Now in the opt folder a folder by the name of apache-tomcat-8.5.5 must have got created.
Now we will have to change the default listening port number from 8080 to something else.
We will change it to 5050. We have to do this, because we have installed oracle and it has grabbed port number 8080 for its web interface. Hence tomcat cannot start on 8080.

edit the following file using vi editor
/opt/apache-tomcat-8.5.5/conf/server.xml
and change 8080 to 5050, you will find it 3 times, don't change anything else.
Now move to the /opt/apache-tomcat-8.5.5/bin folder and
type
./startup.sh
Tomcat will start
Now move to gui mode and start browser and in the address bar type the following URL
http://localhost:5050 and you will see the Tomcat's default page.
To stop the server while staying in the bin folder type
./shutdown.sh
Now we will have to give the execution rights of the startup and shutdown scripts to student login.
First of all move to /opt folder and then type the following
chmod -R o+rwx apache-tomcat-8.5.5



```
[root@tmserver opt]# pwd
/opt
[root@tmserver opt]# chmod -R o+rwx apache-tomcat-8.5.5/
[root@tmserver opt]# 
```

Now logout from root and login as student
Verify that you are able to start tomcat as student, for that after login as student type
/opt/apache-tomcat-8.5.5/bin/startup.sh
then , if you see the message that server has started, open browser and type http://localhost:5050 and you should see the tomcat's default page
Then to shutdown type
/opt/apache-tomcat-8.5.5/bin/shutdown.sh

Now let us develop our first application. Our objective is as follows.

We have already created BusinessLayer code for Brand using ORM (Hibernate) to handle the data layer part to persist data in ORACLE 11g.

Prerequisite
Database : (We have ORACLE 11g with XE instance with brand table
(We already have it ready as example2 of hibernate)
We have our necessary business layer jar file in
/home/student/hibernateeg/example2/target folder
and we have the required dependencies in
/home/student/hibernateeg/example2/target/dependency folder

Now we will create a web application with context name ws1.com in folder wsexmaple1
Then we will use jersey framework to expose our business layer methods as webservice.

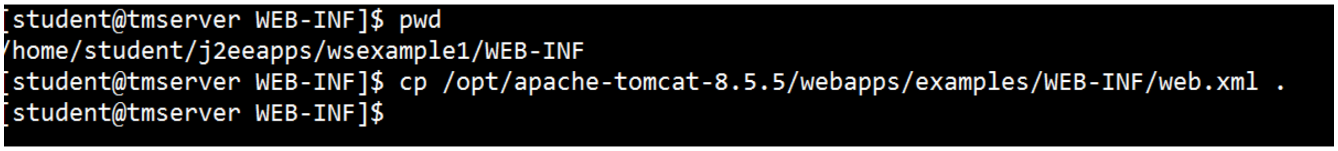
First of all let us create a web application folder structure
Note : Earlier we use to move into the tomcat's webapps folder and create structure.
This time we will create the folder somewhere else and configure the tomcat to consider the folder as a webapplication folder.
Move to /home/student/ folder.
In it create a folder named as j2eeapps

In the j2eeapps folder create the following structure

wsexample1 - WEB-INF - classes
wsexample1 - WEB-INF – lib

Now we need web.xml file in WEB-INF, instead of creating it from scratch we will copy it from the examples application of tomcat, for that, first move to the WEB-INF folder of our site.

Then type
cp /opt/apache-tomcat-8.5.5/webapps/examples/WEB-INF/web.xml .


```
[student@tmserver WEB-INF]$ pwd
/home/student/j2eeapps/wsexample1/WEB-INF
[student@tmserver WEB-INF]$ cp /opt/apache-tomcat-8.5.5/webapps/examples/WEB-INF/web.xml .
[student@tmserver WEB-INF]$
```

Now edit the web.xml file using vi editor and remove the unwanted lines between the
<web-app>
and
</web-app>
tags.

Finally my web.xml is as follows

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">

<description>
My wsexample1 depolyment descriptor
</description>
    <display-name>wsexample1 application</display-name>
</web-app>
```

Now move to wsexample1 folder and create the following test.html file

test.html

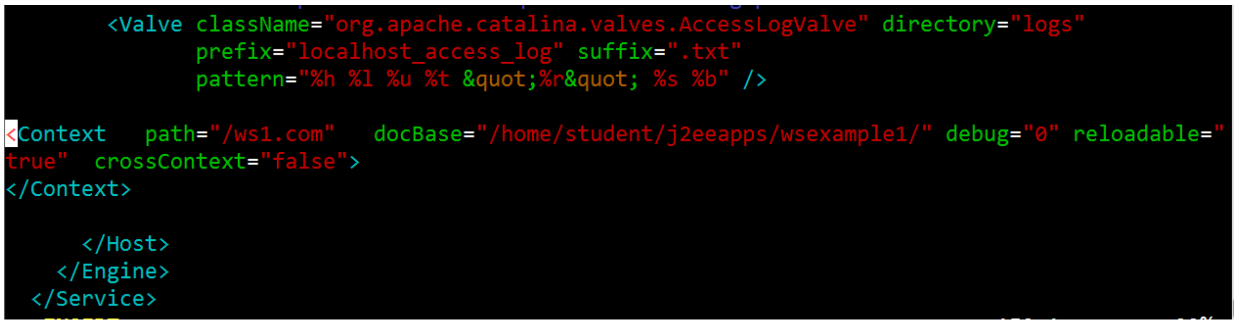
```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Title of the document</title>
</head>
<body>
<h1>Test Page of ws1.com</h1>
</body>
</html>
```

Now let us configure tomcat to consider our wsexample1 folder as webapplication folder even though it is not in tomcat's webapps folder.

To do that, move to /opt/apache-tomcat-8.5.5/conf folder
in it edit the file server.xml and insert the following line just before the </Host> in the end

**<Context path="/ws1.com" docBase="/home/student/j2eeapps/wsexample1/" debug="0" reloadable="true" crossContext="false">
</Context>**

Following is the screenshot of the end part of the server.xml file



```
        <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
               prefix="localhost_access_log" suffix=".txt"
               pattern="%h %l %u %t &quot;%r&quot; %s %b" />

<Context   path="/ws1.com"   docBase="/home/student/j2eeapps/wsexample1/" debug="0" reloadable="
true"  crossContext="false">
</Context>

      </Host>
    </Engine>
  </Service>
```

After making the changes, start tomcat as done earlier, then start browser and type the following URL

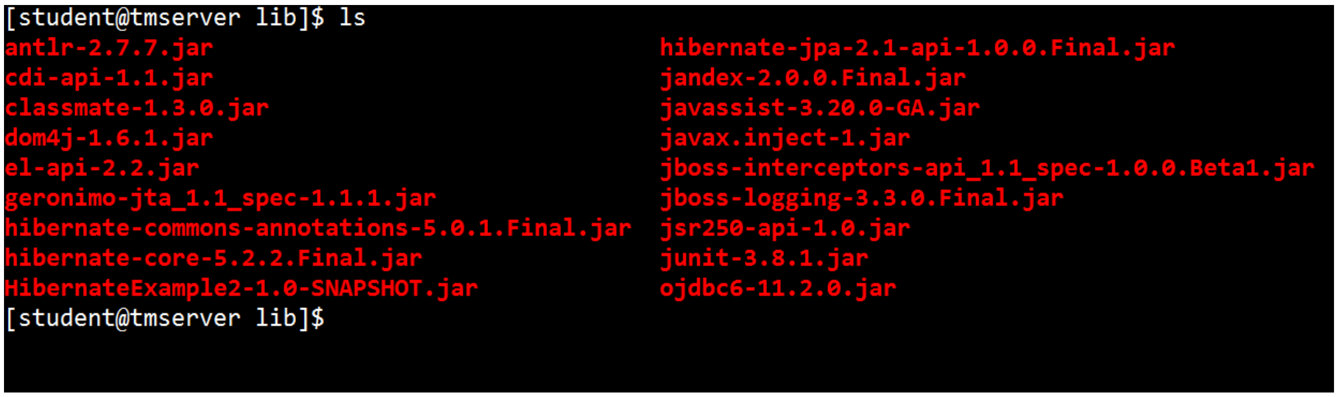
http://localhost:5050/ws1.com/test.html and you should be able to see the ui that says

Test Page of ws1.com

Now let us copy the required jar files that we already have
move the wsexample1/WEB-INF/lib folder and type the following

```
cp   /home/student/hibernateeg/example2/target/HibernateExample2-1.0-SNAPSHOT.jar   .
cp   /home/student/hibernateeg/example2/target/dependency/*.jar     .
```

Now my lib folder contains the following jar files



```
[student@tmserver lib]$ ls
antlr-2.7.7.jar                                 hibernate-jpa-2.1-api-1.0.0.Final.jar
cdi-api-1.1.jar                                 jandex-2.0.0.Final.jar
classmate-1.3.0.jar                             javassist-3.20.0-GA.jar
dom4j-1.6.1.jar                                 javax.inject-1.jar
el-api-2.2.jar                                  jboss-interceptors-api_1.1_spec-1.0.0.Beta1.jar
geronimo-jta_1.1_spec-1.1.1.jar                 jboss-logging-3.3.0.Final.jar
hibernate-commons-annotations-5.0.1.Final.jar   jsr250-api-1.0.jar
hibernate-core-5.2.2.Final.jar                  junit-3.8.1.jar
HibernateExample2-1.0-SNAPSHOT.jar              ojdbc6-11.2.0.jar
[student@tmserver lib]$
```

Now let us download jersey framework
Note : Remove all the jersey jars or zip files that I had asked you to download earlier. And also empty the contents of the folder : wsexample1/WEB-INF/lib.

We will first create 2 examples by the name of jerseyone and jerseytwo and then complete the wsexample1

in home/student folder if not yet created, create a folder named as jersey and move into it

Now lot of dependencies are need to be downloaded from various sites, it will be very easy if we use maven, but we will do that later, right now just download the following zip file using

wget "http://tm-certificates.com/Jersey2.15RequiredJars.zip"

After the download is complete unzip it using

unzip Jersey2.15RequiredJars.zip

After unzipping, you can remove the zip file using the rm command.

jerseyone

Now move to /opt/apache-tomcat-8.5.5/webapps folder and create a folder named as jerseyone followed by WEB-INF, WEB-INF\classes and WEB-INF\lib folders as we always do.

Copy the jar files from /home/student/jersey folder to WEB-INF\lib folder
while staying in lib folder type the following

```
cp   /home/student/jersey/*.jar   .
```

Now create the following classes
Note : first of all create the necessary package folders in classes folder.

```
package com.thinking.machines.webservices.pojo;
public class Brand implements java.io.Serializable,Comparable<Brand>
{
private int code;
private String name;
public Brand()
{
code=0;
name="";
}
public void setCode(int code)
{
this.code=code;
}
public int getCode()
{
return this.code;
}
public void setName(String name)
{
this.name=name;
}
public String getName()
{
return this.name;
}
public boolean equals(Object object)
{
if(!(object instanceof Brand)) return false;
return this.code==((Brand)object).code;
}
public int compareTo(Brand brand)
{
return this.name.compareToIgnoreCase(brand.name);
}
public int hashCode()
{
return this.code;
}
}
```

```
package com.thinking.machines.webservices;
import com.thinking.machines.webservices.pojo.*;
import javax.ws.rs.*;
import java.util.*;
import javax.ws.rs.core.*;
@Path("/brand")
public class BrandService
{
@GET
@Produces(MediaType.APPLICATION_JSON)
@Path("getAll")
public List getBrands()
{
List brands=new ArrayList();
Brand brand=new Brand();
brand.setCode(101);
brand.setName("Picasso");
brands.add(brand);
brand=new Brand();
brand.setCode(102);
brand.setName("Flora");
brands.add(brand);
brand=new Brand();
brand.setCode(103);
brand.setName("Novelty");
brand=new Brand();
brand.setCode(104);
brand.setName("Bosco");
brands.add(brand);
return brands;
}
@POST
@Consumes({MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_JSON})
@Path("add")
public Map<String,Object> addBrand(Brand brand) throws Exception
{
System.out.println("Adding brand : Code - "+brand.getCode()+", Name - "+brand.getName());
HashMap<String,Object> response;
response=new HashMap<String,Object>();
response.put("success",true);
response.put("message","Brand added.");
return response;
}
}
```

---

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">
    <description>
      Servlet and JSP Examples.
    </description>
    <display-name>Servlet and JSP Examples</display-name>
<servlet>
        <servlet-name>Rest With Jersey2</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>com.thinking.machines.webservices</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Rest With Jersey2</servlet-name>
        <url-pattern>/inventoryService/*</url-pattern>
    </servlet-mapping>
</web-app>
```

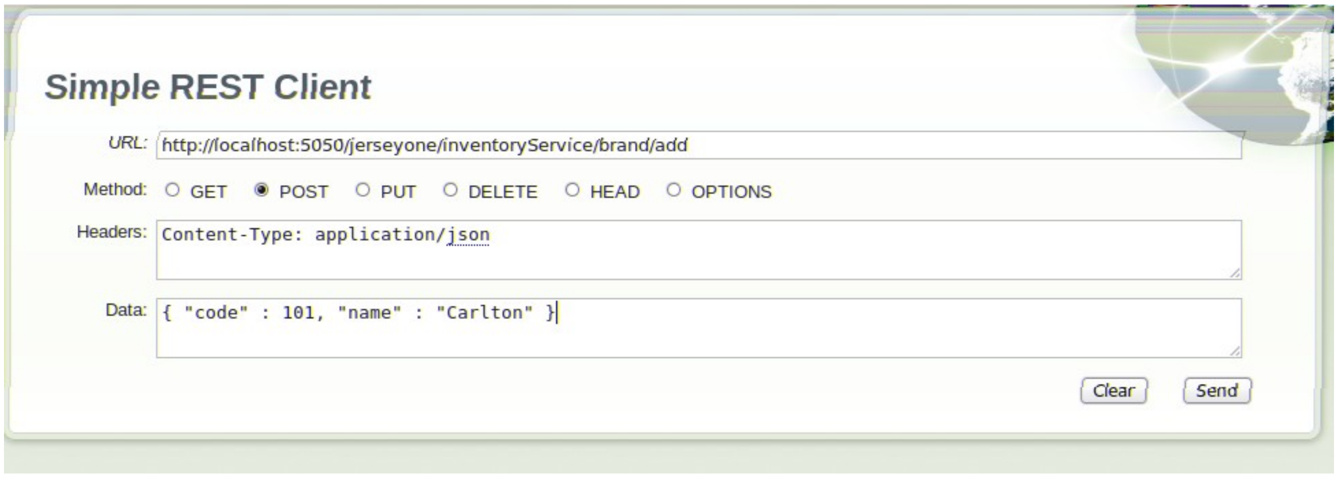
Thats it
Now start server
Switch to GUI
Now using browser we will send the get request t get all brands, for that type

http://localhost:5050/jerseyone/inventoryService/brand/getAll
in the address bar
You should see the json response.

Now we need to send a post request, that cannot be done by typing in the address bar. Hence there are 2 options.

Either we create a html with form and the necessary code to post data or we use some REST client add on available for opera as discussed in the classroom session.
Start opera and use the following link and then add the extension.
https://addons.opera.com/en/extensions/details/simple-rest-client/
Now on the upper right corner (Globe) , you should see an icon in opera, click it to see the following UI, type as I did and click the send button and you should see the success json as shown below

Before clicking Send button



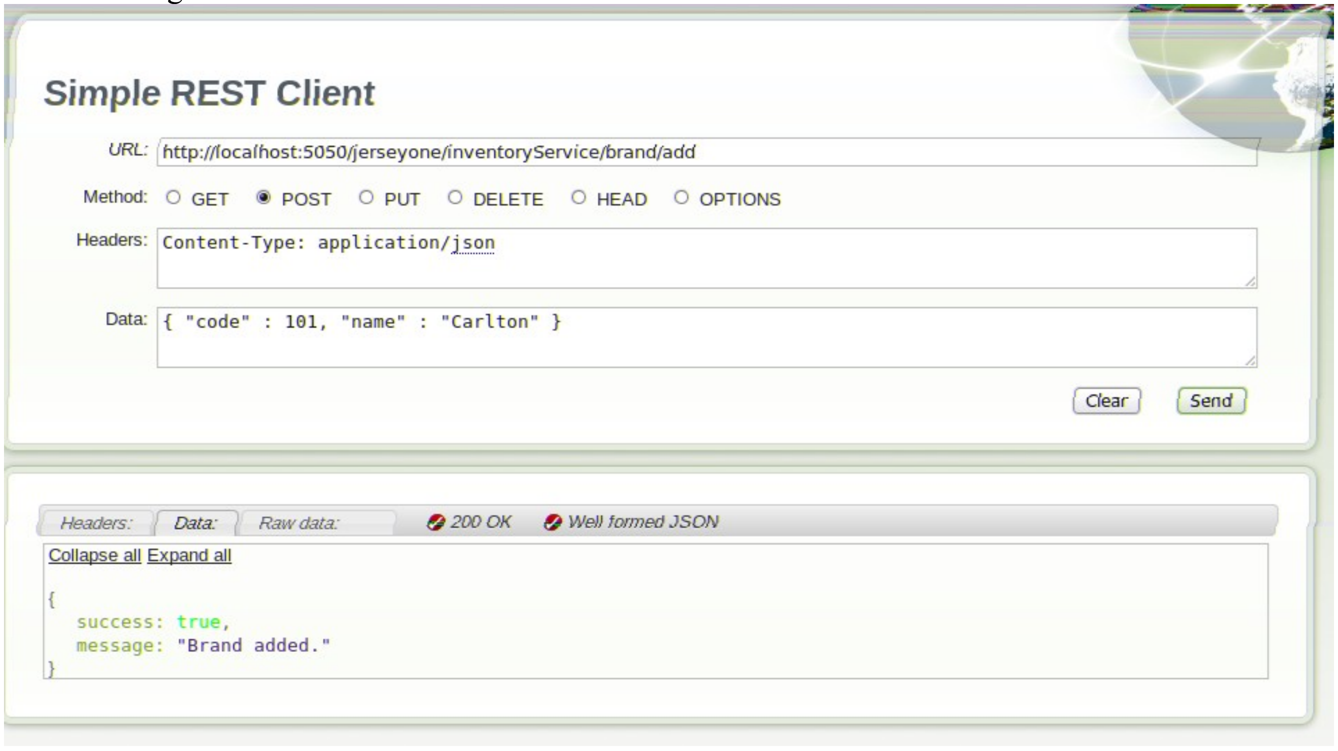
URL: http://localhost:5050/jerseyone/inventoryService/brand/add

Method: POST

Headers: Content-Type: application/json

Data: { "code" : 101, "name" : "Carlton" }

After clicking send button



URL: http://localhost:5050/jerseyone/inventoryService/brand/add

Method: POST

Headers: Content-Type: application/json

Data: { "code" : 101, "name" : "Carlton" }

200 OK Well formed JSON

```
{
   success: true,
   message: "Brand added."
}
```

jerseytwo

Now following are the files for webapplication jerseytwo, create that in the webapps folder and test it as we did for jerseyone.

```
package com.thinking.machines.webservices.pojo;
import com.fasterxml.jackson.annotation.*;
import com.fasterxml.jackson.annotation.JsonInclude.*;
@JsonInclude(Include.NON_EMPTY)
public class Brand implements java.io.Serializable,Comparable<Brand>
{
private int code;
private String name;
public Brand()
{
code=0;
name="";
}
public void setCode(int code)
{
this.code=code;
}
public int getCode()
{
return this.code;
}
public void setName(String name)
{
this.name=name;
}
public String getName()
{
return this.name;
}
public boolean equals(Object object)
{
if(!(object instanceof Brand)) return false;
return this.code==((Brand)object).code;
}
public int compareTo(Brand brand)
{
return this.name.compareToIgnoreCase(brand.name);
}
public int hashCode() { return this.code; }
}
```

```
package com.thinking.machines.webservices.provider;
import javax.ws.rs.ext.*;
import com.fasterxml.jackson.annotation.JsonInclude.*;
import com.fasterxml.jackson.databind.*;
@Provider
public class InventoryWebServiceJSONProvider  implements ContextResolver<ObjectMapper>
{
private static ObjectMapper OBJECT_MAPPER=new ObjectMapper();
static
{
System.out.println("Setting up Provider Data Structure");
OBJECT_MAPPER.setSerializationInclusion(Include.NON_EMPTY);
OBJECT_MAPPER.disable(MapperFeature.USE_GETTERS_AS_SETTERS);
}
public InventoryWebServiceJSONProvider()
{
System.out.println("Inventory Web Service JSON Provider Instantiated");
}
public ObjectMapper getContext(Class<?> type)
{
System.out.println("Providing Provider Data Structure for type : "+type);
return OBJECT_MAPPER;
}
}
```

```
package com.thinking.machines.webservices.application;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import java.util.*;
@ApplicationPath("/inventoryService")
public class InventoryApplication extends Application
{

public Set<Class<?>> getClasses()
{
System.out.println("Building  resources data structure.");
Set<Class<?>> resources=new HashSet<>();
resources.add(org.glassfish.jersey.jackson.JacksonFeature.class);
resources.add(com.thinking.machines.webservices.provider.InventoryWebServiceJSONProvider.class);
resources.add(com.thinking.machines.webservices.BrandService.class);
System.out.println("Resource building complete.");
return resources;
}
public Set<Object> getSingletons()
{
System.out.println("Setting up zero singletons");
return Collections.emptySet();
}
public Map<String,Object> getProperties()
{
System.out.println("Disabling WADL, which is enabled by default in jersey 2.5 and above.");
Map<String,Object> properties=new HashMap<>();
properties.put("jersey.config.server.wadl.disableWadl",true);
return properties;
}
}
```

```
package com.thinking.machines.webservices;
import javax.ws.rs.*;
import javax.ws.rs.core.*;
import com.thinking.machines.webservices.pojo.*;
import java.util.*;
@Path("/brand")
public class BrandService
{
@Path("getAll")
@GET
@Produces({MediaType.APPLICATION_JSON})
public List getBrands()
{
List brands=new ArrayList();
Brand brand=new Brand();
brand.setCode(101);
brand.setName("Picasso");
brands.add(brand);
brand=new Brand();
brand.setCode(102);
brand.setName("Flora");
brands.add(brand);
brand=new Brand();
brand.setCode(103);
brand.setName("Novelty");
brands.add(brand);
return brands;
}
@POST
@Consumes({MediaType.APPLICATION_JSON})
@Produces({MediaType.APPLICATION_JSON})
@Path("add")
public Map<String,Object> addBrand(Brand brand) throws Exception
{
System.out.println("Adding brand : Code - "+brand.getCode()+", Name - "+brand.getName());
HashMap<String,Object> response;
response=new HashMap<String,Object>();
response.put("success",true);
response.put("message","Brand added.");
return response;
}
}
```

---

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">
    <description>
      Servlet and JSP Examples.
    </description>
    <display-name>Servlet and JSP Examples</display-name>
</web-app>
```
