Introduction

java.exe is a program implemented in **C** language.
We use this program to run our java programs. (Code in the class bytes)
This program is called as **java virtual machine**.
java.exe is nothing but our **JVM** (java virtual machine).

D:\> java App

When we issue the above command the program called JVM will be started.

This JVM searches for the file App.class in the CLASSPATH.
If App.class is not found it (JVM) thoughs **java.lang.NoClassDefFoundError.**

CLASSPATH :- is a combination of one or more paths in which the class bytes are stored.
For Ex : if he want to use **D:\>ework\App.class** Then we can specify that the class path is **D:\>ework**

On windows O/S we can use the command **set CLASSPATH = d:\ework**

We can specify multiple directory as part of the classpath is
For Ex : **D:\> set CLASSPATH = d:\ework;d:** in this case the JVM first checks for the class in work **d:\ework** directory if it is not available then it will check in the **d:** directory.

D:\ java –verbose App

As part of java 2 standard edition.

Software we get a set of **jar files .** Ex : **rt.jar**

These jar files contains the standard classes like java.lang.String , java.Util.Date etc.

Once the App.class is found the JVM loads the App.class in the memory. Apart from loading App.class it will load several other classes to see this classes we can run the JVM using **–verbose** option.

When we try to create an object ,the JVM checks for the memory in the area called as **Heap** area if the memory is not available it will through **OutOf memoryError.**

We can control the size of the Heap used by the JVM using the option

D:\>java –Xms1M App

1M stands for 1 MB memory.
ms stands for initial Heap Size

When we use above command the JVM will allocate 1 MB of memory space to store the java objects.

If we run the program

D:\> java -Xms1M -Xmx2M App

Initially 1MB of space will be allocated if that is not enough the JVM keeps on increasing the memory upto 2MB.

Note : **ms** value must be less then **mx** value**.**

The memory allocated for storing java objects is called as Heap Space.

When ever the java class is loaded JVM is responsible for executing the static blocks.

To set classpath in current directory is **set CLASSPATH = . ;**

```
class App
{
  static
   {
    System.out.println("o/p from main static block one");
   }
      public static void main(String a[])
      {
          System.out.println("o/p from main method");
      }
       static
       {
           System.out.println("o/p from main static block two");
        }
}
```

* * As part of the class we can provide any no.of static blocks.
* * When we compile a java source file the java compiler checks for the constructor in the source file. if the constructors are not there in source file add a zero arguments constructor to the class file (it can be done by jvm).

Ex:-

```
class App
      {
        Public static void main(String args[])
         {
            System.out.println("o/p from main method");
         }
       }
```

In this above program the JVM will create one default constructor **App();** .

This will see in command in dos prompt C:\>**javap App**

First we have to set current directory classpath is C:\>**set CLASSPATH=.;**

After we have to type c:\>**javap App**

* * Even if we provide multiple static blocks the java compiler merges all the blocks together and treats them as a single block.

Ex: class App

```
    {
        public App()
        {
          static {
          // some method
                }
           static {
          // some method
                }
         static {
          // some method
                }
      }
       public static void main(String args[])
 {
  System.out.println("o/p from main method ");
 }
}
```

In this above program we have multiple static methods then the compiler merges all the blocks together and treats them as a single block.

Note :

If we compile the App.java program & issue the command **java App**

STEP 1:

The JVM searches for **App.class** in the class path. If it is not available it throughs an Exception .

STEP 2:

If the file is available the JVM load the class (means reading the what ever in the class) & then executes the static block.

STEP 3 :

The JVM checks for the main method if the method is not available its throughs the Exception, if it is available the JVM starts executing the code of the main method .

```
public class MyCls{
 public void mone() {
  System.out.println("m one ");
 }
  public void mtwo()  {
   System.out.println("m two");
  }
```

```
  static    {
   System.out.println("st blk of my cls Class");
  }
}
```

* * In the above program its compiling but run time its executing only static method & its giving Exception is

Exception in thread "main" java.lang.NoSuchMethodError: main

```
public class Test2
{
 public static void main(String a[])
 {
  System.out.println("-------1-------");
  MyCls o = null;
  System.out.println("-------2-------");
 }
  static
  {
   System.out.println("st blk of Test class");
  }
}
```

* When we run the above program the JVM will load **Test2.class** but it will not load **MyCls.class**

* The JVM loads the class when the class is required.

```
public class Test2
{
 public static void main(String a[]) throws Exception
 {
  System.out.println("-------1-------");

  MyCls o = new MyCls();
  System.out.println("-------2-------");
 Class.forName("MyCls");
}
  static
  {
   System.out.println("st blk of Test class");
  }
}
```

* in the above program The JVM load s the class of MyCls.class file and display the in that static method results.

* Class.forName() method loads the java class if it was not loaded earlier .

Class c = Class.forName("MyCls");

When the above line is executed if require the class MyCls will be loaded and an object will be created (the object will be of type class)

* * By using the object of type class

We will be able to get the information like

1. name of the class .
2. the methods in the class
3. the constructors in the class
4. the information about the package.
5. the name of the super class etc.

Class c1 = Class.forName("MyCls");
Object o1 = c1.newInstance();

The above two statements can be considered as to be equivalent to

Object o1 = new MyCls();

Object :- Instances of class is called Object .

```
public class SPApp
{
public static void main(String args[]) throws Exception
 {
  String vspone = null , vsptwo = null;

  System.out.println (vspone);
  System.out.println (vsptwo);

  vspone = System.getProperty ("one");
  vsptwo = System.getProperty("two");

  System.out.println(vspone);
  System.out.println(vsptwo);
 }
}
```

These two methods called **System Property.**

If we run the above program by using the following command.

java -Done="MyName" -Dtwo="YourName" SPApp

One & two are called as the names of property My Name & Your Name are called as the values of the property.

In this above command the out put is first two statements are give **null** after that System.getProperty one & two gives is **My Name & Your Name.**

One & Two are passed using **–D** option in the above example we can detect the information about the o/s, the user who is running the o/s & the version of java using

os.name → means Operating System
user.name → means User Name
java.vm.vendor → means java company name (Sun)
java.vm.version → means java version.

properties.

These can be done in JVM.

```
public class App
{
 public static void main(String args[]) throws Exception
 {
  Class c1= Class.forName("MyCls");
  Object o1= c1.newInstance();
  System.out.println(o1.getClass());
 }
}
```

When line one is executed MyCls.class will be loaded .When the second line is executed an object will be created based on the class MyCls. While creating the object the zero Argument constructor will be used. (if there is no zero argument constructor it throughs Instances of Exception).

```
public class App2
{
 public static void main(String args[]) throws Exception
 {
  String vclsname = System.getProperty("clsname");

  System.out.println("vclsname === " +vclsname);  ----------> 1.

  Class c1= Class.forName(vclsname);  ----------> 2.

  Object o1= c1.newInstance();  ----------> 3.

  System.out.println(o1.getClass());
 }
}
```

By looking at the code we can't say which object will be created if we run the above program using the command shown below.

Java –Dclsname =”MyCls” App2

→ **D** means define the name of the property.

When line **1**. is executed **MyCls** will be stored **vclsname**

When line **2.** is executed **MyCls** will be loaded.
When line **3.** is executed it creates **MyCls** object.

E:\j2ee>java -Dclsname="MyCls" App2

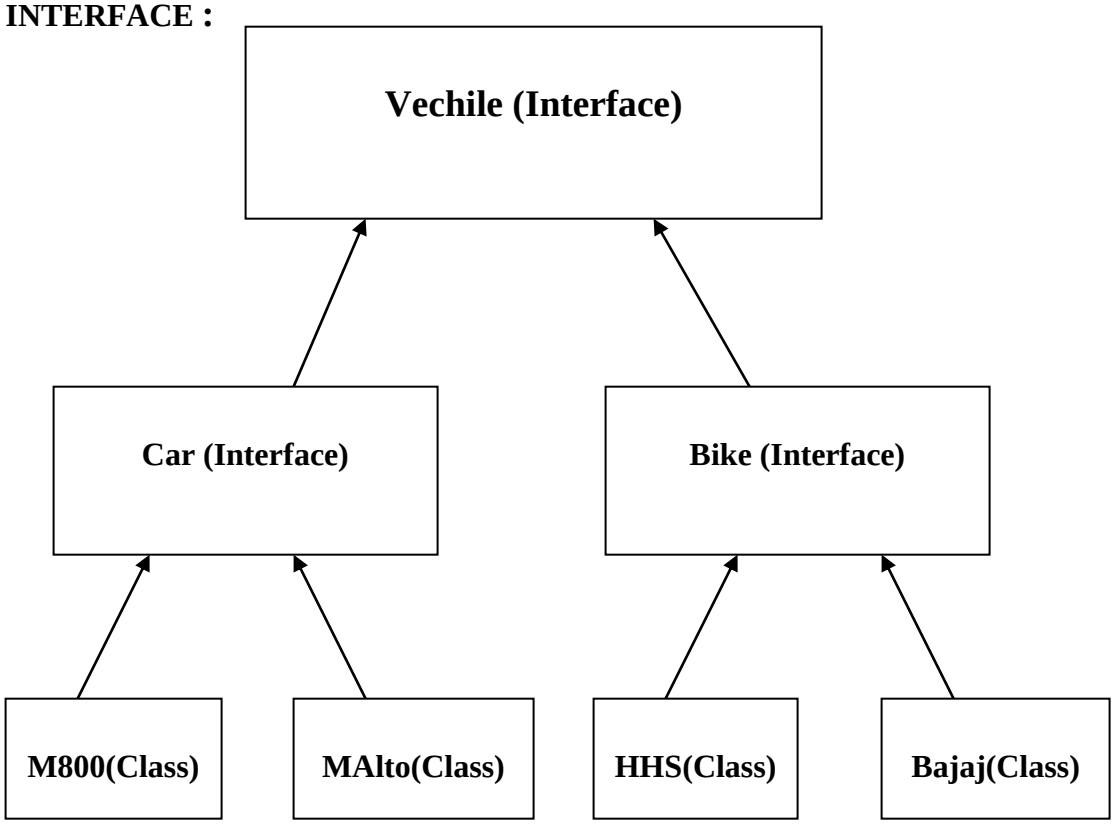
The out put of above program is

vclsname === MyCls
st blk of my cls Class
class MyCls

* * The above program can create an object based on any class in which a zero argument constructor is provided.

- * The above technique is used to create object in the servers like **TOMCAT,WEB LOGIC, WEB SPEHERE, GPASS Etc.**

INTERFACE :



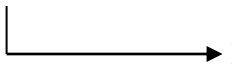
M800 x = new M800(); **HHS x = new HHS();**

Vechile v = new M800(); **Bajaj c = new Bajaj();**
Car c = new M800(); **Vechile v = new HHS();**

The above Statements are valid in java.

A variable of type Vechile can refer to M800 object MAlto object HHS object & Bajaj object.

Vechile v = new M800();
Car c = (Car)v;



It is an example of Narrowing .

Vechile v = new Vechile(); this Statement is invalid why because Vehile is an **Inerface**

* We can't create an object to **Interface**.
* We can't create an object based on abstract class.

```
public class Driver
{
 public static void main(String args[]) throws Exception
 {
  String vclsname = System.getProperty("clsname");
  System.out.println("vclsname === " +vclsname);
  Class c1= Class.forName(vclsname);
  Object o= c1.newInstance();
  Vechile v = (Vechile)o;
 }
}
```

If the above program is executed by using

Java –Dclsname=M800 Driver

It creates M800 object and line code will be executed successfully.

* * if we run the above program using the following command

Java –Dclsname=MyCls Driver

It creates MyCls object but MyCls can't be refered using the variable of type vechile.

When line 4 is executed **Class.CasteException** will be thrown .

* To perform any kind of data base operations the client programs must establish the connection with the server.

* After establishing the connection the client can request the server to carry out some kind of task by sending SQL statement to the server.

* After using the server the client program disconnects from the server.

/* Assumption : This interface is designed by company one */

```
public interface IDBops
{
 void openConnection(String vname, String pwd);
 void closeConnection();
 void executedSQL(String sqlstmt);
}
```

* The above interface is designed without keeping a specific database server in mind.

```
public class ConeDBops implements IDBops
{
 public void openConnection(String vname ,String pwd)
 {
  System.out.println("---one : now connect to oracle server");
 }
 public void closeConnection()
 {
  System.out.println("---one : now closing connection to oracle ");
 }
 public void executeSQL(String sqlstmt)
 {
  System.out.println("---one : now executingsql on oracle ");
 }
}
```

* Once an interface is designed any one can implement the class.
  For Ex : the above interface is designed by **company one** any no of company's can provide the classes implementing the above interface.

* the above class is implemented by company one which contains the code that can deal with oracle server.

```
public class CXDBops implements IDBops
{
 public void openConnection(String vname ,String pwd)
 {
  System.out.println("***CX : now connect to mysql server");
 }
 public void closeConnection()
 {
  System.out.println("**CX : now closing connection to mysql ");
 }
 public void executeSQL(String sqlstmt)
```

```
 {
  System.out.println("*CX : now executingsql on mysql ");
 }
}
```

* the above class provides the code that deals with mysql server.

```
/* code developed by the client */
`
public class App3
{
 public static void main(String args[]) throws Exception
 {

  String vclsname = System.getProperty("clsname"); ——————→ 1

  Class c = Class.forName(vclsname); ——————————→ 2

  Object o = c.newInstance(); ——————————→ 3

  IDBops i = (IDBops)o; ——————————→ 4

  i.openConnection("aaa","bbb"); ——————————→ 5

  i.closeConnection();
//  i.executeSQL("select * from emp");

 }
}
```

If the above code is executed by using the following command.

Java –Dclsname=ConeDBops App3

1. When the line 1 is executed ConeDBops will be stored in vclsname variable .
2. When the line 2 is executed ConeDBops class will be loaded.
3. When line 3 is executed ConeDBops object will be created.
4. When line 4 is executed variable i will refer to the object of type ConeDBops
5. When line 5 is executed openConnection method ie provided by company one in the class ConeDBops will be executed.

If we run the same program using the command
Java –Dclsname=CXDBops App3

* The above program makes use of an interface and it can deal with any server.

JDBC

API (Application Programming Interface) :

When we develop an application using **C** language we will use several API's an API for **C** programmer is a set of **C** functions.

* For a java programmer a set of inter faces & classes is an API.
* java soft has designed several API's.
* JDBC is an API designed by java soft.
* This API contains various **classes & interfaces .**
* A java programmer can use this API to access the data in the data base (we can access the data being managed by various data base server.

* As part of JDBC API java soft has designed several interfaces like.
1.Driver
2.Connection
3.Resultset.
4.Statement.
5.Callable Statement.
6.PreparedStatement.
7.ResultsetMetadata.
8.DatabaseMetadata.

* Several Company's like ORACLE, JAVASOFT, MYSQL, INET have provided the classes implementing the above interfaces.

* The classes developed by a company like ORACLE implementing the above interfaces combined together is called as JDBC Driver.

* * to see the content of jar file in dos prompt we can use the following command.

For Ex: we take **classes12.jar** file to see

D:\ temp> jar tf classes12.jar

t means table of contents, **f** means files.

* * to Extract the content of jar file we can use.

D:\ temp> jar xf classes12.jar (or) D:\ temp> jar xvf classes12.jar

X stands for Extract.

* * to combined the content of jar files we can use .

D:\ temp> jar cvf classes12.jar

c stands for combined.

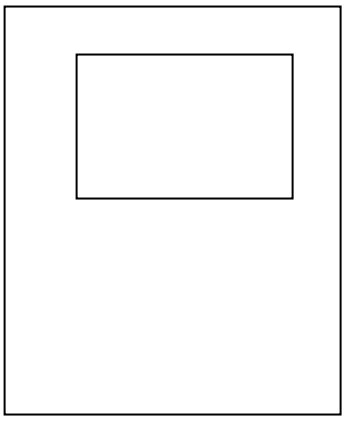
classes12.jar ⟶ **JDBC** Driver implemented by oracle corporation for oracle server.

Gate.jar ⟶ **JDBC** Driver implemented by INET for oracle , Sybase sever.

MySql…….jar ⟶ **JDBC** Driver implemented by MySql for MySql server.

In order to establish the connection with any data base server we need to use the following four things.

1. The name of the driver class.
2. JDBC URL.
3. Data base user name.
4. Data base password.

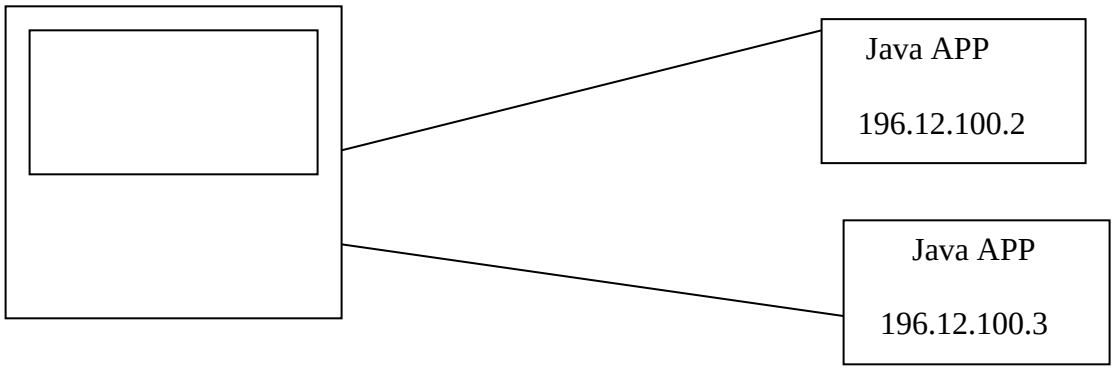


196.12.100.1
(XYZ.ABC.COM)

Oracle Administrator has to provide in address or host name of the machine on which oracle server is running port no used by oracle service name.

* When oracle is installed oracle Administrator decides about the port no used by oracle service name used by oracle.

* If we installed oracle on our machine using the default settings the port no will be **1521** & service name will be **ORCL** .



In case of MySql the seup will be similar to oracle server in case of MySql instead of service name we use the data base name.

```
import java.sql.*;
public class AppOne
{
 public static void main(String args[]) throws Exception
 {
  Driver d = new oracle.jdbc.driver.OracleDriver();

  /* we can get the name of the driver class from the documentation provided

     by the vendor.
  */

  DriverManager.registerDriver(d);

  // the above code is to register a dirver with DM.

 Connection con =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","
tiger");
 System.out.println("connected");

// to disconnect we can use.

 Con.close();

System.out.println("Disconnected");

 }
}
```

* * To establish the connection :

1. Register the Driver
2. Call DriverManager.getConnection () method.

* AppOne connect to Oracle using the JDBC Drive provided by Oracle Corporation..

```
import java.sql.*;
public class AppTwo
{
 public static void main(String args[]) throws Exception
 {
  Driver d = new Com.inet.ora.OraDriver();

  /* we can get the name of the driver class from the documentation provided
    by the vendor.  */

  DriverManager.registerDriver(d);

  // the above code is to register a dirver with DM.

 Connection con =

DriverManager.getConnection("jdbc:inetora:localhost:1521:orcl","scott","tige

r");
 System.out.println("connected");
con.close();
 System.out.println("Disconnected");

}
}
```

```
import java.sql.*;
public class GenApp
{
 public static void main(String args[])throws Exception
 {
  String vdrvcls,vurl,vuname,vpwd;
  vdrvcls = System.getProperty("drvcls");
  vurl = System.getProperty("url");
  vuname = System.getProperty("uname");
  vpwd = System.getProperty("pwd");
  Class.forName(vdrvcls);
  Connection con = DriverManager.getConnection(vurl,vuname,vpwd);
  System.out.println("Connected....");
 }
}
```

* The above program can establish the connection with any database server.

* The Commnand is :

java -Ddrvcls=oracle.jdbc.driver.OracleDriver
-Durl=jdbc:oralce:thin:@localhost:1521:orcl -Duname=scott
-Dpwd=tiger GenApp

***** If we develop an application as shown above as part of our code we can need to check the data base server. With which the program has established the connection for this we can use data base Meta Data.

*** * DatabaseMetaData** is an Interface.

databaseMetaData dbmd = con.getMetaData();
System.out.println("connected….to "+dbmd,getDatabaseProductName());

* Why because this above code is display the server name.
System.out.println("Db version " +dbmdget);

* The above command is to find the version of oracle.
* What is the Maximum size of the SQL Statement ?

Ans: DataBaseMetaData can be used to get the information about the data to a server as well as the JDBC Driver that is being used by our java application.

Class.forName("oracle.jdbc.driver.OracleDriver"); this command is used to register the Driver.

* To execute SQL statement from a java application we can use **Statement Object.**
* **Statement** is not a class it is an **Interface**.

```
import java.sql.*;
import java.io.*;
public class App10
{
 public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
       Connection con=
  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected.....");
  Statement stmt=con.createStatement();
  stmt.executeUpdate("create table tabone(col number,col2 number)");
  System.out.println("table created.............");
  //stmt.executeUpdate("insert into tabone values(1,1)");
  for(int i=1; i<10; i++)
  {
   String sqlstmt = "insert into tabone values("+i+","+i+")";
   System.out.println(sqlstmt);
```

```
  stmt.executeUpdate(sqlstmt);
 }
 System.out.println("values inserted.....");
  stmt.close();
  con.close();
 }
}
```

* * stmt executeUpdate is a method. it is executed in SQL statements.
* * The above program is insert the 10 records in database.

* We can use executeUpdate method on the statement object & execute ***non-select statement.***

* To get the data from the database we can execute a select statement it is not recommend to use **Statement.executeUpdate()** to execute a select statement.

* Some drivers throws an Exception if we use executeUpdate to execute Select Statement.

* To execute an sql statement

ResultSet rs = stmt.executeQuery(sqlstmt);

* When execute a query is executed successfully we will get an object of type ResultSet.
* We can access ***one row at a time*** using the ResultSet.

```
ResultSet rs = stmt.executeQuery(sqlstmt);
System.out.println (rs.getRow());
rs.next();
System.out.println (rs.getRow());
```

* When the ResultSet is open it will point to ***before first row.***

* When we execute ***rs.next()*** for the first time it will point to the ***first row***.

* If there is no row returned by the select statement rs.next() will not be able to point to the first row.

* If **rs.next()** is able to point to the next row it will return **true** .otherwise it will return **false**.

* **rs.getRow()** returns the current row .that is pointed by the result set . a value **zero** will be return by this method if the ResultSet is pointing to before **first row** or after **last row**.

```
 Statement stmt=con.createStatement();
 String sqlstmt = "select * from dept";
 ResultSet rs = stmt.executeQuery(sqlstmt);
 while(rs.next())
 {
  System.out.println("data in row no == "+rs.getRow());
  System.out.println("val of colone == "+rs.getString(1));
```

```
   System.out.println("val of colone == "+rs.getString(2));
   System.out.println("val of colone == "+rs.getString(3));
   System.out.println("------------------------------");
  }
  con.close();
 }
}
```

* the while loop will be executed for **n** no.of times where **n** = no.of rows return by the query.

* In the above program we have used **rs.getString(1)**
→ Coloumn name.

instead of that we can use coloumn names **rs.getString(Coloumn.name);**

Note: It is better to avoid it using * in the select statement.

```
rs.next();
System.out.println(rs.getRow());
rs.previous();
System.out.println(rs.getRow());
```
→ it will give exception given below

Exception in thread "main" java.sql.SQLException: Result set type is TYPE_FORWARD_ONLY

* When we execute **stmt.executeQuery()** of the above program it opens forward only and read only ResultSet.
* By using forward only ResultSet we can move only in one direction by using **rs.next()** It is not valid to use the methods like **rs.previous() , rs.last() , rs.first(), rs.absolute()**

rs.first() :
rs.first() can be used to point the first row.

rs.absolute() :
rs.absolute() can be used to point the particular row.
Ex: rs.absolute(3) means it will point the Third row.

rs.last() :
rs.last() can be used to point to the last row of the ResultSet.

rs.previous() :
rs.previous() can be used to move to the previous row in the ResultSet.

```
import java.sql.*;
import java.io.*;
public class jdbc1
```

```
{
 public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con =
  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected.....");
  Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE ,
  ResultSet.CONCUR_READ_ONLY);
  String sqlstmt = "select * from tabone";
  ResultSet rs = stmt.executeQuery(sqlstmt);
  while(rs.next())
  {
    rs.previous();
    System.out.println(rs.getRow());
    rs.next();
    System.out.println(rs.getRow());
    System.out.println("data in row no == "+rs.getRow());
    System.out.println("val of colone == "+rs.getString(1));
    System.out.println("val of coltwo == "+rs.getString(2));
    System.out.println("------------------------------");
 }
  stmt.close();
 }
}
```

The above program in the data base we have tabone table 10 rows then the output is

```
   connected.....
0
1
data in row no == 1
val of colone == 1
val of coltwo == 1
------------------------------
1
2
data in row no == 2
val of colone == 2
val of coltwo == 2
------------------------------
2
3
data in row no == 3
val of colone == 3
val of coltwo == 3
------------------------------
3
4
data in row no == 4
```

```
val of colone == 4
val of coltwo == 4
------------------------------
4
5
data in row no == 5
val of colone == 5
val of coltwo == 5
------------------------------
5
6
data in row no == 6
val of colone == 6
val of coltwo == 6
------------------------------
6
7
data in row no == 7
val of colone == 7
val of coltwo == 7
------------------------------
7
8
data in row no == 8
val of colone == 8
val of coltwo == 8
------------------------------
8
9
data in row no == 9
val of colone == 9
val of coltwo == 9
```

---

* **ResultSet.TYPE_SCROLL_SENSITIVE** ⟶ **1**

*** ResultSet.TYPE_SCROLL_INSENSITIVE** ⟶ **2**

We can open a bidirectional or scrollable result we can use either **1 or 2**
In case most of the JDBC Drivers there will be no different between using **1 or 2** irrespective of the changes that are made by the other client , we get the same data that was there in the data base at the time of opening the result set but theoretical Changes.

* **ResultSet.CONCUR_READ_ONLY :**

If we use CONCUR_READ_ONLY we can execute **rs.getString()** ie we can read the data but we can't execute the methods like **rs.updateString() , rs.updateDeleteRow()** etc.

Note : We can use updatable ResultSet to insert a row in a table.

```
import java.sql.*;
import java.io.*;
public class jdbc2
{
public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con =
  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected.....");
  Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE ,
  ResultSet.CONCUR_UPDATABLE);
  String sqlstmt = "select col1, col2,col3 from tabtwo";
  ResultSet rs = stmt.executeQuery(sqlstmt);

  rs.absolute(2);
  rs.deleteRow();
  rs.absolute(3);
  rs.deleteRow();
```

This statements are use to delete the particular rows in data base table.

```
 /*
  rs.moveToInsertRow();
  rs.updateString(1,"8");
  rs.updateString(2,"sdddsd");
  rs.updateString(3,"rdsser");
 rs.insertRow();
  rs.updateRow();
```

This statements are used to insert the rows in data base table.

```
rs.absolute(1);
rs.updateString(1,"shisdhi");
rs.updateString(2,"sdhishi");
  */

stmt.close();
 }
}
```

This are used to update the particular row in the data base table.

* To delete a row we can move to a specific row and execute **rs.deleteRow()**

Ex: rs.absolute(3);
rs.deleteRow();

* We can update the data in a specific row below

```
Ex:   rs.absolute(3);
      rs.updateString(2,"sssss");
      rs.updateString(3,"dsdss");
      rs.updateRow();
```

Concurrent Updatable , ResultSet can be used to perform insert , update , delete operations using the code shown above . Even though it is easy to write the code with this approach it is better to directly make use of stmt.executeUpdate() & execute the SQL statements.

* Generally the performance of an application can be improve the increasing the value of the **fetchSize** .

stmt.setfetchSize(256);

* The fetcheSize is its depend on the data base driver. Default of oracle driver is **10**.

* We can use **stmt.getFetchSize()** to get the value of the fetchSize every JDBC driver uses some default value for this. If the fetchSize is 10 the JDBC driver allocates the memory in the java application to hold 10 rows. In this case the JDBC Driver keeps on transferring if there are 1000 rows then the data will be transfer between the server & client in 100 trips . we can use the method **stmt.setFetchSize()** to set the fetch in this we have to do in before opening the ResultSet . If we set the value 100 in this space the rows can be transferred in 10 times. By reducing the no.of trips more amount of memory required in the client this shows –ve effective.

* The optimum value for the fetchSize can be decided only by experting.

* most of the applications respectively executes the same statements with a different set of values.

```
Ex:   insert into student values(1,"ravi");
      insert into student values(2,"raju");
      insert into student values(3,"sudhi");
```

the above three statements can be considered as say with a different set of values.

* We can improve the performance of such an application by using **PreparedStatement** instead of statements.

insert into student(id,name,address) values('1','sudhi','cherlapally');

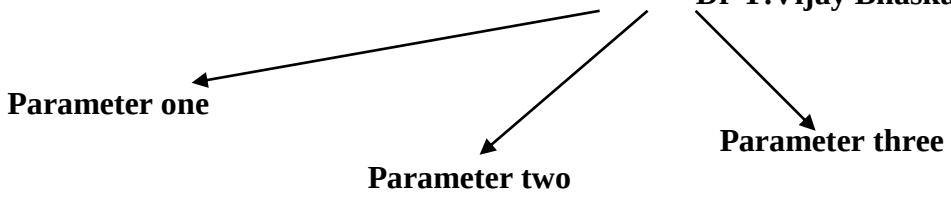
insert into student(id,name,address) values('2','malli','ramainpet');

insert into student(id,name,address) values('3','prashant','lalapet');

* The above three statements are same but with different set of values.

* We use the Prepared Statements we must use SQL statements with **place holders /parameters(represented using ?)**

For Ex: **insert into student (id, name, address) values (? , ? , ?);**



Procedure for using PrepareStatement :

1. instead of using **con.createStatement()** we must use **con.prepareStatement()**

Ex: **PreparedStatement stmt = con.prepareStatement("insert into student (id, name, address) values (?, ?, ?);**

2. set the values of the parameters using the method like **stmt.setString()**

Ex:
```
stmt.setString(1,"1");
stmt.setString(2,"sudhi");
stmt.setString(3,"cherlapally");
```

sample program is :

```
import java.sql.*;
import java.io.*;
public class jdbc2
{
public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con =
 DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
 System.out.println("connected.....");

PreparedStatement stmt = con.prepareStatement("insert into student (id,name,address)
   values(?,?,?)");
  stmt.setString(1,"1");
  stmt.setString(2,"sudhi");
  stmt.setString(3,"cherlapally");
  stmt.executeUpdate();
  System.out.println("Record is inserted.....");
  stmt.close();
 }
}
```

3. To execute the statement use executeQuery for select Statement & Execute Update for other statement is **stmt.executeUpdate()**.

4. We can repeat step2 & step3 any no.of times with different sets of values.

For executing the following three SQL statements

delete from student where id=1

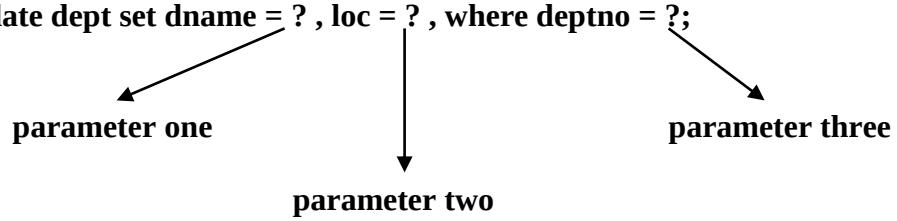
delete from student where id=2
delete from student where id=3

for the prepared statement we must provide the following SQL statement.

PreparedStatement stmt = con.prepareStatement("delete from student where id=?")
stmt.setString(1,"1");
stmt.executeUpdate();
stmt.setString(1,"2");
stmt.executeUpdate();
stmt.setString(1,"3");
stmt.executeUpdate();

* To execute the SQL statements shown below
update dept set dname = 'new research' , loc = 'hyd' , where deptno = 20;

We can use
update dept set dname = ? , loc = ? , where deptno = ?;



preparedStatement stmt = con.prepareStatement("update dept set dname =?, loc=?, where deptno=? ");

* Every business application issued several select statements which returns **zero** or **one** rows

For Ex**: zero row** means **select dname,loc from dept where deptno=10;** it is there in data base

One row means **select dname,loc from dept where deptno=22;** it is not there in data base

For retrieving the data base table in our java program is shown sample program:

```
import java.sql.*;
 import java.io.*;
 public class jdbc2
{
public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con =
  DriverManager.getConnection("jdbc:oracle:thin:@localhos:1521:orcl","scott","tiger");
  System.out.println("connected.....");
```

```
 PreparedStatement stmt = con.prepareStatement("select dname ,loc from dept where
 deptno=? ");
 stmt.setString(1,"10");
 ResultSet rs = stmt.executeQuery();
 if(rs.next())
 {
  System.out.println(rs.getString("dname"));
  System.out.println(rs.getString("loc"));
 }
  else
  {
   System.out.println("no record found");
  }
}
}
```

Note :

1.When we start oracle server it allocates sum amount of memory space for storing things like "**PARSED SQL STATEMENT**" . this area is called as **shared pool** . A data base administrator can use the following SQL statement to clear shared pool.

a**lter system flush shared_pool;**

2. A data base administrator can use the following SQL statement to the parsed SQL statements that are currently available.

select sql _text from v$sqlarea;

3. When an SQL statement is sent to the oracle server the server splits up the statement into multiple pieces it checks whether the statement is valid or not.

4. If the statement is not valid oracle server returns an error. If the statement is valid oracle server stores this statement in shared pool.

5. After placing the statements in the shared pool oracle server will execute the statement.

```
import java.sql.*;
import java.io.*;
public class jdbc5
{
 public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=
  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected & Please click the enter button.....");
  Statement stmt = con.createStatement();
  stmt.executeUpdate("create table sudhi(col1 number, col2 number , col3 number)");
  System.in.read();
```

```
  System.in.read();
  System.out.println(" Table is created & Please click the enter button.....");
  System.in.read();
  System.in.read();
  System.out.println("First Record is inserted & Please click the enter button.....");
  stmt.executeUpdate("insert into sudhi values(1,1,1)");
  System.in.read();
  System.in.read();
  System.out.println("First Record is inserted & Please click the enter button.....");
  System.in.read();
  System.in.read();
  stmt.executeUpdate("insert into sudhi values(2,2,3)");
  System.out.println("Second Record is inserted & Please click the enter button.....");
  System.in.read();
  System.in.read();
  stmt.executeUpdate("insert into sudhi values(3,1,1)");
  System.out.println("Third Record is inserted & Please click the enter button.....");
  System.in.read();
  System.in.read();
  System.out.println("Thank u for inserting the records & bye");
  stmt.close();
 }
}
```

* In the above program JDBC Driver sends the statement to oracle server & oracle server parses the statement & if statement is correct then its creating table & inserted the tables.

* When the java program executes **stmt.executeQuery()** The JDBC driver sends the statements to the data base server. After receiving the statement the server is responsible for

1. Parsing the statement.
2. If the statement is valid the data base server executes the statement.

* When the above code is executed three times the data base server parses the statements & three times the data base server executed the statement.

```
import java.sql.*;
import java.io.*;
public class jdbc7
{
 public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected & Please click the enter button.....");
  String sqlstmt= "insert into sudhi values (?,?,?)";
  PreparedStatement stmt = con.prepareStatement(sqlstmt);
  System.in.read();
  System.in.read();
  stmt.setInt(1,30);
```

1 ⟶

```
  stmt.setInt(2,31);
  stmt.setInt(3,32);
  stmt.executeUpdate();  ------------------------------------------------------> 2
  System.out.println("First record is inserted");
  System.in.read();
  System.in.read();
  stmt.setInt(1,40);
  stmt.setInt(2,41);
  stmt.setInt(3,42);
  stmt.executeUpdate();  ------------------------------------------------------> 3
  System.out.println("First record is inserted");
  System.in.read();
  System.in.read();
  stmt.setInt(1,50);
  stmt.setInt(2,51);
  stmt.setInt(3,52);
  stmt.executeUpdate();  ------------------------------------------------------> 4
  System.out.println("First record is inserted");
  System.in.read();
  System.in.read();
  System.out.println("Thank u for inserting the records & bye");
  stmt.close();
 }
}
```

The above program when

Step1 is executed the JDBC driver make **1** keeps the statement. Send the SQL statements to the data base server so that the data base server can parse the statement .

Step2 is executed the driver may **First** send to the oracle server for parsing & send the values to the data base server for executing. **Second** if the statement is already is parsed the JDBC driver send the values to the server & the server execute the statement.

Step3 is executed the JDBC driver sends the values to the server & the server directly execute the statements.

Step4 is executed same as step3.

* as the no.of parses (or) less in case of PreparedStatement the application that has to repeatedly executed the same statement with different values get benefited.

* In every data base application to carry out a task the application need to execute several statements failure in executing any of the statements leaves to a problem to avoid this problem we can execute a set of statements together as part of a **TRANCATION** .

* By default the JDBC drivers work in **auto commit mode**.

* In auto commit mode after executing the SQL statements the JDBC driver asks the data base server to commit to the transaction.

* **in java program to control the transactions we can write the code using the following steps :**

Step1 : **setAutoCommit(false);**

Step2 : execute the SQL statement to perform the task.

Step3 : if all the statements are executed successfully execute **con.commit();**

Step4 : if application fills to execute one of the statement execute **con.rollback();**

* In JDBC API there is no method available to start a transaction .

* A transaction will be started then the client establishes the connection with the server or the transaction will be started after ending the current transaction using methods like **con.commit() & con.rollback();**

```
import java.sql.*;
import java.io.*;
public class jdbc8
{
 public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=
  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected ......");
  Statement stmt= con.createStatement();
  con.setAutoCommit(false);
  try
  {
   stmt.executeUpdate("create table pastab (name varchar(10),id number)");
   System.out.println("table created.....");
   stmt.executeUpdate("insert into pastab values('sudarshan',1)");
   stmt.executeUpdate("insert into pastab values ('********************',2)");
   System.out.println("executed sucessfully");
   con.commit();
  }
  catch (Exception e)
   {
     System.out.println(e);
     con.rollback();
   }
 }
}
```

When the above program establishes the connection a transaction will be started .as we have **setAutoCommit(false)** the JDBC driver will not issue a commit after executing the SQL statements. First insert statement will be executed successfully. Second insert statement fails as

the size of the first column in the table to chars, as this statements fails the remaining part of try block will not be executed the JVM executes the catch block. In the catch block it ill executes **con.rollback()** when this is executed the transaction will be rolled back.

* If the above two insert statements executes successfully **con.commit()** will be executed. So the changes made to the data base will be made permanently.

* If a program gets terminated abnormally while performing a transaction the changes made in the current transaction will be rolled back.

* If the data base server gets terminated when a program is performing transaction the transaction will be rolled back.

* * Transactions are not supported by MySql.
* the default drivers in MySql does not support transaction in java.
* MySql data base support multiple engines to manage the data.
* The default engine doesn't support any transaction .
* **Innodb** engine supports transactions. If **Innodb** engine has to be used we must create the table in shown below

Create table MyTable(col1 varchar(20),col2 varchar(20)) type= innodb

* Desc is not SQL statement.
* To get the information about the columns in a java program can use result set metadata.

```
import java.sql.*;
import java.io.*;
public class jdbc9
{
 public static void main(String args[]) throws Exception
 {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=
DriverManager.getConnection("jdbc:iracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected ......");
  Statement stmt= con.createStatement();
  try
  {
  ResultSet rs=stmt.executeQuery("select * from pastab");
  ResultSetMetaData rsmd=rs.getMetaData();
  int noc= rsmd.getColumnCount();
  System.out.println("Number of counts ="+noc);
  }

  catch (Exception e)
   {
    System.out.println(e);
    con.rollback();
   }
 }
```

```
}
```

* get Column count returns the no.of columns selected by the SQL statement * is use as in the above example it returns the no.of columns.

* getMetData() in connection gives information of database

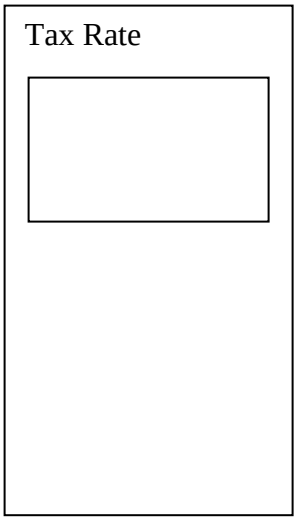
```
import java.sql.*;
import java.io.*;
public class jdbc10{
 public static void main(String args[]) throws Exception {
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection con=
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  System.out.println("connected ......");
  Statement stmt= con.createStatement();
  ResultSet rs=stmt.executeQuery("select * from pastab");
  ResultSetMetaData rsmd=rs.getMetaData();
  int noc= rsmd.getColumnCount();
  System.out.println("Number of counts ="+noc);
  for(int i=1; i<=noc;i++)  {
   System.out.println("column name is = = = "+rsmd.getColumnName(i));
   System.out.println("column type is = = = "+rsmd.getColumnType(i));
  }
 }
}
```

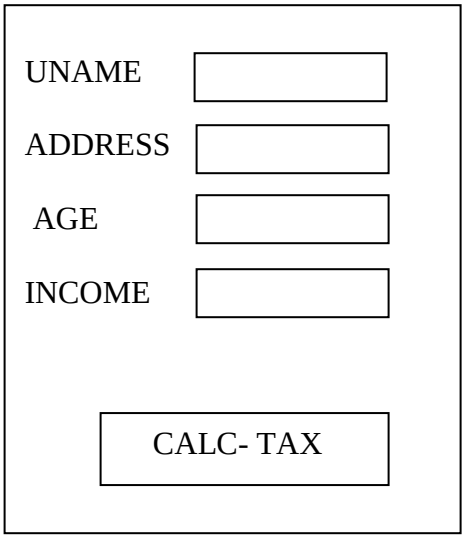
* getColumnType returns a no indicating the data type of the column.

```
import java.sql.*;
public class MyClass
{
 public static void main(String args[])throws Exception
 {
  System.out.println(Types.VARCHAR); ⟶ Ans is : 12
  System.out.println(Types.DATE);  ⟶ Ans is :91
  System.out.println(Types.INTEGER); ⟶ Ans is :4
  System.out.println(Types.FLOAT);  ⟶ Ans is :6
  System.out.println(Types.BOOLEAN); ⟶ Ans is :16
  System.out.println(Types.TIME);  ⟶ Ans is :92
  System.out.println(Types.TIMESTAMP); ⟶ Ans is :93
  System.out.println(Types.NUMERIC);  ⟶ Ans is :2
 }
}
```

* As a part of java.sql **Type class** several constants are defined representing various data types shown in the above program.
* **SQL 92** is standard that's specify the grammar for the SQL language and various data types that has to be supported by a data base server some of these data types are **bit , varchar , Boolean , blob , clob , time , date , time , timestamp.**
* In the JDBC API as part of types java.sql.types class various constants are defined to representing various data types.

* A data base server may not support all these data types, for ex: MySql doesn't support the data type Boolean instead of Boolean we can use **bit** .
* according to SQL92 standard DATE data type can be use to store only date time data type can be use to store only time . timestamp can be used to store DATE+TIME.
* But in oracle by using the Date data type we can store both date & time , we can say SQL 92 timestamp=oracle date.
* In every business application we need to write some amount of code i.e responsible for presenting the information to the user & taking the inputs from the user this code (or) this logic is called as presentation logic.
* If we are building a business application to calculate the tax we need provide some code i.e responsible for taking from information from the user by displaying the form shown below.





Data Base Server **front end**

This code returns as presentation logic.

* In this application we need to provide the code that takes care about calculating the taxes according to the tax rules of the central government.
* This logic is called as business logic.
* Once we build a business application if the rules for calculating the tax are changed then we need to modify business logic. After building the application the customer may ask the software developer to change the code that takes care of presenting information (or) taking the information from the user.
* most of the in experiences developer mix the presentation logic with business logic.
* This is not advisable as it takes more amount of time in future while maintaining the project.
* To reduce the cost of maintenance we need build our application by supporting the business logic from presentation logic, there are multiple ways of supporting the business logic from presentation logic.

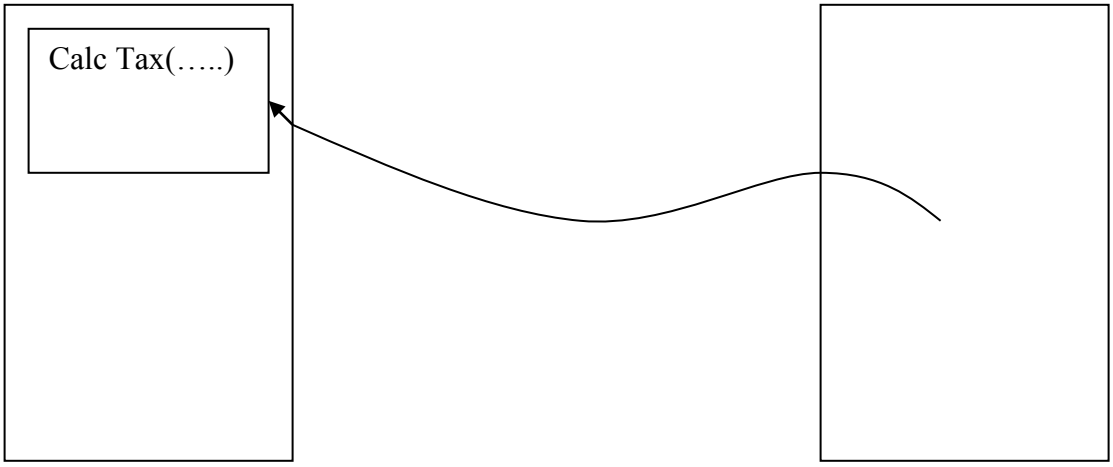
* A java programmer can place the presentation logic in a set of java classes as shown below.

```
public class TaxCalculater
{
  public float CalculateTax(int age , float income)
  {
    System.out.println("getting tax rates from db");
```

```
   System.out.println("Calc….. tax accr.to 2005 rules");
  return 20000.00f;
 }
}
```

The above code is responsible for calculating the taxes according to the rules specified for the year 2005. if there is a change in tax rules we need to modify the code in the tax calculator class.

* Another way of supporting the business logic from the presentation logic is to implement the business logic, as a stored procedure a java programmer can assume a stored procedure as a piece of code implemented in some language and stored inside the database it is the responsibilities to executes the stored procedure.



ORACLE SERVER **JAVA APPLICATION**

* By using **CallableStatement** we can invoke a stored procedure.

Example 1:

* In databse we have to create a table ie

create table mytab(col1 number, col2 number);

* After that we have to create stored procedure in oracle.
  ie

-- procedure that takes no params

```
create or replace PROCEDURE proc
as
  i number;
  BEGIN
        i :=44+ 1;
        insert into mytab values(1000,i);
 END proc;
/
```

* After that in oracle we type the below command then it will insert the record.

execute proc;

* The above code creates a stored procedure in the oracle server the name of the procedure is **proc** & it takes no parameters.

* **Procedure for calling a stored procedure from java code.**

Step 1 : Create CallableStatement object by using Connection.prepareCall method.

CallableStatement stmt = con.prepareCall(" { call proc } ");

Name of the procedure

Step2 : to invoke the stored procedure call statement.execute method.

stmt.execute();

program is below :

```
import java.sql.*;
import java.io.*;
public class proc
{
 public static void main(String args[])throws Exception
 {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection("jdbc:odbc:sudhi","scott","tiger");
  System.out.println("connected.....");
  CallableStatement stmt = con.prepareCall( " { call  proc } " );
  stmt.execute();
  System.out.println("record is inserted.....");
 }
}
```

Example 2:

Create table mytab1(col1 number, col2 number);

-- procedure that takes params

```
create or replace PROCEDURE anotherproc(pone IN NUMBER , ptwo IN NUMBER)
as
     BEGIN
          insert into mytab1 values(pone,ptwo);
  END  anotherproc;
/
```

* The above procedure name is anotherproc & it takes two input parameters & both of them of type **number**.

Step1: Create CallableStatement as shown below.

CallableStatement stmt = con.prepareCall(" { call anotherproc(?,?) } ");

Name of the procedure

Two **? ?** are represents that there are two parameters.

Step2: Using statement.setxxxxxx method set the values of the parameters.

```
int a = 22 ,b = 33 ;
stmt.setInt(1,a);
stmt.setInt(2,b);
```

Step3: Invoke the stored procedure by using statement.execute method.

stmt.execute();

program is :

```
import java.sql.*;
import java.io.*;
public class anotherproc
{
 public static void main(String args[])throws Exception
 {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection("jdbc:odbc:sudhi","scott","tiger");
  /*
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
          Connection con;
  con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott", "tiger");
  */
  System.out.println("connected.....");
  CallableStatement stmt = con.prepareCall( " { call  anotherproc(?,?) } " );
  int a = 22 ,b = 33 ;
  stmt.setInt(1,a);
  stmt.setInt(2,b);
  stmt.execute();
  stmt.setInt(1,15);
  stmt.setInt(2,95);
  stmt.execute();
  System.out.println("record is inserted.....");
 }
}
```

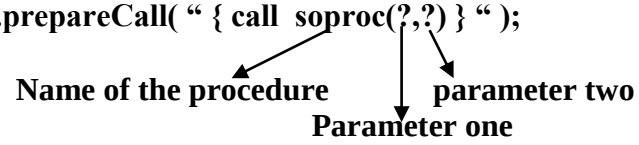
Example 3:

-- procedure that takes params

```
create or replace PROCEDURE soproc(pone OUT NUMBER , ptwo OUT NUMBER)
as
    BEGIN
          pone := 444;
          ptwo := 555;
 END  soproc;
/
```

Step1: Create CallableStatement as shown below.

CallableStatement stmt = con.prepareCall(" { call soproc(?,?) } ");



Step2: Register the data type of the output parameters of

stmt.registerOutParameter(1,Types.INTEGER);
stmt.registerOutParameter(2,Types.INTEGER);

Step3: Execute the stored procedure by calling

stmt.execute();

Step4: get the values of the output parameters

System.out.println(stmt.getInt(1));
System.out.println(stmt.getInt(2));

Sapmple program is :

```
import java.sql.*;
import java.io.*;
public class soproc
{
 public static void main(String args[])throws Exception
 {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection("jdbc:odbc:sudhi","scott","tiger");
  /*
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
          Connection con;
  con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott", "tiger");
  */
  System.out.println("connected.....");
  CallableStatement stmt = con.prepareCall( " { call  soproc(?,?) } " );
```

```
  stmt.registerOutParameter(1,Types.INTEGER);
  stmt.registerOutParameter(2,Types.INTEGER);
  stmt.execute();
  System.out.println(stmt.getInt(1));
  System.out.println(stmt.getInt(2));
 }
}
```

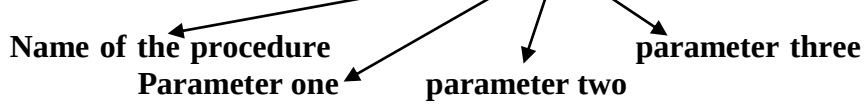
Example 4:

-- procedure that takes params

```
create or replace PROCEDURE CalTax(age IN NUMBER , income IN NUMBER , tax OUT
NUMBER)
as
     BEGIN
          Tax:=income*0.33;
  END  CalTax;
/
```

Step1: Create CallableStatement as shown below.

CallableStatement stmt = con.prepareCall(" { call CalTax(? , ? , ?) } ");



Step2: set the values of the input parameters.

```
stmt.setInt(1,22);
stmt.setFloat( 2,123456.66f );
```

Step3: Register the data type of the output parameters.

```
Stmt.registerOutParameter(3,Types.FLOAT);
```

Step4: Execute the stored procedure by calling

```
stmt.execute();
```

Step5: get the values of the output parameters

```
System.out.println(stmt.getFloat(1));
```

Sample program is :

```
import java.sql.*;
import java.io.*;
public class CalTax
{
 public static void main(String args[])throws Exception
```

```
 {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection("jdbc:odbc:sudhi","scott","tiger");
  /*
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection con;
  con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott", "tiger");
  */
  System.out.println("connected.............");
  CallableStatement stmt = con.prepareCall( " { call  CalTax(?,?,?) } " );
  stmt.setInt(1,2);
  stmt.setFloat( 2,123456.66f );
  stmt.registerOutParameter(3,Types.FLOAT);
  stmt.execute();
  System.out.println("CalTax is ="+stmt.getFloat(3));
 }
}
```

Example 5:

-- procedure that takes params

```
create or replace PROCEDURE sample(x IN NUMBER , y OUT NUMBER , z IN OUT
NUMBER)
as
    BEGIN
        y:=z+x;
        z:=y+1000;
  END  sample;
/
```

* name of the procedure is sample

input's : parameter **1** and parameter **3**

output's : parameter **2** and parameter **3**

Total no.of parameters = **3 .**

Step1: Create CallableStatement as shown below.

CallableStatement stmt = con.prepareCall(" { call CalTax(? , ? , ?) } ");

Name of the procedure
Parameter one
parameter two
parameter three

Step2: set the values of the input parameters.

```
stmt.setInt(1,22);
 stmt.setInt(3, 2 );
```

Step3: Register the data type of the output parameters.

```
Stmt.registerOutParameter(2,Types.INTEGER);
Stmt.registerOutParameter(3,Types.INTEGER);
```

Step4: Execute the stored procedure by calling

```
stmt.execute();
```

Step5: get the values of the output parameters

```
System.out.println(stmt.getInt(2));
System.out.println(stmt.getInt(3));
```

Sample program is:

```
import java.sql.*;
import java.io.*;
public class sample
{
 public static void main(String args[])throws Exception
 {
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  Connection con = DriverManager.getConnection("jdbc:odbc:sudhi","scott","tiger");
  /*
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection con;
  con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott", "tiger");
  */
  System.out.println("connected.....");
  CallableStatement stmt = con.prepareCall( " { call  sample(?,?,?) } " );
  stmt.setInt(1,1);
  stmt.setInt( 3,2);
  stmt.registerOutParameter(2,Types.INTEGER);
  stmt.registerOutParameter(3,Types.INTEGER);
  stmt.execute();
  System.out.println("sample is ="+stmt.getInt(2));
  System.out.println("sample is ="+stmt.getInt(3));
 }
}
```

* In oracle we can implement the function which are similar to a stored procedure a function can return a value :

Example 6 :

-- procedure that takes params

```
create or replace function myadd(i NUMBER,j NUMBER)
return number
as
     BEGIN
          return i+j;
    END  myadd ;
/
```

Step1: Create CallableStatement as shown below.

CallableStatement stmt = con.prepareCall(" { ?=call myadd(? , ?) } ");

Name of the procedure

(parameter two & parameter three these are Input parameters)

(Parameter one this is output parameter means it returns value of the function)

Step2: set the values of the input parameters.

stmt.setInt(2,10);
stmt.setInt(3, 20);

Step3: Register the data type of the output parameters.

Stmt.registerOutParameter(1,Types.INTEGER);

Step4: Execute the stored procedure by calling

stmt.execute();

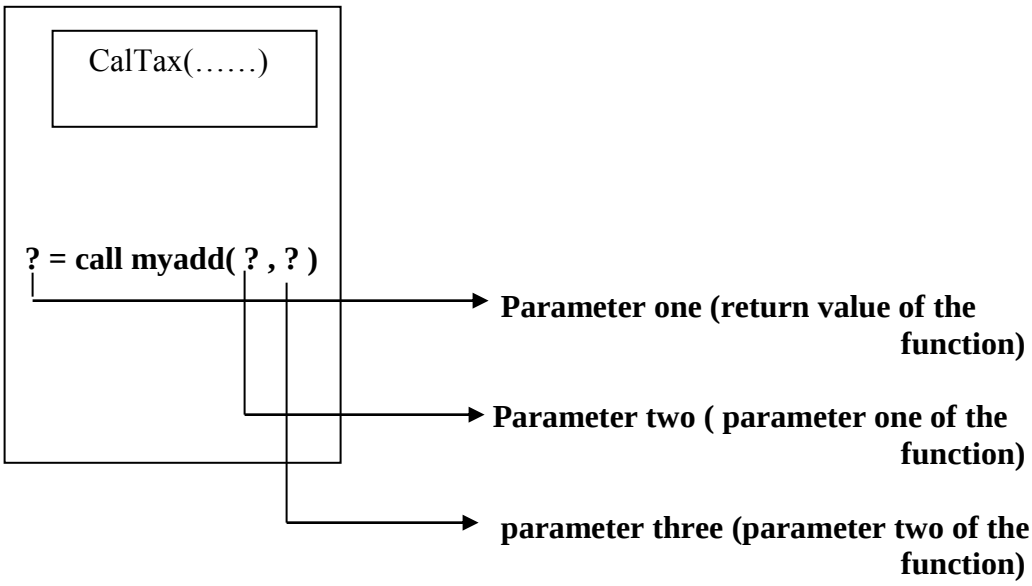
Step5: get the values of the output parameters

System.out.println(stmt.getInt(1));

Sample program is:

```
import java.sql.*;
import java.io.*;
public class myadd
{
 public static void main(String args[])throws Exception
 {
```

```
 Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
 Connection con = DriverManager.getConnection("jdbc:odbc:sudhi","scott","tiger");
 /*
 DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
        Connection con;
 con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott", "tiger");
 */
 System.out.println("connected.....");
 CallableStatement stmt = con.prepareCall( " { ?=call  myadd(?,?) } " );
 stmt.setInt(2,10);
 stmt.setInt( 3,20);
 stmt.registerOutParameter(1,Types.INTEGER);
 stmt.execute();
 System.out.println("addtion is ="+stmt.getInt(1));
  }
}
```



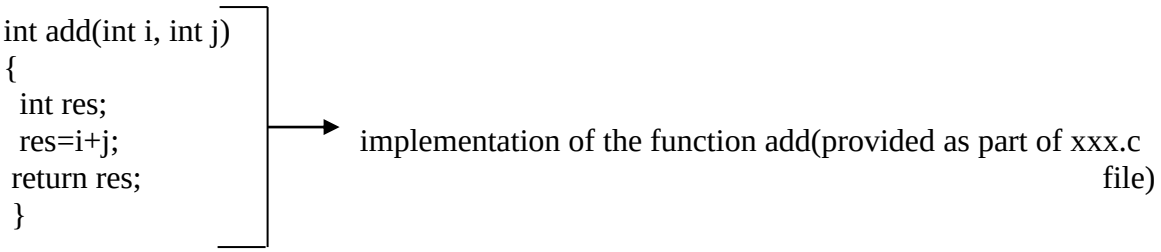
* most of the developers are using java beans (or) enterprise java beans to implement the business logic.

* A java programmers who has experience in **c,c++** codding can provide the code for the native methods start rocket, stop rocket that are declared as native in the above class.

* to do this

Step1 : we must generate the **.h** file that contains the prototype of the functions equalent the above tool native methods.

Ex: **c:\> javah ManHWDev**



int add(int i, int j);⟶ prototype of a function.

Step2 : provide the implementation of all the functions that are generated by the **javah** tool.

* Once the code is implemented we can use a compiler to generate machine level language instructions , after this a linker can be used to generate a **.dll** file .

* In the **dll** file code will be available in machine level language instructions.

* As part of the java class we can use a static block.

* In this case when the class file is loaded the JVM will execute the static block loads code ie available in **xyz.dll** now when the method start rocket is called the virtual machine is called executes the start rocket function whose code is available in **xyz.dll**.

OCI ➔ oracle call interface

* Oracle corp provides a set of **.age** & set of **.lib** & **.dll** files which can be used by a **c** programmer to developed an application in **c** language some of the functions that are available in OCI are **olog , open, oparse , ofetch , oexec ,ocom ,ologof .**

* as part of **MySql** also we get a set of **.h , .lib , .dll** files that can be used by **c** programmer the functions that are provide as part of **MySql** are called as **MySql c API** .
* some of the functions that are part of this API are **MySql_connect , MySql_close , MySql_fetch_row , MySql_query , MySql_commit , MySql_rollback , MySql_execute** like OCI for oracle for almost every type of data base server we have an API.

* ODBC is an **openAPI** Microsoft has release the documentation of ODBC API to the public.

OCI ⟶ (set of functions provided by **oracle**).

MySql C API ⟶ (set of functions provided by **MySql**).

DBlibrary API ⟶ (set of functions provided by Microsoft for **MySql**).

xxx API ⟶ for xxx DBserver.

* DBlibrary API as well as **Sybase API**.

* An open specification is a document giving the complete information and is available for every one, once the specification is available multiple company's can provided the implementation of the specification.

* J2EE is a open specification that's why many servers can be developed means **weblogic ,webspeher , sun one server ,pramathi one server ,tomcat** etc.

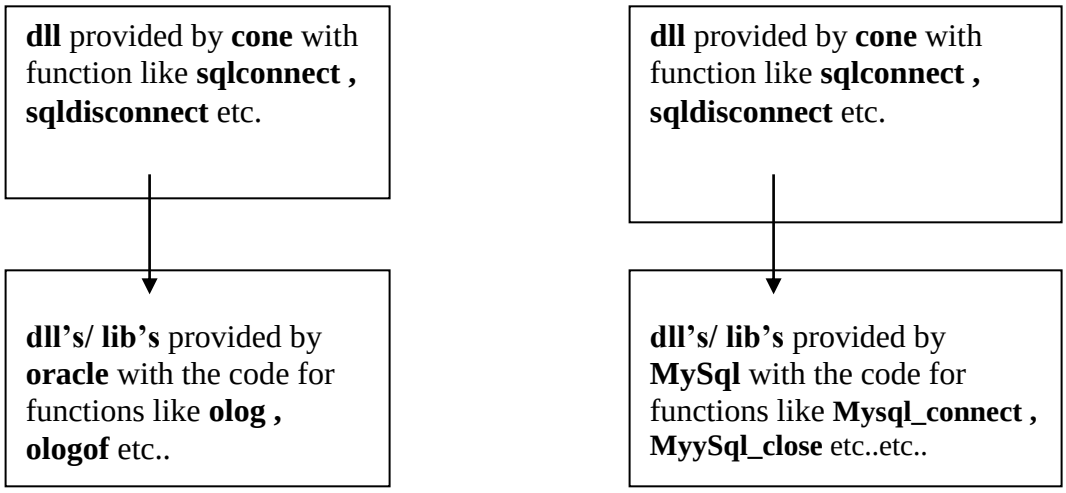
* Microsoft ODBC API specification

Info about odbc api----------

----------sqlconnect(------) ➔ is a function it is establish connection in any data base server.

* As part of ODBC specification Microsoft has specify several functions some of them are **sqlconnect , sqldisconnect , sqlexecute , sqlfetch** etc..

* A company one can provide the implementation of ODBC AP by internally making use of functions like **olog , ologof** etc. these functions implemented by company one will be provided as part of **dll** this **dll** iss called as ODBC driver for oracle , similarly company two can provided the implementation of the functions like **sqlconnect , sqldisconnect** by internally using the functions like **Mysql_connect , MySql_close** etc.. and provide this code as part of a **dll** this **dll** is called as ODBC driver for **MySql**.



* A JDBC driver implementer can implement a driver in four different ways.

*** TYPE 2** driver in these kind of drivers the implementer will internally make use native methods , inside this native methods code will be provided to make use of **data base specific API ➔ olog() ,ologof() & MySql_connect() , MySql_close() .**

* these drivers are not pure java drivers.

* **TYPE 1** driver in these kind of driver the implementer internally uses native methods in these native methods internally the ODBC functions will be used, this driver is also not a pure java driver.

* **TYPE 4** the implementer of these drivers implements the entire code for the JDBC driver in java language only . this driver is a pure java driver.

Oracle corp--------Type 2 & Type 4

* Oracle corp has implemented there JDBC driver in two different ways .
1. Type 2 driver **(** oracle corp refers to type2 driver has **OCI** driver).
2. Type 4 driver (oracle corp refers to type4 driver has **Thin** driver).

* If we develop a program using a **type 2** driver by writing thecode as shown below.

```
Class.forName("oracle.jdbc.driver.OracleDriver);
Connection con;
con= DriverManager.getConnection("jdbc:oracle:oci8:@nsname","scott","tiger");
```

* to run the program that uses **type2** drivers we must install the following softwares.
1. the JDBC driver ie **classes12.jar** .
2. data base specific client software .

* type 2 we have to use **classes12.jar + oracle client software**.

* to use **type 4** driver we need to install the only one jar file ie **classes12.jar** .

* to use **type 1** driver we must install
1. the java classes means JDBC drivers.
2. the ODBC drivers.
3. data base specific client software.

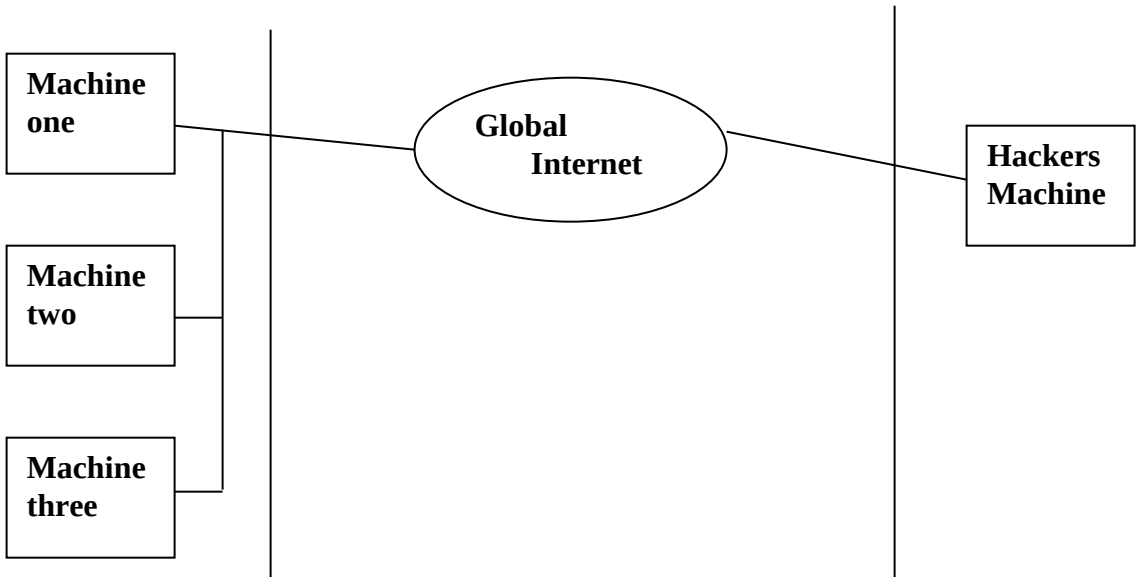
* java soft has implemented JDBC driver ie supplied as part of the java development kit this driver is implemented as **type 1** driver.

Type 1 ➔ java classes + ODBC driver +db client software.

Type 2 ➔ java classes + db client software.

Type 4 ➔ java classes.

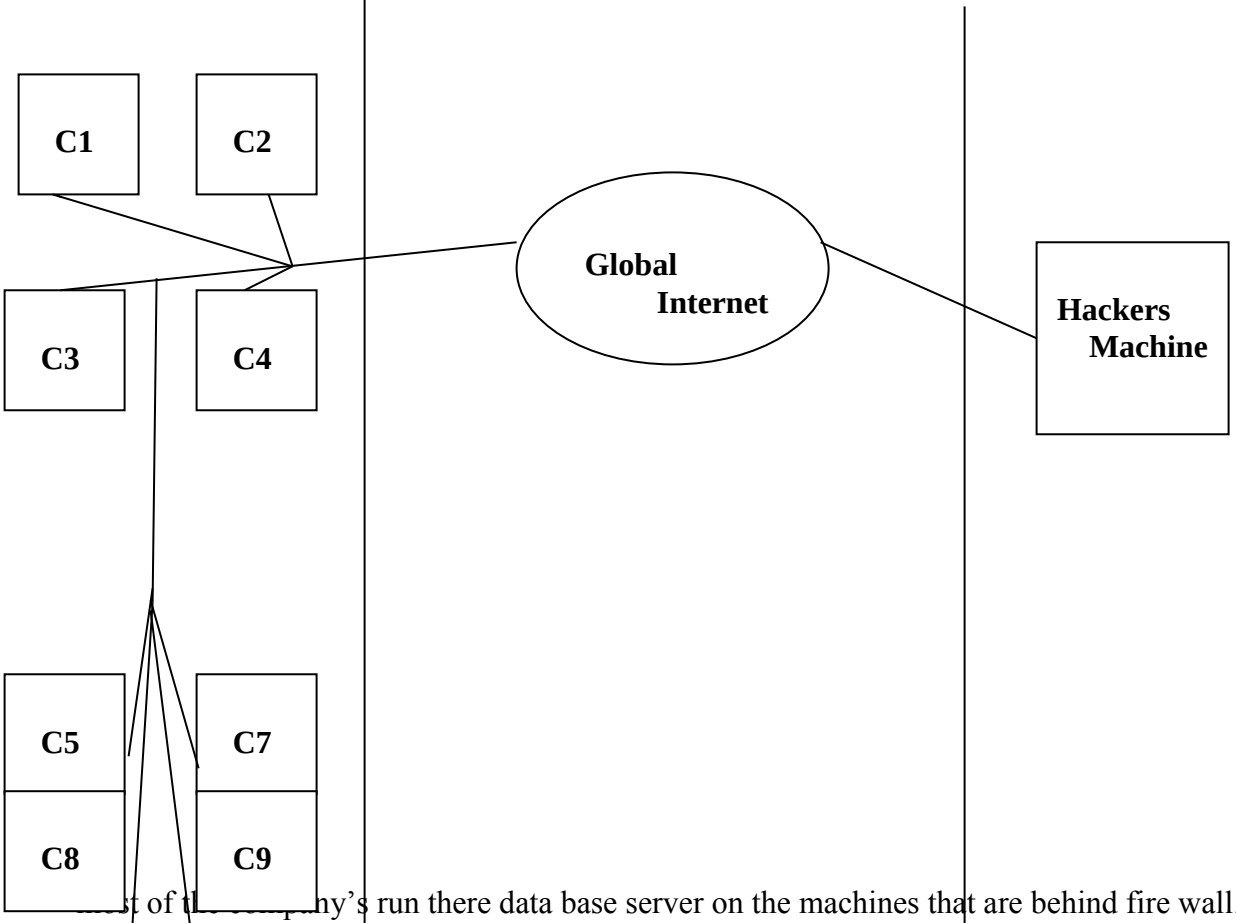
* **Type 3** drivers will be used in some special cases.



***** If a Machine is exposed to the Global internet & if we run a database server on this machine there is a high chance of attaching ie any one can write a program by using standard user names & pass wards & run this programs on there own machines that are connected to the Global internet to avoid this problem most of the company's uses a fire wall set up.

* A network admin is responsible setting up a fire wall he can set up a fire wall to unauthorized access this can be done either by using software or by using hardware devices.

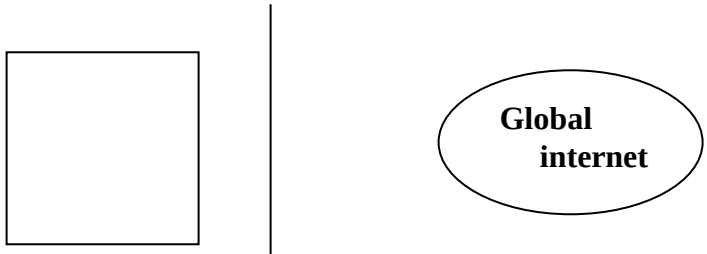
***** most of the company's setup there network by using fire wall as shown below in these kind of setup the network administrator can configure a fire wall in such a way that some machines (**c1,c2,c3,c4)** are directly exposed to the internet &the remaining machines (**c5,c6,c7,c8**) are not exposed to the Global Internet ie a program running on **Hackers Machine** can access the program's running on **c1 , c2 , c3 , c4** but not the programs running on **c5 , c6, c7 , c8** .

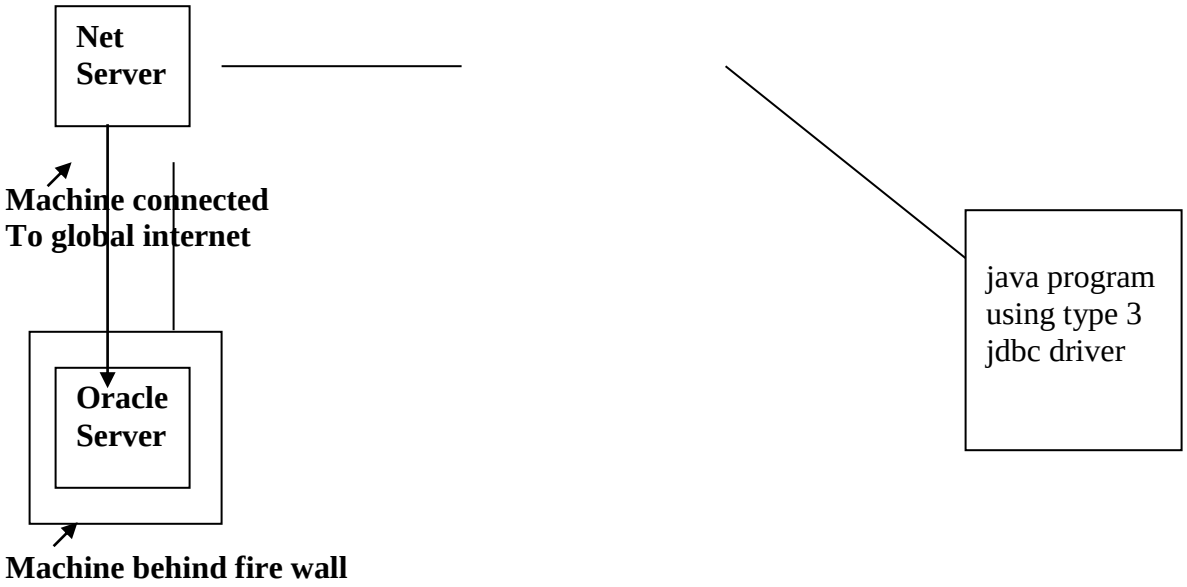


most of the company's run there data base server on the machines that are behind fire wall.

* the vendors of **type 3** drivers will provide a pure java code as part of the JDBC driver & a program called as net server will be provided

* A **Type 3** JDBC driver will not be establishing the connection directly with the data base server. it will communicate with the data base server through by the program called **net server**





* Even though there are various ways of implementing a jdbc Driver the java programmer uses the same code to access the data bases.

```
public class Appli
{
  public static void main(String args[])throws Exception
  {
    Object o = new String("aaaaaaa");
    System.out.println(o.getClass());
     o = new java.util.Vector();
     System.out.println(o.getClass());
  }
}
```

* Connection , statement , ResultSet etc…. are interfaces a jdbc driver vendor provides a set of classes implementing these interfaces.

* When our java program executes DriverManager.getConnection() it internally executes the code that is provided by the JDBC driver vendor & this code is responsible for creating an object based on a class that implements connection interface.

* We can store video/audio/images etc by using **blob (binary long object)** data type & to store huge amount of text we can use **clob (character long object)** data type.

* We can use a PreparedStatement & set the value of **blob** column using , using **stmt.setBinaryStream()** or we can use **stmt.setblob()** .

* **blob** is an interface so we can't create an object.

* Any thing that occurs repeatedly is called as a pattern for Ex : 10101010 is a pattern.

* To simplify the design of a software project most of the developers are using the design pattern.

* A design pattern is a best solution for a reoccurring problem there are so many standard design patterns being use to day the software designers.

* An antipattern is a worst or a bad solution .

* There are so many design patterns.

* **singleton design pattern :**

```
public class GlobalData
{
 /*
  some methods to manage the data used by several   methods of my

  project
 */
  public GlobalData()
  {
   System.out.println("------GlobalData cretaed-----");
  }
}
```

* If we provide a class as shown above we can create **any no.of objects** by writing the code as shown below.

```
import java.util.*;
public class appl
{
 public static void main(String args[])throws Exception
 {
  GlobalData d1= new GlobalData();
  GlobalData d2= new GlobalData();
  GlobalData d3= new GlobalData();

  System.out.println("d1-->"+d1.hashCode());
  System.out.println("d2-->"+d2.hashCode());
  System.out.println("d3-->"+d3.hashCode());
 }
}
```

* As part of object class hash code method is provided , this method is implemented to return a unique number for every object.

* above application answer is

------GlobalData cretaed-----
------GlobalData cretaed-----
------GlobalData cretaed-----
d1-->32124414
d2-->24216257
d3-->20929799

* The above application can create multiple objects based on the GlobalData class.

```
public class GlobalData1
{
  private GlobalData1()
  {
   System.out.println("------GlobalData1 cretaed-----");
  }
}
```

* As we have declared the constructor as a private we can't create an object from the code ie part of other classes.

Ex:

```
  public class GlobalData1
{
 private static GlobalData1 obj = null;
 public static GlobalData1 create()
 {
  if(obj==null)
  {
   System.out.println("object available");
   obj= new GlobalData1();
  }
   else
   {
    System.out.println("object is not available");
   }
    return obj;
 }
   private GlobalData1()
    {
     System.out.println("it is created");
    }
}
```

* A **singleton** is a class which allows us to create only **one object**.

As we show below example

```
 import java.util.*;
public class Applic
{
 public static void main(String args[])throws Exception
 {
  GlobalData1 d1 = GlobalData1.create();
  GlobalData1 d2 = GlobalData1.create();
  GlobalData1 d3 = GlobalData1.create();

  System.out.println("d1-->"+d1.hashCode());
  System.out.println("d2-->"+d2.hashCode());
  System.out.println("d3-->"+d3.hashCode());
 }
}
```

In the above application answer is

object available
it is created
object is not available
object is not available
d1-->32124414
d2-->32124414
d3-->32124414

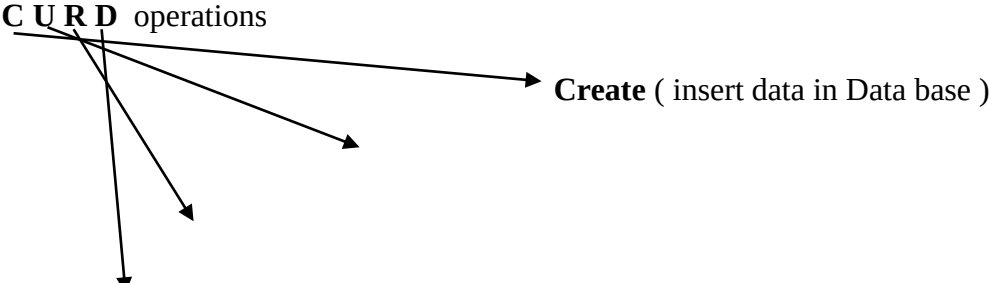
* In the above program when the **create()** method is executed at first time the object will be created . from the next time onwards when we call the **create()** method it will not creates the new object.

* As we have not provided the **clone()** method as part of our class we will not be able to create another object using the **clone()** method .

* In every project we may need to hold some GlobalData , the data that has to be access, as part of various places in the application. If we place this data in a singleton only one copy of data will be maintained .

JNDI

C U R D operations



Update (modify data in Data base)

Read (get data from Data base using select stmt)

Remove (remove data from Data base)

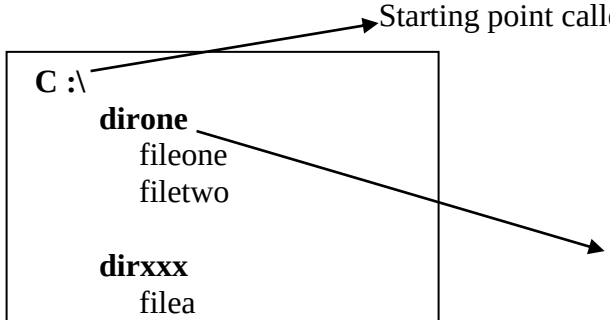
* We use the data base server to manage the data (to perform the operations like creating ,update, read, delete).
* There are several products like
    1. Microsoft active directry server (ADS).
    2. Novell directry server (NDS).
    3. Light weight directry access protocol server (LDAP).
    4. DNS server.
    5. RMI registry.
    6. NIS plus (n/w information server) from sunsoft.

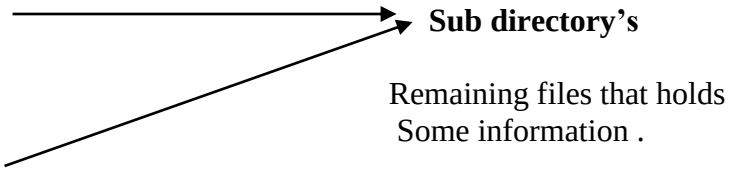
* The above products are called as **Directry Servers** .
* A Directry server is similar to a **Data base** server ,we can use **Directry servers** to manage the data.
* The Directry server are good at searching for the information ie stored in the server & they are good at managing less amount of data.

Data Base Server	Directry Server
1.used to manage data can manage huge amount of data efficiently.	**1.** used to manage data cannot manage huge amount data efficiently.
2. takes more amount of time in searching for information compared to **Dir servers**.	**2**. takes less amount of time in searching Information compared to **data base servers.**
3. insert , update , delete operations can be performed quickly compared **dir servers**.	**3.** these operations takes more amount of Time.
4. support **SQL .**	**4.** doesn't support **SQL.**

* When we need to build the application that has to perform search operations for more member of times than for other operation we can prefer directry server.

* most of the directry severs allows us to organize the data using a tree structure. like how we can organize the files in a file system.

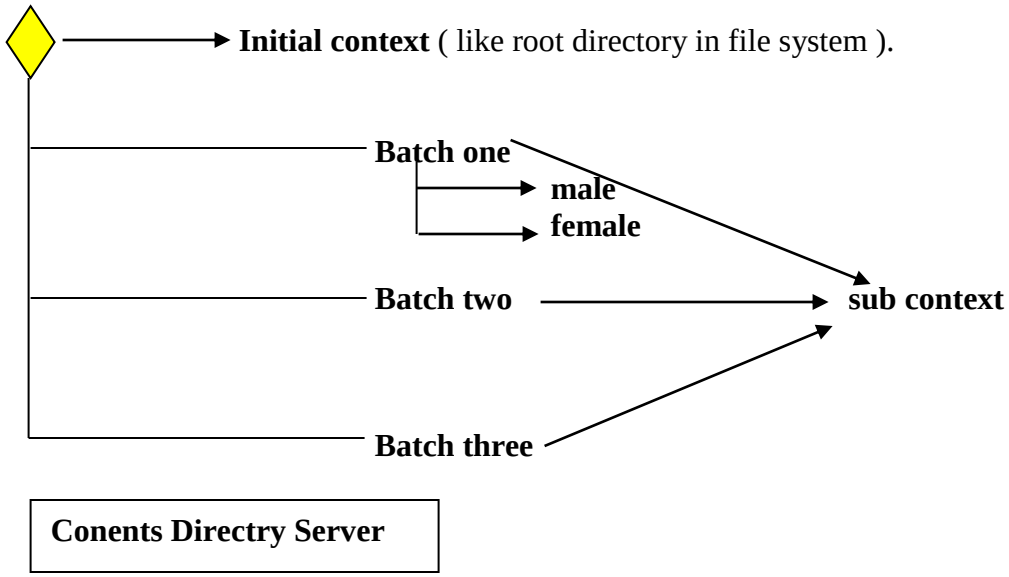




* In the directory server we can organize data in similar way.

* A directory server allows us to organize the information similar to a file system.

* In the case of directory servers we organize the data under various **contexts** (It is like a directory in a file system) .



***** As part of j2se **JNDI (java naming & Directry interface)** API is provided to access the directry servers.

* JDBC API is for accessing data bases.
* JNDI API is for accessing Directory Servers.
* In case of JDBC API to interact with a database we must first establish a connection with the database server (ie we must create a connection object) .

* In case of JNDI first we need to get the reference of initial context object. (we can assume this to be same as establishing a connetion).

* the servers like **web logic , web sphere , Jboss , sun one server** etc provides a directry service.

* **procedure for configuring weblogic server .**

Step 1. start domain configuration wizard.

Start → programs → BEA weblogic → Domain configuration wizard.

Step 2. choose WLS Domain and enter the name of the Domain (Ex: sudhi) & click next button.

Step 3. provide the user name and the password.

* If we give sudhi as the domain name the wizard creates a directory with the name sudhi & places several files under this directory.

After that we open the dos prompt then type following

C:**cd bea;**
C:\bea>**cd user_projects;**
C:\bea\user_projects>**cd sudhi;**
C:\bea\user_projects\sudhi> **startweblogic;**

* **weblogic.server** is a java class.
* weblogic is a java program.
* weblogic , webspeher ,tomcat , Jboss etc are java programs.

* to set classpath:

C:\bea\user_projects\sudhi> **setenv**
C:\bea\user_projects\sudhi> set CLASSPATH=%CLASSPATH%;
C:\bea\user_projects\sudhi>**echo %CLASSPATH%**

In Linux Operating System set Class path is:

setEnv.sh

export CLASSPATH=$ CLASSPATH:.

* To deal with directory server plus we need to get initial context object.
* To get initial context as part of servlets (or) as part of EJB'S running inside weblogic ,websphere etc using .

Context ic = new InitialContext();

* But to get initial context from a standalone java application we need to
1.set up the values for a set of properties.

Note : for this we must refer to docs provided by the vendors.

2.Create an initial context object by passing these properties.

* Most of the java developers uses a standard design pattern called as factory pattern.

* In the factory pattern for creating the object for example dog ,cat, xxxx etc. the developers will provide the factory classes (cat factory , dog factory , xxx factory..).

* To get the initial context in case of weblogic we can write the code as shown below.

```
import javax.naming.*;
import java.rmi.*;
import java.util.*;
public class UseJndi0{
public  static void main(String args[]){
 try{
        Hashtable h = new Hashtable();
         h.put(Context.INITIAL_CONTEXT_FACTORY,
            "weblogic.jndi.WLInitialContextFactory");
          h.put(Context.PROVIDER_URL, "t3://localhost:7001");
          h.put(Context.SECURITY_PRINCIPAL, "sudhi");//user
          h.put(Context.SECURITY_CREDENTIALS, "soujanya");
          Context ic = new InitialContext(h);
          System.out.println(" initial context = "+ ic);
          ic.createSubcontext("batchone");
          //ic.destroySubcontext("batch two");
          Integer i1 = new Integer(20);
          Integer i2 = new Integer(20);
          Integer i3 = new Integer(20);
          ic.bind("sudhi",i1);
          ic.bind("sodhi",i2);
          ic.bind("sudha",i3);
         }catch (Exception e){System.out.println(e);}
   }
}
```

* We can store a serializable java objects in the directory server.

* We can use **context.createSubcontext()** to create a sub context.
ic.createSubcontent() .
* To remove the existing sub context we must use **ic.destroySubcontext()**

* When the above code is executed three integer objects will be stored in the directory server, with names of **sudhi ,sodhi , sudha** .

* **.** and **/** acts as a separator in weblogic and most of the servers used in **/**
* To remove an existing object from the directry server we can use **unbind** method.

```
ic.unbind("sudhi");
ic.unbind("sodhi");
ic.unbind("sudha");
```

```
import javax.naming.*;
import java.rmi.*;
import java.util.*;
  public class UseJndi0{
  public  static void main(String args[]){
      try{
          Hashtable h = new Hashtable();
          h.put(Context.INITIAL_CONTEXT_FACTORY,
             "weblogic.jndi.WLInitialContextFactory");
          h.put(Context.PROVIDER_URL, "t3://localhost:7001");
          h.put(Context.SECURITY_PRINCIPAL, "sudhi");//user
          h.put(Context.SECURITY_CREDENTIALS, "soujanya");
          Context ic = new InitialContext(h);
          System.out.println(" initial context = "+ ic);
          ic.createSubcontext("batchone");
          Integer i1 = new Integer(20);
          Float f1 = new Float(21.22);
          String s1 = new String("wee");
          ic.bind("sudhi",i1);
          ic.bind("alekha",f1);
          ic.bind("sudha",s1);
          ic.bind("sodhi",f1);
        }catch (Exception e){System.out.println(e);}
   }
}
```

* in the above program its compiling but at the run time it gives Exception below
javax.naming.NameAlreadyBoundException: sodhi is already bound; remaining name ''

* If we use **ic.bind()** method by passing a name which is already available name already bound Exception will be thrown.

* Rebind method stores an object if there is no object.

* If there is an object the rebind method replaces the old object with a new object.
Insert → bind
Delete → unbind
Update → rebind
Select → lookup

* To retrive the objects that are stored by the above program we can write the code as shown below.

```
import javax.naming.*;
import java.rmi.*;
import java.util.*;
  public class UseJndi1{
  public  static void main(String args[]){
      try{
```

```
        Hashtable h = new Hashtable();
        h.put(Context.INITIAL_CONTEXT_FACTORY,
           "weblogic.jndi.WLInitialContextFactory");
        h.put(Context.PROVIDER_URL, "t3://localhost:7001");
        h.put(Context.SECURITY_PRINCIPAL, "sudhi");//user
        h.put(Context.SECURITY_CREDENTIALS, "soujanya");
        Context ic = new InitialContext(h);
        System.out.println(" initial context = "+ ic);
        ic.createSubcontext("batchone");
        Integer x;
        Float y;
        String z;
        x=(Integer)ic.lookup("sudhi");
        y=(Float)ic.lookup("alekha");
        z=(String)ic.lookup("sudha");
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
      }catch (Exception e){System.out.println(e);}
   }
}
```

the out put of above program is

20
21.22
wee

* Object java pooling is a technique of pre creating the objects & using these objects at a later point of time.

* Most of the applications pre create the object at startup time (or) when ever the application is not doing any other task, this technique of pre creating the objects improves the performance of an application this technique is used mainly in developing server side applications (application using servlets , EJB technology).

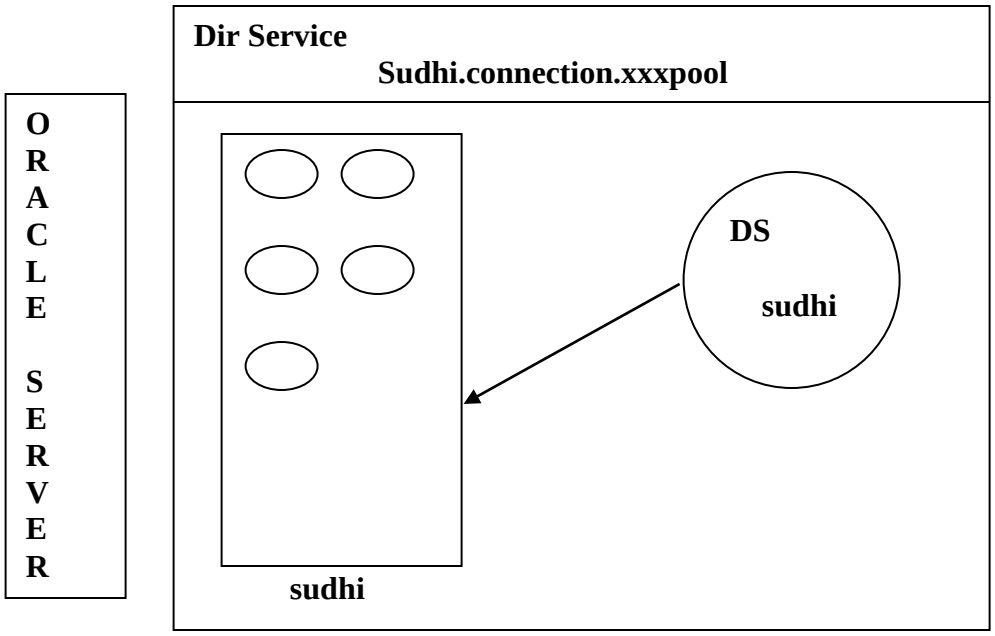
* The servers like weblogic , webspehere support connection pooling technique ie these servers takes care about connecting to the database server & creating the connection object.

* Establishing a connection takes more amount of time instead of establishing the connection when ever the connection is required , the application can take connection Object from the connection pool & place back the connection in the pool so that it can be reused .

* An administrators are responsible for sitting up a connection pool.

* As part of weblogic server there is Dir service.

* Procedure for creating connection pool in weblogic server.



Weblogic server

Step 1: logon to weblogic console
It provide the user name & password to logon.

Step 2: Choose **services** in that Choose **JDBC** after that select **Connection pools**.
Service → JDBC → Connection Pools

Step 3: Click on the link **Configure a new JDBC Connection pool** .

After that it displays new window in that it shows below
Name : _____ (sudhi)
URL :_____ (jdbc:oracle:thin:@localhost:1521:orcl)
DriverClassName :_____ (oracle.jdbc.driver.OracleDriver)
Properties :_____ (user = scott password = tiger)

and finally to click on **apply** button.

Step 4: Choose the **Connections tab** and provide a value for **initial capacity** & **maximum capacity** and click on **apply** button.

* After we click on the apply button the weblogic server while create a pool of 5 Connections if they are not enough at a later point of time it will increasing connections up to 10.

* In order to use the Connection that are available in the Connection pool we must create **javax.sql.DataSource**.

* To Create a Data Source Object

Step 1: Choose **Services** in that select **JDBC** after that select **Tx Data Sources**.

Service → JDBC → Tx Data Sources

Step 2: Click on Configure a new JDBC Tx Data Source... link after that it displays below options.

Name :_____ (sudhi)
JNDI Name:_____ (sudhi.connection.sudhapool)
Pool Name:_____ (sudhi)

Finally click on **create** button.

Step 3: Choose **target option** available then click apply button.

* The above step is responsible for Creating Data Source Object & store information about this Data Source Object in the directory server.

* Procedure to get a Connection from the Connection pool.

Step 1: Get the reference of initial context **(ic)**.

```
Context ic = new InitialContext(h);
```

Step 2: get the data source object by using **ic.lookup();**

```
DataSource ds = ( DataSource )ic.lookup("sudhi.connection.sudhapool");
```

Step 3: To get the connection from connection pool execute **ds.getConnection();**

```
Connection c1 = ds.getConnection();
```

* To return a connection to the connection pool we must use connection.close method.

```
c1.close();
```

SERVLETS

Plz visit www.theserverside.com to know about servers now available in the market.

* A web server is a product that implements HTTP There are so many web servers available in the market , the most popular web servers are **Apache webserver , Microsoft IIS (internet information server)** these two products can't be used to run the servlets.

* According to java soft a product that supports servlets can be called as web container.
Ex : Tomcat , weblogic , sun one server etc provides the implementation of HTTP protocol & these servers support servlets so these servers can be called as web containers.

* weblogic is an application server.

* On the top of web container we can place a HTML file, XML file , IMAGE file , AUDIO file VIDEO file , SERVLET , PERL script etc are called as web resources.

* When we install a webserver we can set the port no, that has to be used by the web server after installing also we can modify the port no.

Note: most of the company's prefer to use port no **80** for the webservers.

E:\ my website under I placed my files & resources **one.html** , **two.html** & under that we create a directory **done** & under that we provide **three.html** , **four.html** files .

* A web application is a collection of web resources.
E:\ my website is called as **DOCUMENT_ROOT**.

* Microsoft document refers to the document root has the home directory of a web application.

* When ever a request is sent by the webserver it checks for the resources under DOCUMENT_ROOT.

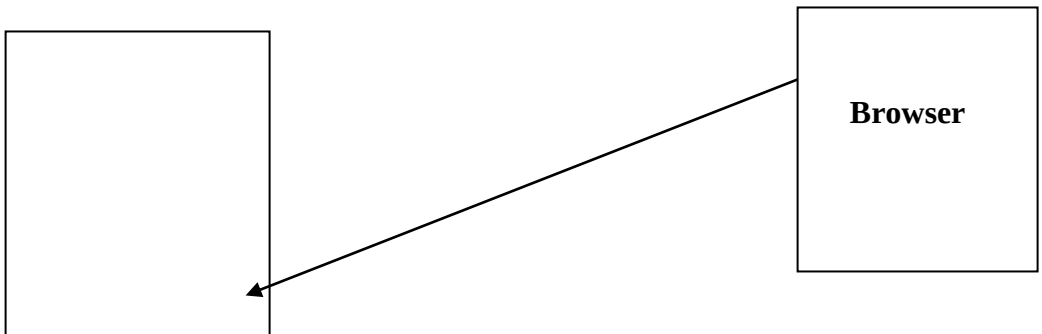
* URL the browser will gets the **IP address & Port no**.

* When we type **http://www.hotmail.com:800/one.html**

Step 1: The browser gets the IP address of ww.hotmail.com

Step 2: Using the IP address & port no 800 the browser establishes the connection with the web server.

Step 3: The browser sends a request to the webserver asking for the resource **/one.html**.



www.hotmail.com

/one.html is in the above example / is the path & it represents the DOCUMENT_ROOT & one.html is resources.

Step 4: The web server checks for one.html in the document root (**E:\ my website**) if the file is available it will be send to the browser, if the file is not available the webserver sends an error message to the browser.

* If we type the URL without specifying a resources like **http://www.hotmail.com/800/** the browser will send a request asking for / .

* The administrator can setup the default documents which will be served to the browser if the resource is not specified.

* If index.html & def.html are defined as default document the webserver checks for the file **index.html** if it is not there it checks for **def.html** , if one of these files are there, the web server serves that find the server.

- If none of the default documents are available the webserver may
- **1.** send an error message to the browser (or)
- **2.** it sends the contents of the directory to the browser .

* Generally directory browsing will be disable.

* When we build a web application using servlet technology as part of the web application we will provide several web resources, some of these web resources are static resources (HTML file, image file , audio file,etc) & dynamic resources(like **servlets / jsp** etc).

* According to servlets specification a web application is a collection of web resources.
Bea → weblogic700 → server → lib → weblogic.jar

* When we develop servlets we will use the packages
javax.servlet
javax.servlet.http

* Every webcontainer vendor provides a set of **jar** files as part of one of the **jar** files they provide the classes & interfaces that are part of the above two packages.

Ex: 1. In case of weblogic these packages are provided as part of **weblogic.jar**

2. In case of tomcat we get these classes as part of **servlet-api.jar**
3. To compile the servlets that are developed by us ,we need to set **CLASSPATH** pointing to these files.

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class FirstServlet extends HttpServlet
{
 public void service (HttpServletRequest req,
                      HttpServletResponse resp)throws ServletException, IOException
  {
   java.io.PrintWriter out = resp.getWriter();
   out.println("<html>");
   out.println("<head>");
   out.println("<title> Welcome !!</title>");
   out.println("</head>");
   out.println("<body>");
   Date now = new Date();
   out.println("Date & Time --> " + now);
   out.println("some content here.....");
   out.println("</body>");
   out.println("</html>");
  }
}
```

* Placing a web application inside a web container is called as deploying a web application in order to deploy first we need to create a WAR file, following the procedure given below.

* **WAR → means Web Archive**

Step 1: Create a folder any name Ex: sudhi **(DOCUMENT ROOT)**
Ex: create a directory
D:\sudhi

Step 2: Under **DOCUMENT ROOT** create a directry with the name **WEB-INF**.
D:\sudhi\WEB-INF

Step 3: Under **WEB-INF** create the Directory's
a) lib
b) classes

Step 4: Copy the servlet classes under classes directory.
Means above servlet program class file paste in under classes directory.

Step 5: Copy the html files etc under the document root (or) a sub directory of DOCUMENT ROOT.
Note : these files should not be copy under **WEB-INF**.

Step 6: Place **web.xml** under **WEB-INF** directory.
ie it locates in **tomcat 5.0 → webapps →tomcat-docs → WEB-INF → web.xml**.

sample xml file we can make changes to it.

Modify in the as shown below Ex:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!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>My First web</display-name>
  <description>
    My first servlet program  </description>
 <servlet>
 <servlet-name>fs</servlet-name>
 <servlet-class>FirstServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>fs</servlet-name>
 <url-pattern>/one</url-pattern>
 </servlet-mapping>
</web-app>
```

Step 7: Move to the DOCUMENT_ROOT
Ex: D:\sudhi >

Issue the following command

D:\sudhi > jar cvf sudhi.war .

Need not be same name it can be any name.

Create verbose file.

* The above steps are same irrespective of the web container we use.

* **Procedure for deploying the web application on tomcat.**

1. Copied the WAR file under **tomcat/webapps**.
Tomcat → webapps

* Like this for various web containers different directories will be provided.

* **Procedure for deploying the web application on Weblogic**.

1. Copied the WAR file under under **bea/user_projcts/sudhi/applications**.

Bea → user_projcts → sudhi → applications

* If run the application in weblogic then the link is

http://localhost:7001/sudhi/one

* if run the application in tomcat then the link is

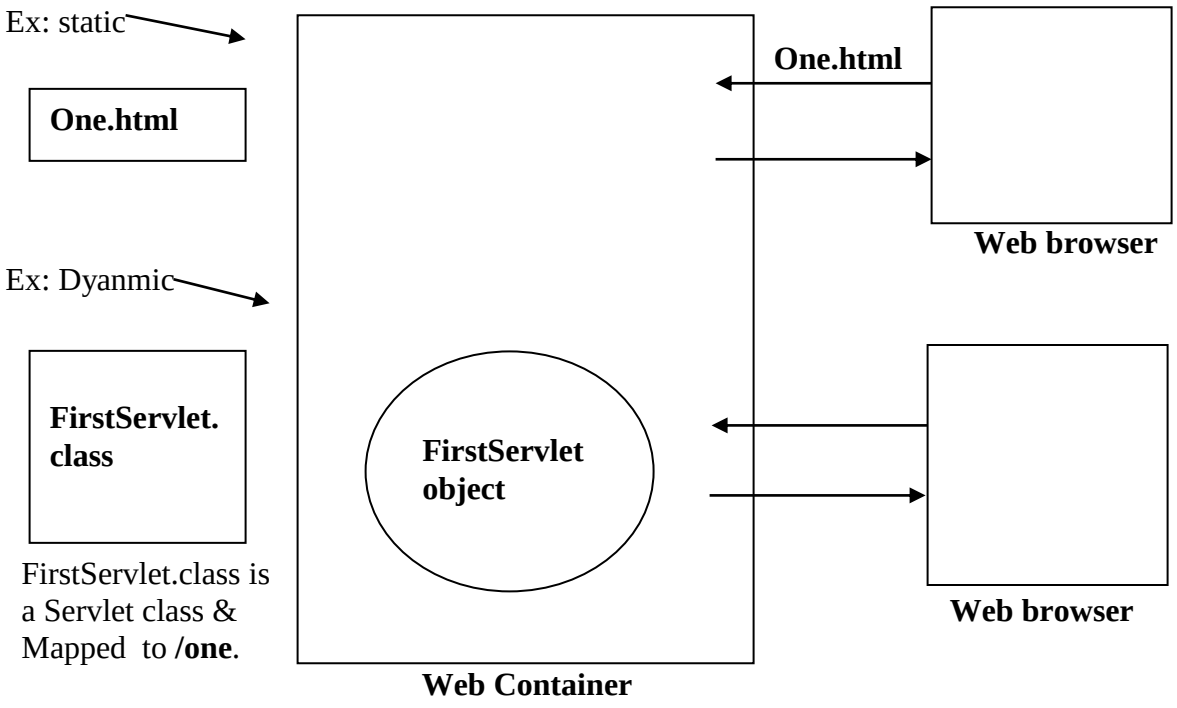
http://localhost:8080/sudhi/one

* **out.println()** that is part of the servlets code sends the content to the Browser.

* Most of the developers provides the WAR files to the customers but during development instead of repeatedly creating the WAR files , the developers will directly copy the DOCUMENT_ROOT to the **webapps** directory of TOMCAT & **applications** directory of weblogic similar kind of directory's of other servers.

* out.println() is going to the print in browser.

* System.out.println() is going to the print in the server.



* When we send a request to **one.html** the web container picks up the content from the files & sends that content to the browser, to the content this is called as a **static** resources as we get the same content irrespective of the no.of requests we send to the server for this resources.

* When we send the request for the servlet the web container if require a java object based on the servlet class will be created . and the code in the service method will be executed, the output generated by this code will be send to the web browser.

* In this case the content is generated **dynamically** by executing the code in the servlet this is why we call the servlet as a dynamic resource.
* It is not advisable to use the servlets for generating the **same content** every time, when a request is sent for this we must use static resources.

* Most of the modern web application generates the content dynamically.

* Most of the company's are using databases to store the information.

1. If we need to provide a web application to display the list of employ's we can use a HTML file but this file must be modified manually, when ever an employee leaves the company are joins the company.
2. For this we can develop a servlet which (**a).**gets a data from the database **(b).** generates the content using this data.

* If two programs has to communicate with each other both the programs must use the same set of rules (a set of rules is nothing but a protocol) .

* **HTTP** is a public (open) protocol in the webserver & the web browser communicates with each other using HTTP protocol.

* According to HTTP protocol the User-Agent must first establish the connection with the server.

* When we type the any links in internet explorer then it first check the **IP Address** & **Host Names** these are stored in **DNS Server**.

* When we type the any links in internet explorer then it first check the IP Address of that particular IP Address in the DNS Server.

* IP Address & Host Names are stored in DNS server.

* Host Name means machine name or any string.

Ex:

```
import java.net.*;
import java.io.*;
public class link
{
```

```
 public static void main(String args[])throws Exception
 {
  InetAddress ia = InetAddress.getByName("localhost");
  System.out.println(ia);
 }
}
```

* **getByName** method returns **InetAddress** object address if it can find the IP Address of the Host name whose name is passed as a parameter, if can't find the IP Address then Exception will be thrown.

* **In order to establish the connection we must use**

Step 1:
1. IP Address
2. The port no.of the server

* using these two we must create a socket object that represent the connection.

```
InetAddress ia = InetAddress.getByName("localhost");  ------------> 1
System.out.println(ia);
System.out.println("connecting the server....");
Socket con = new Socket(ia,7001);  ------------------------------> 2
System.out.println("connected the server.....");
```

* When the above line is executed, it tries to connect to the server if it can't connect to the server it will throw an Exception, if it connect to the server an object of type socket will be created.

Note : A java program (or) assume that the socket is an object that represents the connection.

* After establishing the connection a java application can send the data to the other programs & receive the data sent by the other program for this we must use two objects.

1. **SocketOutputStream**
2. **SocketInputStream**

```
 InputStream   sis = con.getInputStream();
 OutputStream  sos = con.getOutputStream();
  byte inp[] = new inp[200];
  sis.read(inp,0,200);
  String s = new String(inp,0,200);
  System.out.println(s);
  PrintStream ps = new PrintStream(sos);
  ps.print();
```

* To send the data to the other program we can use **sos.write()** method or we can use **ps.print()**.

Step 2:

* After the connection is establish the user agent must send a **request**.
* According to the HTTP protocol the user agent must use the following

* According to the HTTP protocol the request can have any no.of lines followed by a blank lines according to a protocol, a line must be terminated using **\r\n**.

System.out.print("line one \r\n"); ⟶ **1**
System.out.print("line one \r\n\r\n"); ⟶ **2**
System.out.print("line one \r\n"); ⟶ **3**

* In the above lines output is

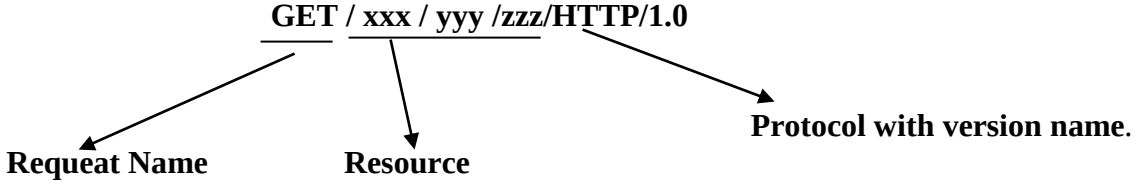
line one
line one
---------- ⟶ this one is 3 that's why it is empty space.
line one

* The first line of the requeat is called as initial request line ,of after this line the remining lines are called as head lines.

REQUEST – FORMAT

Iniatial RequestLine → GET/xxx/yyy/zzz/HTTP/1.0
n no.of Headerlines (user-agent): → our browser.
Blank line
Body (content) [OPTIONAL]

* The request line is split into three parts.



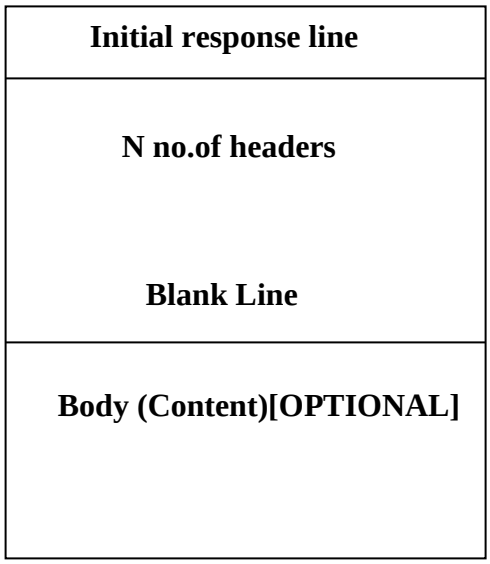
* The Header lines are split in to two parts as shown below
    1.Header name.
    2.Header value.

```
ps.print("GET / xxx / yyy /zzz/HTTP/1.0\r\n");
ps.print("User-Agent:inetsolve Browser 1.0\r\n");
ps.print("\r\n");
```

Step 3: The server process the request.

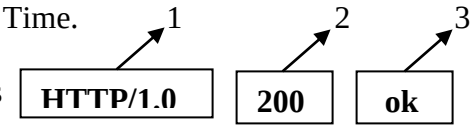
Step 4: The server sends the response using the format ie very mush similar to the request format.

* In case of response the format specify us as shown below



* **GMT** → Green Mitch Time.

* The Initial Response is



Initial response is spilt in to **three** portions.

1. The protocol & the version number is **HTTP / 1.0**
2. Status code is **200**
3. Status message is **ok**.

* According to HTTP protocol the server must use the following status codes as part of the response.

1. 1XX means 100,101,102 etc.

To indicating sum set of information.

2. 2XX

To indicate success.

3. 3XX

To indicates that the browser must send a request to some other resource (the redirection message).

4. 4XX

To indicate an error due to the problem with client.

5. 5XX

To indicate failure due to the problem with server.

* There are so many HTTP request methods

1. **GET:-**

When the client sends GET request method the server will return the headers as well as the body. Get → get the resource.

In the show below example

```
import java.net.*;
import java.io.*;
public class link1
{
 public static void main(String args[])throws IOException
 {
  InetAddress ia = InetAddress.getByName("localhost");
  System.out.println(ia);
  Socket con = new Socket(ia,7001);
  InputStream sis = con.getInputStream();
  OutputStream sos = con.getOutputStream();
  PrintStream ps = new PrintStream(sos);
  ps.print("GET /sudha/one.html HTTP/1.0\r\n");
  ps.print("User-Agent:InternetExplorer6.0 \r\n");
  ps.print ("\r\n");
  byte[] b=new byte[1000];
  sis.read(b,0,1000);
  String s = new String(b,0,1000);
  System.out.println(s);
 }
}
```

The output is =

```
 HTTP/1.0 200 OK
Date: Fri, 26 Aug 2005 10:23:29 GMT
Server: WebLogic WebLogic Server 7.0  Thu Jun 20 11:47:11 PDT 2002 190955
```

Content-Length: 159
Content-Type: text/html
Last-Modified: Tue, 23 Aug 2005 16:10:34 GMT
Connection: Close

```
<html>
<head>
<title> Welcome !! </title>
</head>
<body>
      Date : 22-aug-2005 <br>
      Time : 8.00pm. <br>
      Some content there.....
</body>
</html>
```

In the above example in the content we have one.html so GET method executed and it will get the content of one.html in command prompt.

2. **HEAD :-**

When the client uses HEAD request method , the server will return only the headers.

As shown in the below example

ps.print("HEAD /sudha/one.html HTTP/1.0\r\n"); remaining same as above program

the output is only header will display

HTTP/1.0 200 OK
Date: Fri, 26 Aug 2005 10:24:56 GMT
Server: WebLogic WebLogic Server 7.0 Thu Jun 20 11:47:11 PDT 2002 190955
Content-Length: 159
Content-Type: text/html
Last-Modified: Tue, 23 Aug 2005 16:10:34 GMT
Connection: Close

3. **DELETE :-**

The client can use DELETE request method to ask the server to delete the resource. By default DELETE will not be allowed we can change the settings.

IIS Server Default website → Properties → Home directory we can change settings here.

As shown in the below program

ps.print("DELETE /sudha/one.html HTTP/1.0\r\n");

remaining same as above program

the output is shown in the below :

HTTP/1.0 400 Bad Request
Date: Fri, 26 Aug 2005 10:28:17 GMT
Server: WebLogic WebLogic Server 7.0 Thu Jun 20 11:47:11 PDT 2002 190955
Content-Length: 47
Content-Type: text/html
Connection: Close

Http method DELETE is not supported by this URL

4. **PUT :-**

PUT request method can be used by the clients to place a resource on server will not be allowed.

As shown in the below program

ps.print("PUT /sudha/one.html HTTP/1.0\r\n");

remaining same as above program

the output is shown in the below :

HTTP/1.0 400 Bad Request
Date: Fri, 26 Aug 2005 10:30:46 GMT
Server: WebLogic WebLogic Server 7.0 Thu Jun 20 11:47:11 PDT 2002 190955
Content-Length: 44
Content-Type: text/html
Connection: Close

HTTP method PUT is not supported by this URL

5. **POST** :-

This request method is similar to GET ,but as part of POST method some content will be sent to server.

It will display in the same as GET method result.

6. **TRACE** :-

It can be used by client to diagnose the problems of a server , what ever sent by browser will be resend the body of request, that indicates the ur request is working (or) ok.

As shown in the below program

ps.print("TRACE /sudha/one.html HTTP/1.0\r\n");

remaining same as above program

the output is shown in the below :

HTTP/1.0 200 OK
Date: Fri, 26 Aug 2005 10:35:24 GMT
Server: WebLogic WebLogic Server 7.0 Thu Jun 20 11:47:11 PDT 2002 190955
Content-Length: 66
Content-Type: message/http
Connection: Close

TRACE /sudha/one.html HTTP/1.0
User-Agent: InternetExplorer6.0

7. **OPTIONS** :-

Options can be used by client that can be used on a resource.

Even though the server can support a no of request methods , the browsers will be sending the requests using 1.**GET** 2.**POST** 3.**HEAD**

As shown in the below program

ps.print("OPTIONS /sudha/one.html HTTP/1.0\r\n");
remaining same as above program

the output is shown in the below :

HTTP/1.0 200 OK
Date: Fri, 26 Aug 2005 10:38:47 GMT
Server: WebLogic WebLogic Server 7.0 Thu Jun 20 11:47:11 PDT 2002 190955
Allow: GET, HEAD, OPTIONS, POST
Content-Length: 0
Connection: Close

* As part of servlet API various interfaces like HttpServletRequest , HttpServletResponce , HttpSession , FilterConfig , FilterChain , RequestDispatcher , ServletConfig , ServletContext etc.

* It is the responsibility of the container vendors to provide the classes implementing these interfaces.

1. HttpServletRequest Provides the methods to deal with the request that is send by the client to the server.

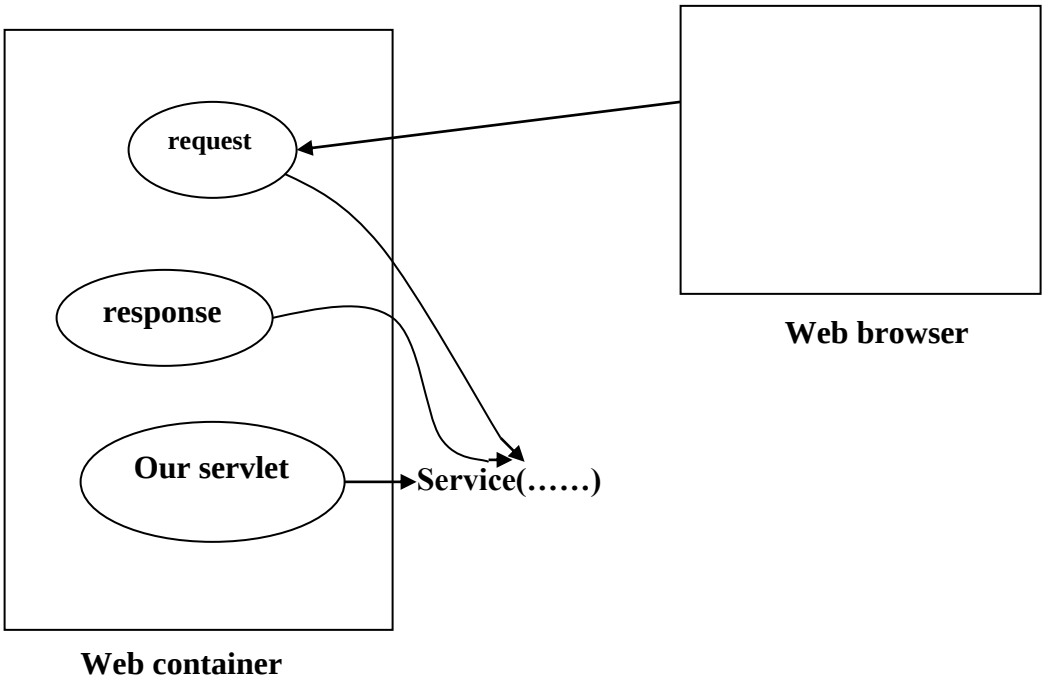
2. HttpServletResponce has methods which can be used to deal with the response ie sent by the server to the client.

Note : As a java developer we can view the request ie sent by the client & the response that is sent by the server as an **object** .

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet3 extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
     System.out.println("------------------------------");
     System.out.println("request---->"+request.getClass());
     System.out.println("response--->"+response.getClass());
    System.out.println("------------------------------");

  }
}
```

* In that above example answer in server is

```
------------------------------
request---->class weblogic.servlet.internal.ServletRequestImpl
response--->class weblogic.servlet.internal.ServletResponseImpl
------------------------------
```



* Our servlet program is not responsible for create request & response interface, tomcat or any web container will take care about that interface.

* After service method will be executed then request & response objects are destroyed.

* There are so many methods available as part of these two interfaces.

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet3 extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   System.out.println("------------------------------");
   System.out.println("request--->"+request.getClass());
   System.out.println("response--->"+responce.getClass());
   System.out.println("method---->"+request.getMethod());
   System.out.println("request URI--->"+request.getRequestURI());
   System.out.println("protocol--->"+request.getProtocol());
   System.out.println("Browser Name---->"+request.getHeader("User-Agent"));
   System.out.println("------------------------------");   }
  }
}
```

* In that above example answer in weblogic server is

```
------------------------------
request--->class weblogic.servlet.internal.ServletRequestImpl
response--->class weblogic.servlet.internal.ServletResponseImpl
method---->GET
request URI--->/soudhi/one
protocol--->HTTP/1.1
Browser Name---->Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
------------------------------
```

* and above same answer in tomcat server is

```
------------------------------
request--->class org.apache.coyote.tomcat4.CoyoteRequestFacade
response--->class org.apache.coyote.tomcat4.CoyoteResponseFacade
method---->GET
request URI--->/soudhi/one
protocol--->HTTP/1.1
Browser Name---->Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
------------------------------
```

* **response.setStatus(400,"failed due to set status");**

* The above statement sets
    1. The status code.
    2. The status message that has to be sent to the client.

* In order to send the error message we can use
    response.sendError(500,"failed due some errors ")

* Send error method used only **400 & 500** messages only.

response.setHeader("Company name","Sudarshan Reddy");

* the above command run in **Weblogic server** that is displays in the Browser Page Cannot displayed & shows on title Bar is **HTTP 500 Internall Server error**

***** And it runs in Tomcat server that is displayed in Browser **HTTP 500 error**

* response.getWriter is a method which returns PrintWriter object in the below.

```
PrintWriter out = response.getWriter();
out.println("<html>");
out.println("<head>");
```

* A servlet can generate any type of content so it is advisable to use the method setContentType as shown below.

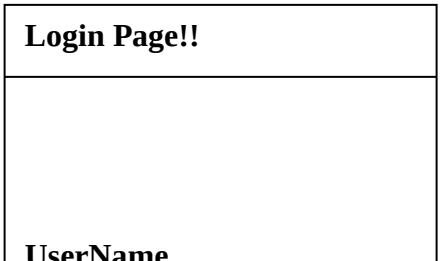
```
Response.setContentType("text/html");
```

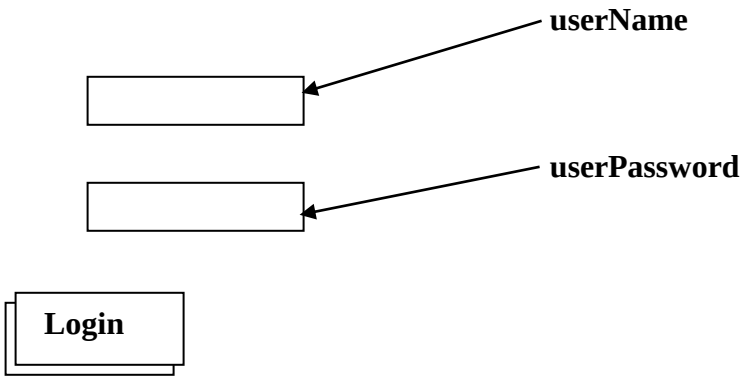
* response.getWriter() method returns a print writer object ie associated with the response when ever our servlet uses the methods like out.print(),out.println() the output will be sent to the client as part of the body of the response.

```
*  Servlet  config = getServletConfig();
   ServletContext  application = config.getServletConfig();
   System.out.println("name-----"+config.getServletName());
   System.out.println("Container name------"+application.getServletInfo());
```

* The web container vendor is responsible for providing the classes implementing servletconfig & servletcontext, it is the responsible of the container to create the object based on these classes.

* As part of the web applications we need to take various inputs from the users for taking the inputs we can use by html forms with different types.






```
 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Sudarshan Login Page </title>
  </head>

  <body bgcolor=yellow>
    <font face="helvetica">

    <form action="url/of/srv" method=GET>
      User Name <input type=text name=userName> <BR>
      Password <input type=password name=userPassword>
                  <br>
<input type=submit  value = submit>
    </form>
    </font>
  </body>
</html>
```

Note : In the above html form there are three input fields only the names.

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet4 extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   String vuserName,vuserPassword;
   vuserName=request.getParameter("userName");
   vuserPassword=request.getParameter("userPassword");
   System.out.println("vuserName--->"+vuserName);
```

```
  System.out.println("vuserPassword--->"+vuserPassword);
 }
}
```

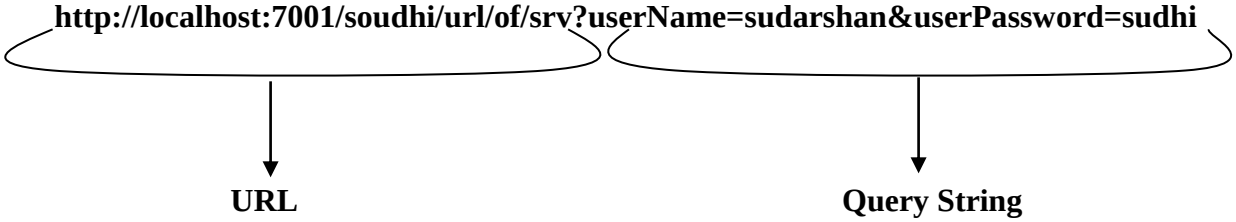
* In **web.xml** file in url-pattern we have to give **/url/of/srv**

* Most of the people developing the applications using html forms prefers not to provide a name to the submit button.

* When we specify **method=POST** as part of the form tag the browser fill send the request using HTTP POST request method

1. If we use GET as a value of method attribute in the request using **HTTP GET** method.

* If we use request method has get the browser picks up the data and appends the information about fields in the form to the URL as a query string & then the browser will send the **getRequest** method to the server.



* The data will be available as part of initial request line if we use **method=GET** .

* The Browser will not send the information about the field to the server if the name of the field is not provided.

* When we use **method=POST** the browser picks up the data entered into the user & place the data in the body not as a query String in URL.

* There is a limitation on the length of the URL ie

* As the data is append to the URL incase of **method =GET** we will not be able send huge amount of data using getRequest method.

* If we have to pass less amount of data from the browser to the server & if there is no problem if the data is displayed in the address bar of the browser it is better to use **GET** as this improves the performance slightly changed.

* If we add validations in the above program then the program is

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet4 extends HttpServlet
```

```
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   String vuserName, vuserPassword;

   // Code to capture the data....

   vuserName=request.getParameter("userName");
   vuserPassword=request.getParameter("userPassword");

   // end of Code to capture the data....

   //Code for debugging......

   System.out.println("vuserName--->"+vuserName);
   System.out.println("vuserPassword--->"+vuserPassword);

   //Code for debugging......

   PrintWriter out=response.getWriter();

    out.println("vuserName--->"+vuserName);
    out.println("vuserPassword--->"+vuserPassword);

   // Code to Validate data (this depend on Project Requirment)..

   if(vuserName==null || vuserName.equals(""))
    {
     out.println("INVALID : user name is Mandatery ");
    }
   if(vuserPassword==null || vuserPassword.equals(""))
    {
     out.println("INVALID : user Password is Mandatery ");
    }

   //end of Code to Validate data (this depend on Project Requirment)..

  }
}
```

* out put of the above program is if we enter the user name & password in the browser then it will successfully executed & it will display in the browser and server side is

vuserName--->Srinivas Reddy
vuserPassword--->srinivas

Ex:

```
public class test
{
 public static void main(String args[])throws Exception
 {
  String s1=null;
  String s2="some value";
  String s3="     ";
  System.out.println("s1..."+s1);
  System.out.println("s2..."+s2);
  System.out.println("s3..."+s3+"....");
 }
}
```

* **s1** is not referring any object here **s2** is referring to an object which is of type String with some value.
* **s1** is a variable pointing to **no object** .
* **s2** is a variable pointing to a string object with a value of some value
* **s3** is a variable pointing to a String object with a value blank string.

Ex:

```
    public class test1
{
 public static void main(String args[])throws Exception
 {
  String s1="123";
 // code to check whether it is an integer.....
  try
  {
   java.lang.Integer.parseInt(s1);
   System.out.println("this is an int");
  }
   catch(Exception e)
   {
    System.out.println("this is not an integer");
   }
  }
}
```

* When parseInt is executed it tries to convert the String into an integer if it can't convert the method throws an Exception.
* If the catch block is not executed we can say it is an integer.
* If the catch block is executed we can say it is not a integer.
* Similarly we can use Float.parseFloat, Double.parseDouble etc..
* If we have a form with the fields
    1. username
    2.userPassword.

and as part of the servlet if we use **request.getParameter("userName");**
request.getParameter("userPassword");

if that is getParameter method returns a blank String, if the user is not provided by the values for userName & userPassword fields.

* **request.getParameter()** method returns a **null** if we try to get a field which is not there in the form.
* We can type a URL pointing to a servlet by adding a query String on our own for example we can type

http://localhost:8080/srinivas/url/of/srv?userName=one&userPassword=two

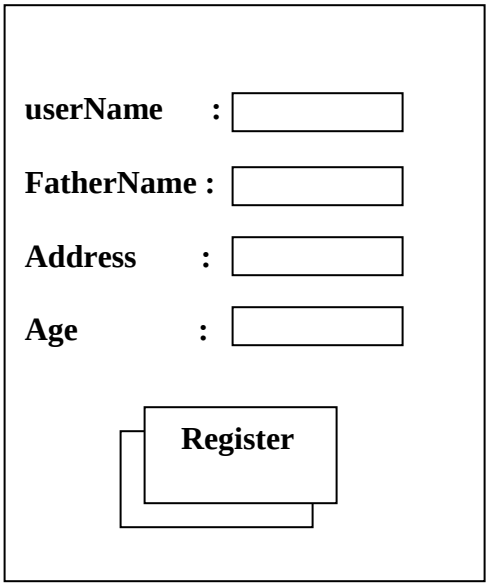
* In this case if the servlet executes **request.getParameter** method username it will get **one** & user password will get **two**.
* The browser uses URL encoding for passing the data according to data URL encoding some characters will not be send as it is.

For Ex: if the user types **space** then it encoding to **+** symbol will send.
if the user types **+** then it encoding to **%2B** symbol will send.
if the user types **&** then it encoding to **%26** symbol will send.
if the user types **%** then it encoding to **%25** symbol will send.

* In every application after taking inputs the application must checks whether the inputs valid or not .

For Ex: In the above form not entering any thing if the username field and userPassword field can be considered as invalid.

Ex: Take a Validation project




```
<html>
 <head>
  <title>Sudarshan Login Page </title>
 </head>
 <body bgcolor= gangadher>
  <font face="helvetica">
  <pre>
  <form action="url/of/srv" method=GET>
    User Name   <input type=text name=userName> <br>
```

```
    Father Name <input type=text name=fatherName> <br>
    Address     <input type=text name=address> <br>
    Age         <input type=text name=age> <br>
                <input type=submit  value = register>
  </form>
 </pre>
 </font>
</body>
</html>
```

Validation Rules :

1.User Name, Father Name, Addrress age are mandatory(required).
2. age must be an integer.
3. age must be below 0 and 150.

* During the design the validation rules will be decided we must provide the code according to the validation rules.

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ValidationServlet2 extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   String vuserName,vfatherName,vaddress,vsage;
   int viage;

   // Code to capture the data....

   vuserName=request.getParameter("userName");
   vfatherName=request.getParameter("fatherName");
   vaddress=request.getParameter("address");
   vsage=request.getParameter("age");

   // end of Code to capture the data....

   //Code for debugging.....

   System.out.println("vuserName--->"+vuserName);
   System.out.println("vfatherName--->"+vfatherName);
   System.out.println("vaddress--->"+vaddress);
   System.out.println("vsage--->"+vsage);

  //Code for debugging.....

   PrintWriter out=response.getWriter();
```

```
  out.println("vuserName--->"+vuserName);
  out.println("vfatherName--->"+vfatherName);
  out.println("vaddress--->"+vaddress);
  out.println("vsage--->"+vsage);

 // Code to Validate data (this depend on Project Requirment)..

  if(vuserName==null || vuserName.equals(""))
   {
    out.println("INVALID : user name is Mandatery ");
   }
  if(vfatherName==null || vfatherName.equals(""))
   {
    out.println("INVALID : father name is Mandatery ");
   }
  if(vaddress==null || vaddress.equals(""))
   {
    out.println("INVALID : address is Mandatery ");
   }
   if(vsage==null || vsage.equals(""))
   {
    out.println("INVALID : vage is Mandatery ");
   }
   else
   {
    try
    {
     viage=Integer.parseInt(vsage);
     out.println("ur data is sucessfull insert into data base");
     System.out.println("ur data is sucessfull insert into data base");
      if(viage>150 || viage<0)
      {
       out.println("INVALID : age must be below 0 to 150 ");
      }
   }

    catch(Exception e)
    {
     out.println("ur data is invalid");
     System.out.println("ur data is invalid");

   }

   //end of Code to Validate data (this depend on Project Requirment)..
  }
 }
}
```

The output of the above program in Browser & server is

vuserName--->Srinivas Reddy.
vfatherName--->Raghuma Reddy.
vaddress--->Hyderabad.
vsage--->24
ur data is sucessfull insert into data base

Ex :

```
 import java.util.Vector;
public class utilvec
{
 public static void main(String args[]) throws Exception
 {
  Vector errs = new Vector();
  System.out.println("Size--->"+errs.size());
  errs.add("Error msg one");
  errs.add("Error msg two");
  System.out.println("1---Size--->"+errs.size());
  errs.add("Error msg three");
  System.out.println(errs);
 }
}
```

The output is

Size--->0
1---Size--->2
[Error msg one, Error msg two, Error msg three]

* In the above example **errs.size()** returns **0** indicating that there are no objects in the Vector when we call it for the first time then second time it returns **2** indicating there are two objects in the vector.

Ex :

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ValidationServlet3 extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   String vuserName,vfatherName,vaddress,vsage;
   int viage;

    //Vector to store validation errors.....
```

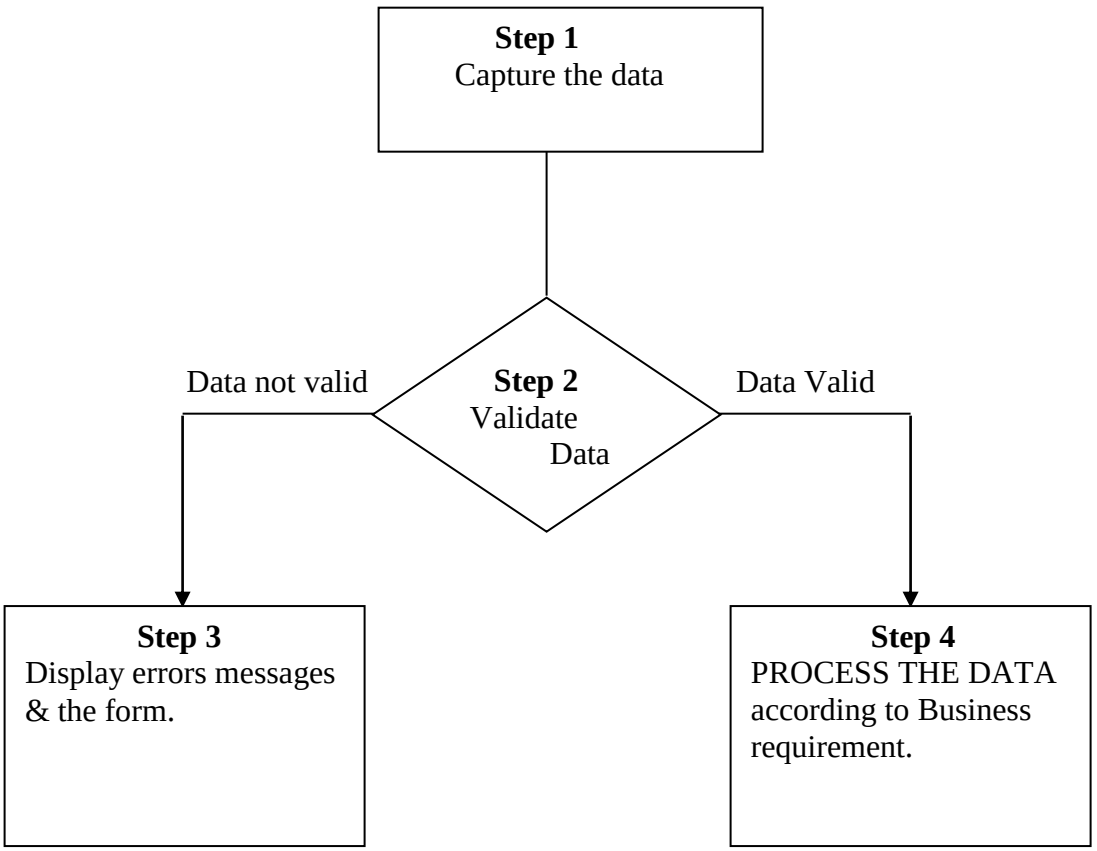
```
  Vector errs = new Vector();

 // Code to capture the data....

 vuserName=request.getParameter("userName");
 vfatherName=request.getParameter("fatherName");
 vaddress=request.getParameter("address");
 vsage=request.getParameter("age");

 // end of Code to capture the data....

 //Code for debugging.....

 System.out.println("vuserName--->"+vuserName);
 System.out.println("vfatherName--->"+vfatherName);
 System.out.println("vaddress--->"+vaddress);
 System.out.println("vsage--->"+vsage);

//Code for debugging.....

 PrintWriter out=response.getWriter();

 out.println("vuserName--->"+vuserName);
 out.println("vfatherName--->"+vfatherName);
 out.println("vaddress--->"+vaddress);
 out.println("vsage--->"+vsage);

// Code to Validate data (this depend on Project Requirment)..

 if(vuserName==null || vuserName.equals(""))
  {
  errs.add("<br>INVALID : user name is Mandatery<br>");
  }
 if(vfatherName==null || vfatherName.equals(""))
  {
   errs.add("<br>INVALID : father name is Mandatery<br>");
  }
 if(vaddress==null || vaddress.equals(""))
  {
   errs.add("<br>INVALID : address is Mandatery<br>");
  }
  if(vsage==null || vsage.equals(""))
  {
   errs.add("<br>INVALID : age is Mandatery<br>");
  }
  else    {
   try     {
    viage=Integer.parseInt(vsage);
```

Step 1

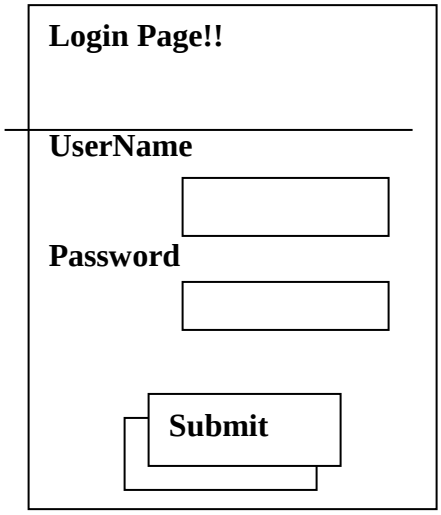
```
    if(viage>150 || viage<0)
     {
      errs.add("<br>INVALID : age must be below 0 to 150<br>");
     }
   }catch(Exception e)
    {
     errs.add("ur data is invalid");
     System.out.println("ur data is invalid");
   }
        //end of Code to Validate data (this depend on Project Requirment)..
  }
   if(errs.size()==0)                                          Step 4
    {
     out.println("storing data in our db");
    }                                                          Step 3
     else
     {
      out.println("please fill the form with correct data");
          out.print(errs);
     }
  }
}
```

* The code in the above servlet is develop according to the flow chart given below the designers are struts frame work has used the same steps as part of these frame work.

* Struts framework automates in a complex, weblogic we can reduce the total amount of code that we have to implements as part of a Struts based webapplication.



Ex :



Validations Rule:

1. User Name & Password are required.
2. Minimum length of user name must be 5 Characters & maximum length is 15 Characters.

Ex: A Simple project first a simple login page will be displayed if we don't have login id then it will say plz register sir after registration is over then it will say plz login sir.

First login page html code:

```
 <html>
 <head>
  <title>Sudarshan login page!!</title>
 </head>
 <body bgcolor=gangadher>
  <font face="helvetica">
<pre>
  <form action="login" method=GET>
  User Name    <input type=text name=userName> <BR>
    Password   <input type=password name=userPassword>
 <BR><BR>                  <input type=submit  value = login>
  </form>
       <b> If you are a new user Please <a href = userreg.html>Register</a></b>
 <pre>
  </font>
 </body>
```

```
</html>
```

Register login page html code :

```
 <html>
  <head>    <title>Sudarshan Login Page </title>  </head>
  <body bgcolor= gangadher>    <font face="helvetica">
   <pre>
    <form action="register" method=GET>
      User Name          <input type=text name=userName> <br>
      Wife Name          <input type=text name=wifeName> <br>
      Password           <input type=password name=pwdOne> <br>
      Re-Type Password   <input type=password name=pwdTwo> <br>
      Address            <input type=text name=address> <br>
                         <input type=submit  value = register>
    </form>
   </pre>
   </font>
  </body>
</html>
```

* The user name & passwords can be stored in a data base/directory servers.

* **First we use LoginServlet java program.**

```
import java.sql.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   String vuserName,vuserPassword;

   // Code to capture the data....

   vuserName=request.getParameter("userName");
   vuserPassword=request.getParameter("userPassword");

   // end of Code to capture the data....

   //Code for debugging.....

    System.out.println("vuserName--->"+vuserName);
    System.out.println("vuserPassword--->"+vuserPassword);

    //Code for debugging.....
```

```
   PrintWriter out=response.getWriter();

 /*   out.println("vuserName--->"+vuserName);
   out.println("vuserPassword--->"+vuserPassword);
 */
  // Code to Validate data (this depend on Project Requirment)..

  if(vuserName==null || vuserName.equals(""))
   {
    out.println("INVALID : user name is Mandatery ");
   }
  if(vuserPassword==null || vuserPassword.equals(""))
   {
    out.println("INVALID : user Password is Mandatery ");
   }

  //end of Code to Validate data (this depend on Project Requirment)..

  else
  {
   try
   {
    Class.forName("oracle.jdbc.driver.OracleDriver");
    Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");

    Statement stmt=con.createStatement();
    String sqlstmt="select * from myappusers where username='"+vuserName+"' and
password='"+vuserPassword+"'";
    ResultSet rs= stmt.executeQuery(sqlstmt);
    System.out.println("Executed-->"+sqlstmt);
    if(rs.next())
      {
       out.println("<body bgcolor=gangadher>");
       out.println("<b>Now you Access Your Web Site</b>");
      }
       else
       {
        out.println("<body bgcolor=gangadher>");
        out.println("<b>Wrong user Name & Password please Retry<a
href=\"login2.html\">Login</a> .....</b>");
       }
   }
       catch(Exception e)
          {
        out.println("<body bgcolor=gangadher>");
         out.println("<b>your link is not correct plz re type.....</b>");
       }
```

```
    }
  }
}
```

* **Second We use RegisterServlet program.**

```
import java.sql.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RegisterServlet1 extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   String vuserName,vwifeName,vpwdOne,vpwdTwo,vaddress;

 // Code to capture the data....

   vuserName=request.getParameter("userName");
   vwifeName=request.getParameter("wifeName");
   vpwdOne=request.getParameter("pwdOne");
   vpwdTwo=request.getParameter("pwdTwo");
   vaddress=request.getParameter("address");


   // end of Code to capture the data....

   //Code for debugging.....


   System.out.println("vuserName--->"+vuserName);
   System.out.println("vwifeName--->"+vwifeName);
   System.out.println("vpwdOne--->"+vpwdOne);
   System.out.println("vpwdTwo--->"+vpwdTwo);
   System.out.println("vaddress--->"+vaddress);

  //Code for debugging.....

   PrintWriter out=response.getWriter();

  // Code to Validate data (this depend on Project Requirment)..

   if(vuserName==null || vuserName.equals(""))
    {
     out.println("INVALID : user name is Mandatery ");
    }
   if(vwifeName==null || vwifeName.equals(""))
```

```
    {
     out.println("INVALID : wife name is Mandatery ");
    }
   if(vpwdOne==null || vpwdOne.equals(""))
    {
     out.println("INVALID : Password is Mandatery ");
    }
   if(vpwdTwo==null || vpwdTwo.equals(""))
    {
     out.println("INVALID : Re-Type Password is Mandatery ");
    }
  if(vaddress==null || vaddress.equals(""))
    {
     out.println("INVALID : address is Mandatery ");
    }
    else
    {
     try
     {
      Class.forName("oracle.jdbc.driver.OracleDriver");
     Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");

     Statement stmt=con.createStatement();
     String sqlstmt="insert into myappusers values('"+vuserName+"','"+vwifeName+"',
'"+vpwdOne+"','"+vaddress+"')";
     stmt.executeUpdate(sqlstmt);
     System.out.println("Executed-->"+sqlstmt);
     con.close();
  out.println("<body bgcolor= gangadher>");
  out.println("<b>Your Registeration is Successfully completed Sir Please <a
href=\"login2.html\">Login</a>..... </b>");
    }
     catch(Exception e)
     {
      out.println("sorry sir.....");
     }
    //end of Code to Validate data (this depend on Project Requirment)..
   }
  }
}
```

* **Problems in the above code.**

1. The performance of the Servlet will be effected as we are directly getting the data base connection using DriverManager.getConnection method to improve the performance we must modify the code to get the connection from the connection pool, using data source.getConnection.

2. The above Servlet fails if the data base administrator changes the data base settings like user name & password etc……

3. As part of the servlet we should not hard code things like driver class URL, user name ,password in servlets we can avoid hard coding **init params (or) context params**.

* We can avoid Hard Coding in the servlets using initialization parameters (init parameters).

Ex:

```
import java.sql.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetdataServlet extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
    PrintWriter out=response.getWriter();
    try
    {
     Class.forName("oracle.jdbc.driver.OracleDriver");
     Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");

     Statement stmt=con.createStatement();
     String sqlstmt="select empno,ename,sal,deptno from emp";
     ResultSet rs= stmt.executeQuery(sqlstmt);
     System.out.println("Executed-->"+sqlstmt);
     out.println("Data obtained at"+new java.util.Date()+"<br>");
     while(rs.next())
     {
      out.println("----------------------<br>");
      out.println(rs.getString(1)+"<br>");
      out.println(rs.getString(2)+"<br>");
      out.println(rs.getString(3)+"<br>");
      out.println(rs.getString(4)+"<br>");
      out.println("----------------------<br>");
     }
    }
     catch(Exception e)
     {
      out.println(e);
     }
  }
}
```

* in the above program to get the data from data base throw browser .
* Every Browser allocates some space on the Hard disk to hold the content that is serving by a web container this is called as browser **Cache** in case Internet Explorer it is temporary files folder.

To see this content please go on Internet Explorer

Tools → Internet Options → Settings → View Files

* The content stored in the Browsers Cache link temporary memory if we close the Browser then it is also closed means remove the file in cache link.

* When we type the URL of the servlet the Browser checks whether the content generated while that servlet is available in its cache & it checks the expire time also, if the content is not expire the browser displays the content that is available in its cache instead of sending the request to the web container.

* By default if we have not set the expire time the browser assume that the content will never expire.

* As part of our servlets we must use **response.setDateHeader()** method to set the expire Date & time of the content if the content has to expire immediately we can add

response.setDateHeader("Expires",1000*60*60);

mille seconds

* If the content has to expire after one hour we can add

```
java.util.Date  now=new java.util.Date();
long   l = now.getTime();
 response.setDateHeader("Expires"l+(1000*60*60));
```

* To tell the browser that it shouldn't cache the content as part of our servlet we can use

response.setHeader("Cache-Control","no-store");

* If we use old browsers like **HTTP 1.0** then we write

response.setHeader("pragma","no-cache");

* In some project the content generate in servlet may be value for some amount of time in this case we must use **response.setHeader()** by passing the time when the content has been expires.

RequestDispatcher is an object that is responsible for dispatching the request to a resource (a resource may be a dynamic resource like servlet or a static resource like HTML file).

Note: In case of servlet executing the service method can be considered as dispatching the request.

* As part of requestDispatcher interface two methods are provided
ie 1.include 2.forward.

1.**Include method**.

* A simple project using Include method.

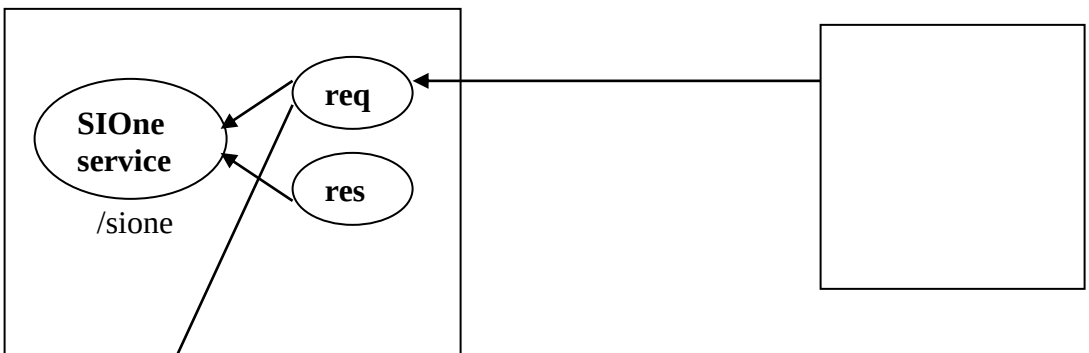
1.**SIOne.java**

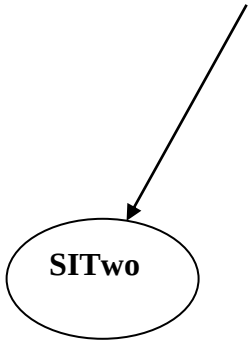
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SIone extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   PrintWriter out=response.getWriter();
   ServletConfig config=getServletConfig();
   ServletContext application=config.getServletContext();
   RequestDispatcher rd=application.getRequestDispatcher("/sitwo");
   out.println("<b><br>line one of sione<br></b>");
   rd.include(request,response);
   out.println("<b><br>line two of sitwo<br></b>");
  }
}
```

2.**SITwo**.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SItwo extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   PrintWriter out=response.getWriter();
   out.println("<b>the output is generated by sudhi soudhi</b>");

 }
}
```

* If **multiple servlets** are involved in processing **one request** we can say that a chain of servlets are involved in processing a request (this is called as servlet chaining).





WebBrowser

WebContainer

Step1 : When the user types the url **/sione** the browser sends the request to the web container.
Step2 : The web container creates a request object & a resource object, and it calls the service method of **SIOne** by passing the request object & the response **object**.
Step3 : When **SIOne** executes
RequestDispatcher rd=application.getRequestDispatcher("/sitwo");
a RequestDispatcher object will be created which can dispatch the request to **SITwo**.

Step4 : When **SIOne** executes rd.include(request,response);
The RequestDispatcher object starts the execution of the service method of **SITwo** by passing the same request object & response object which were pass as parameters to **SIOne**.

Step5 : Finally the browser gets the output ie generated by **SIOne** as well as **SITwo**.
Note : In this case two servlets are getting executed when one request is sent to by the browser this is called as servlet chaining.

2.forward method.

* simple project using forward method.

1.SFOne.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SFone extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   PrintWriter out=response.getWriter();
   ServletConfig config=getServletConfig();
   ServletContext application=config.getServletContext();
```

```
  RequestDispatcher rd=application.getRequestDispatcher("/sftwo");
  rd.forward(request,response);
  out.println("<b><br>line one of sfone<br></b>");
 }
}
```

2.SFTwo.java

```
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SFtwo extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
   PrintWriter out=response.getWriter();
   out.println("<b>the output is generated by sudhi soudhi</b>");
  }
}
```

Step1, Step2,Step3 are same as above.

Step4 : When SFOne executes RequestDispatcher.forward the output ie generated by SFOne will be discarded and it will start the execution of the service method of **SFTwo** by passing the same request & response .

Step 5 : Finally the browser gets the output ie generated by **SFTwo**. In this case also multiple servlet are included in processing one request, this is also called as servlet chain.

* Dispatching means Handing over the request.

* We can use RequestDispatcher.include() multiple times to include the ouput of multiple servlets but a servlet can forward a request to only one resource.

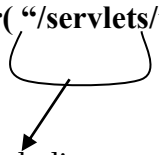
* If we can access to **two.html** file by typing the following url in the browser.

http://localhost:8080/sudhi/servlets/two.html

Context path

* As part our servlet forgetting the servlet dispatcher we can use

application.getRequestDispatcher("/servlets/two.html");



url of the resource execluding context path.

* As part of most of the web application some portions of the web pages will be same or similar instead of repeatedly providing the same code as part of every servlet to generate the

same output we can develop the code as part of one servlet & include the output of that servlet in other servlets.

* In java we use try catch blocks to separate the code that takes care of the actual task from the code that takes care of handling the Exception. In case of our servlets we can implement a servlet to exclusively take care of handling the error.

Ex: if we take a simple project then if we have emp table in data base then retrive the emp no , emp name otherwise give the Exception in servlet in browser page……..

The code is

1. **ActualTask.java**

```
import java.sql.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ActualTask extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
    PrintWriter out=response.getWriter();
    try
    {
     Class.forName("oracle.jdbc.driver.OracleDriver");
     Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");

     Statement stmt=con.createStatement();
     String sqlstmt="select empno, ename from emp";
     ResultSet rs= stmt.executeQuery(sqlstmt);
     out.println("<body bgcolor=gangadher>");
     out.println("<b><h2>Data from emp table</h2></b>");
     System.out.println("Executed-->"+sqlstmt);
     while(rs.next())
     {
      out.println(rs.getString(1)+"<br>");
      out.println(rs.getString(2)+"<br>");
      out.println("----------------------<br>");
    }
     con.close();
    }
     catch(Exception e)
     {
      ServletContext application=getServletContext();
      RequestDispatcher rd=application.getRequestDispatcher("/eh");
      rd.forward(request,response);
```

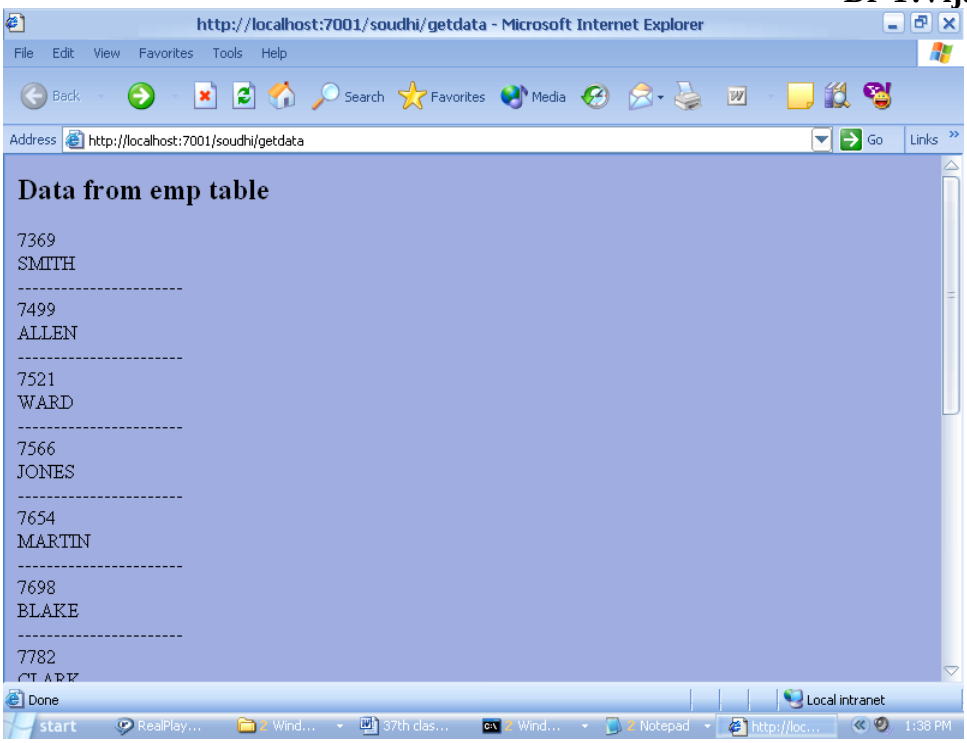
```
    }
  }
}
```

2. To handle the Exception ie **Error.java**

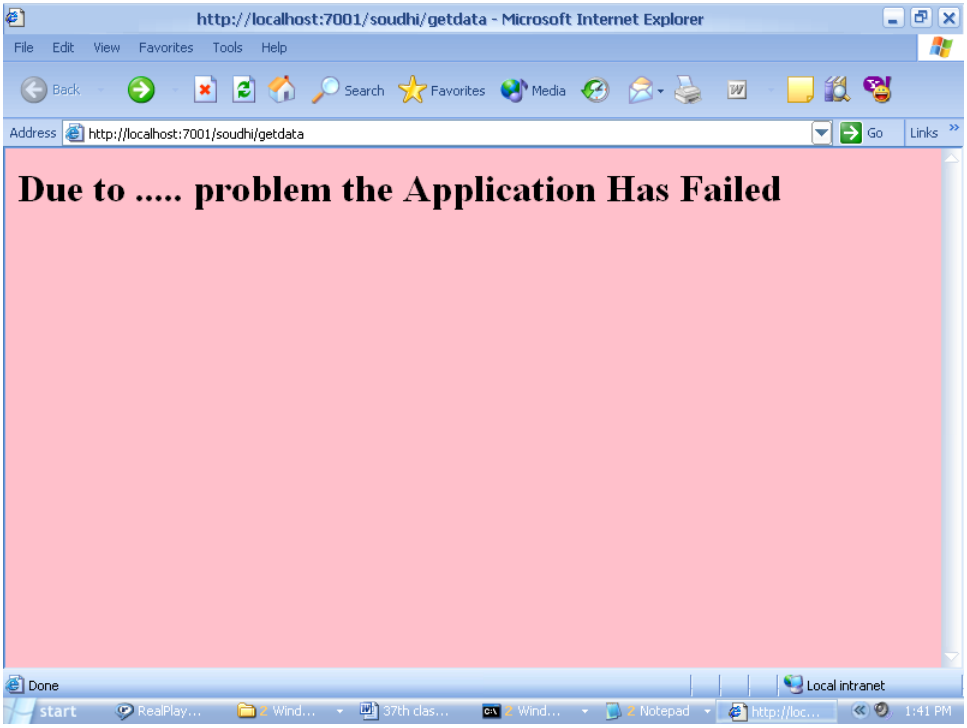
```
  import java.sql.*;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Error extends HttpServlet
{
 public void service (HttpServletRequest request,
                      HttpServletResponse response)throws ServletException, IOException
  {
    PrintWriter out=response.getWriter();
    System.out.println("Executing Error Handler");
    out.println("<body bgcolor=pink>");
    out.println("<b><h1>Due to ..... problem the Application Has Failed</h1></b><br>");

   }
}
```

* If the try block of the **ActualTask** servlets executes successful the complete output ie generated by the actual task servlet will be send to the browser.
* If it fails to executes the try block of actual task it will start executing the catch block, as we have used **rd.forward** method the output ie generated by the **ActualTask** till that point will discarded & Error Handler servlet will be executed finally in the browser we will see either the output of actual task or the output of Error Handler.
* **RequestDispatcher.forward / include** can be used to develop an application using Servlet Chaining by using this feature we can provide the code to carry out the task using multiple servlets this technique reduces the redundant code an improves the maintability of the code.

* The output is if we success then the Browser displays



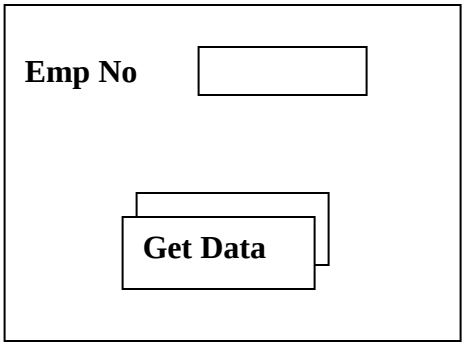
Then if the table is not there in data base then it displays Error page in browser ie



Purpose of using :-

1.request.setAttribute
2.request.getAttribute

3.request.removeAttribute



* When the user provides the emp no & clicks on get data button the application must send the details of the emp to the browser.
* In this application they can develop

1. Only one servlet to take care of getting data from data base & send the data to the browser.

(or)

2. We can develop two servlets "one to get data from the data base & other for sending data to browser.

Using first method :-

* ServletContext has two methods 1.**ServletContext.getRequestDispatcher()**
2.**ServletContext.getNamedDispatcher()**

* What is the difference between **ServletContext.getRequestDispatcher() & ServletContext.getNamedDispatcher() ?**

Ans: For **ServletContext.getRequestDispatcher()** the parameter is the path to the resource excluding the context path (which is nothing but the URL of a servlet specified in **web.xml** file). For **ServletContext.getNamedDispatcher()** the name of the Servlet specified in web.xml must be passed as a parameter.

* What is the Difference between ServletContext.getRequestDispatcher() & request.getRequestDispatcher() ?

Ans: In case of ServletContext.getRequestDispatcher() as well as request request.getRequestDispatcher() method we can pass the path of the resource excluding the context path & the path must start with a / .
* The relative path we should not start / .
* In case of **request.getRequestDispatcher()** we can specify the relative path of the resource.

AbsolutePath → CompletePath

D:\j2ee>dir d:\j2ee\servlets\ActualTask.java
* For giving relative path when we use **request.getRequestDispatcher()**

D:\j2ee>dir servlets\ActualTask.java

* If we use the following mapping.

Servlet Name	**URL Name**

One /done/dtwo/sone
Two /done/dtwo/stwo

Relative path of "two from "one" → dthr/stwo

One /done/sone
Two /done/dtwo/dthr/stwo

Relative path of "two" from "one" → dtwo/dthr/stwo

One /sone
Two /stwo

Relative path of "two" from "one" → stwo

One /done/dtwo/sone
Two /done/dthr/stwo

D:\done\dtwo>dir ../dthr/stwo

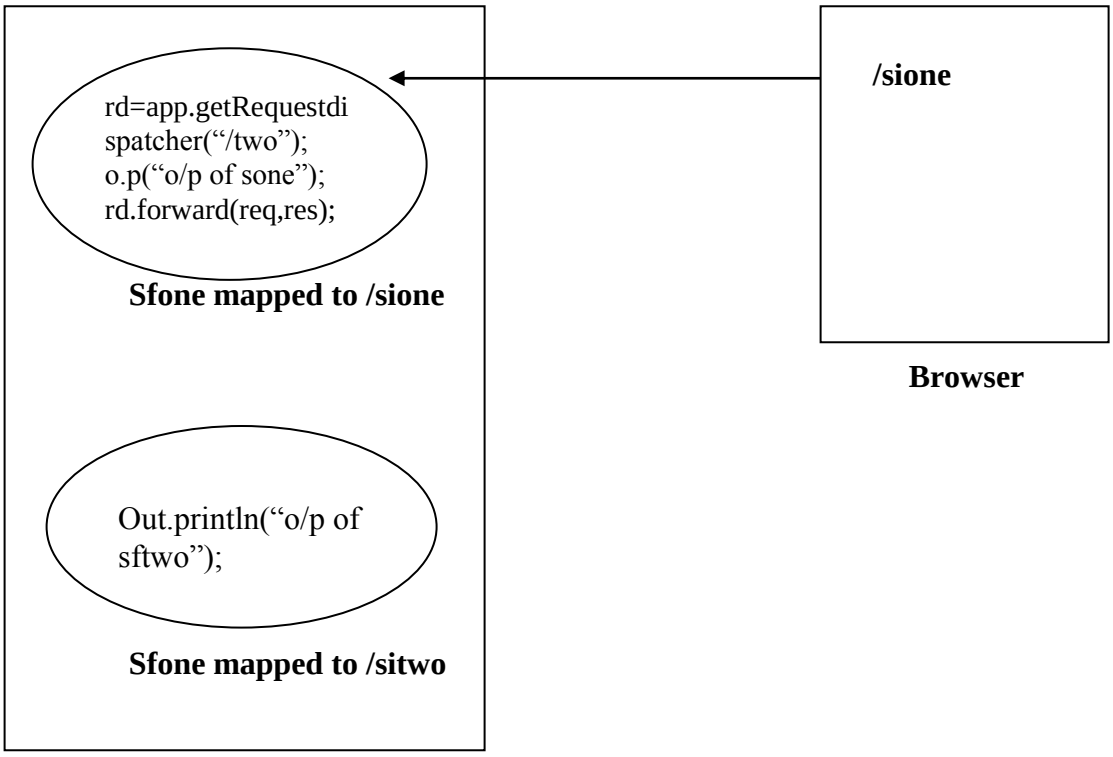
.. means Parent directory

* We can specify relative path request.getRequestDispatcher()

* What is the Difference between RequestDispatcher.forward()
& response.SendRedirect()

Ans: In case of **RequestDispatcher.forward()** finally we get the output by **sftwo** servlet.
In this case the browser will send only one request & the container will executes two servlets.

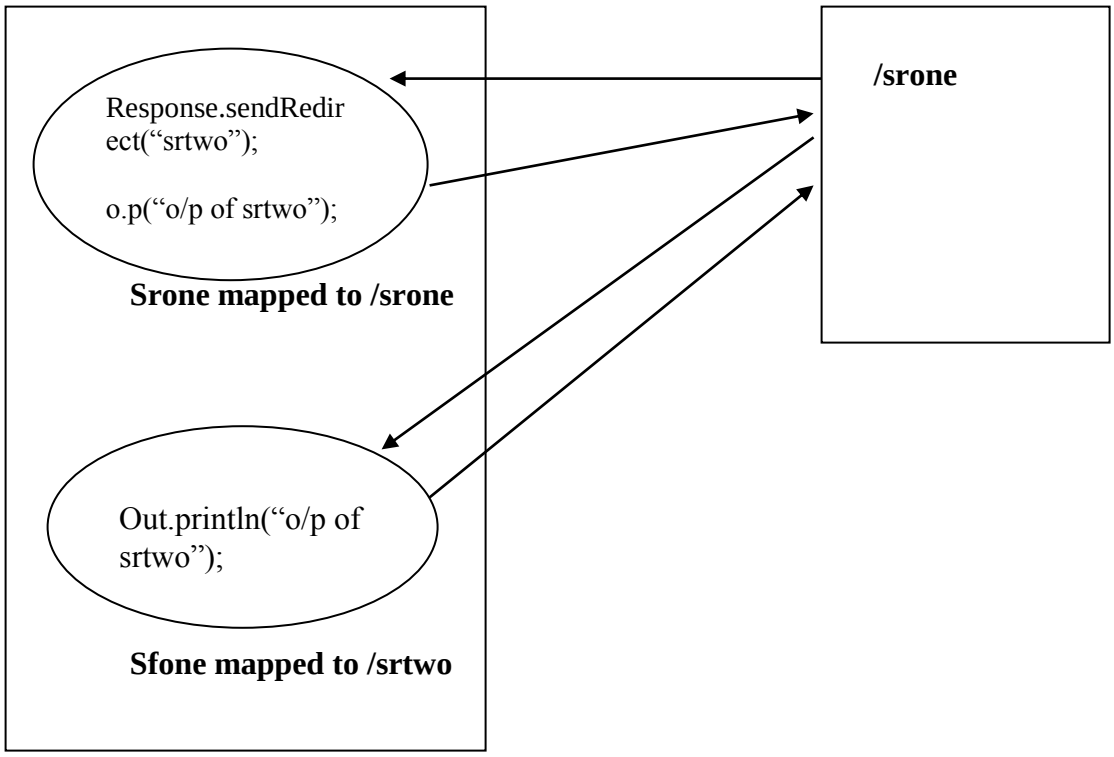
RequestDispatcher.forward()



* In case of rd.forward() finally we get the output by sftwo servlet.

* In this case the browser will send only one request & the container will executes two servlets.

* **response.sendRedirect()**



Step1: When we use srone URL in the browser the browser sends a request for srone to the web container.
Step2: The web container executes srone servlet as srone servlet uses response.sendRedirect() the web container adds a **Header** giving the location (URL) of srtwo.
Step3 : The browser will send a second request to the web container using the URL of srtwo, the container will execute srtwo & send the output to the browser.

* By using **rd.forward()** we can forward the request to any resource that is available in the same web container, but in case of **response.SendRedirect()** we can Redirect the browser to a resource available in the same web container or we may different web container.

Topic : Statefull,Stateless

* A person x may start conversation at 10 o.clock with person one & give some information during the conversation till 10.15 if person can remember the conversational state we can say person one is **statefull** if we can't remember we can say he is **stateless**.

* In case of a web application the user who want to use the application uses a browser and clicks on various links or on various buttons to send the request to the server & the server will send the response this is called as conversation between the client & the server.

The designers of HTTP protocol has design the protocol to be **stateless** ie the web server will remember the conversation state.

* Most of the modern web application uses different techniques like

1.HiddenFields.
2. Cookies
3.
(a). Sessions using **Cookies.**
(b).Sessions using **URL- rewriting**.

* for a sample project using hidden fields to show the input of the content of the previous browser & current browser.

1 first we have to create **two.html** file

```
<html>
  <head>
    <title>Sudarshan login page!!</title>
  </head>
  <body bgcolor=gangadher>
    <font face="helvetica">
<pre>
    <form action="hide" method=GET>
    User Name    <input type=text name=userName> <BR>
      Password   <input type=password name=userPassword>
   <BR><BR>                    <input type=submit  value = login>
   </form></pre></font> </body> </html>
```

2.second we create **hiden.java**

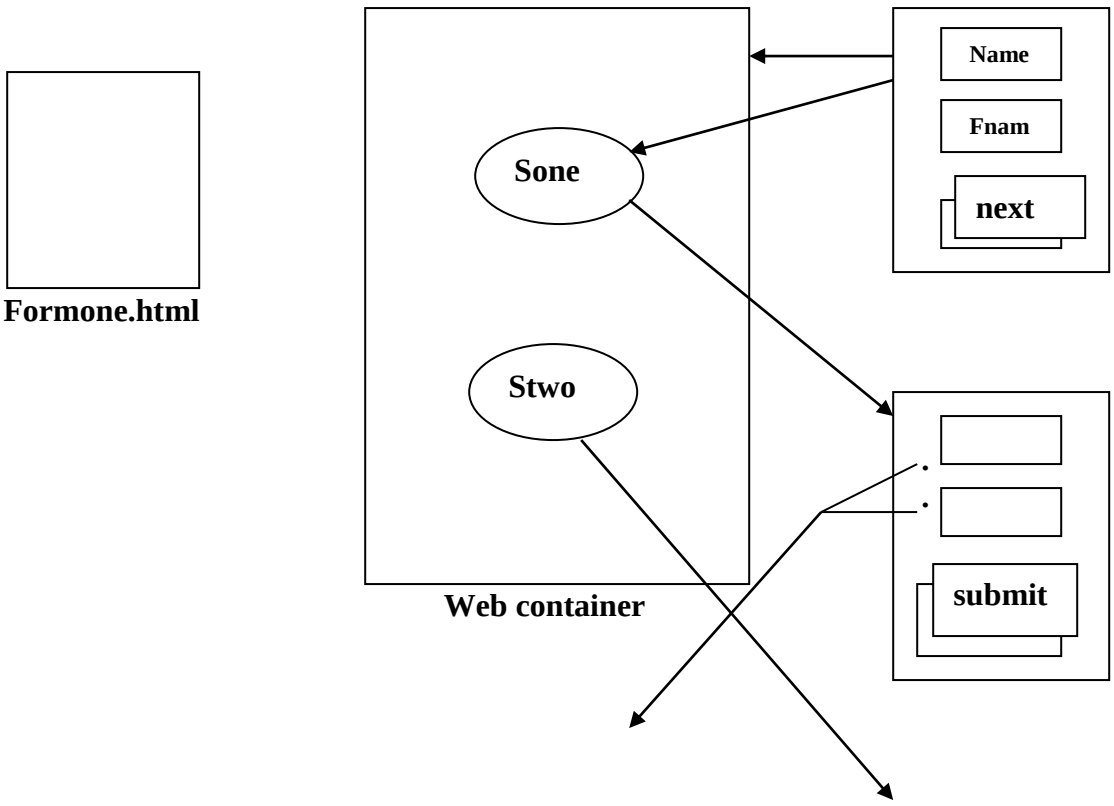
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class hiden extends HttpServlet
{
   public void doGet (HttpServletRequest request,HttpServletResponse response) throws
ServletException, IOException
   {
         PrintWriter out;
         // set content type and other response header fields first
       response.setContentType("text/html");

         // then write the data of the response
         out = response.getWriter();
      //get values submitted by the form
      String sudhi = request.getParameter("userName");
      String soudhi = request.getParameter("userPassword");
```

```
// now we need to genetare the second form dynamically from here
        out.print("<html>   <head>     <title>Personal Details 2 </title>");
        out.print("  </head>  <body bgcolor=gangadher>");
        out.print("<form action=\"hide1\" method=get>");
        out.print("<pre>Wife Name    <input type=text name=wifeName> </pre>");
        out.print(" <pre>password    <input type=password name=password></pre>");
// here we are writing the data back in the form as hidden fields.
        out.print(" <input type=hidden name=userName value ="+ sudhi+">");
        out.print(" <input type=hidden name=userPassword value ="+ soudhi+">");

        out.print(" <pre>                <input type=submit  value = submit></pre>");
        out.print("</form> </font>  </body> </html>");
        out.close();
   }
}
```

3.we create **hiden1.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
public class hiden1 extends HttpServlet
{
   public void doGet (
        HttpServletRequest     request,
        HttpServletResponse  response
   ) throws ServletException, IOException
   {
        PrintWriter               out;
        // set content type and other response header fields first
      response.setContentType("text/html");

        // then write the data of the response
        out = response.getWriter();
//get values submitted by the form
String WifeName= request.getParameter("wifeName");
String Password = request.getParameter("password");
String UserName = request.getParameter("userName");
String userPassowrd= request.getParameter("userPassword");
// here we can use jdbc to store the values in database
   try
   {

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

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

    Statement stmt=con.createStatement();
    String sqlstmt="insert into sample
values('"+UserName+"','"+userPassowrd+"','"+WifeName+"','"+Password+"')";
    stmt.executeUpdate(sqlstmt);
    System.out.println("Executed-->"+sqlstmt);
     out.print("<html>  <head>    <title>Personal Details</title>");
     out.print("  </head>  <body bgcolor=gangadher>");
     out.print("<h1><b>Thanks for submitting U r Details</b></h1><br><br><br>");
     out.print("</body> </html>");
     out.println("<h2><b>"+UserName+"</b>");
     out.println("<b>"+userPassowrd+"</b><br>");
     out.println("<b>"+WifeName+"</b>");
     out.println("<b>"+Password+"</b></h2><br><br><br><br>");
     out.print("<br><br><br><br><br><br><h2>U r information is sucessfully stored in Data
Base");
     con.close();
    }
     catch(Exception e)
     {
      out.println(e);
     }

   }
}
```



Hidden Fields with ram,pass of values.

Stored all the data in Data Base.

* In the above scenario the browser starts the conversation by sending the request for **Formone.html**

The user after providing the values for **uname & pwd** fields clicks on **next** button the browser now sends a second request to the server for the resource as part of this request **uname, pwd** values provided by the user.

* **Sone** Capture the data & generates the second form as part of the second form **two Hidden Fields + two Visible Fields** will be added as part of the Hidden fields the capture data will be placed.

* A simple project by using **Cookies** to input **Name , Father Name** fields & to input **Income Tax** .

1.CTax1.html

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Income tax details -form XXX - Page no -I </title>
  </head>

  <body bgcolor=gangadher>
    <font face="helvetica">
    <pre>
    <form action="sone" method=GET>
       name        <input type=text name=name> <BR>
       Father name <input type=text name=Fname>
            <BR><BR>                    <input type=submit  value = next>
            <br><br> Like this u can have any no of fields <br><br>
      <br > After filling this form submit this form by clicking next <br>
, then next filling second form<br>

    </form>
    </pre>
    </font>
  </body>
</html>
```

2.Sone.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Sone extends HttpServlet
```

```
{
   public void doGet (
         HttpServletRequest     request,
         HttpServletResponse  response
   ) throws ServletException, IOException
   {
         PrintWriter                out;
         // set content type and other response header fields first
      response.setContentType("text/html");
      // then write the data of the response
         out = response.getWriter();
      //get values submitted by the form
      String name = request.getParameter("name");
      String fname = request.getParameter("Fname");

         // create a cookie to store the values of name and fname
         Cookie c1 = new Cookie("name",name);
         Cookie c2 = new Cookie("fname",fname);
         response.addCookie(c1);
         response.addCookie(c2);
      // now we need to generate the second form dynamically from here
        out.print("<html>     <head>        <title>Income tax details -form XXX - Page no -2
</title>");
         out.print("  </head>  <body bgcolor=gangadher>");
         out.print("<form action=\"stwo\" method=get>");
         out.print("<pre>");
         out.print("income for this year <input type=text name=income> <BR>");
         out.print(" <br>Tax                <input type=text name=tax><br>");
         out.print("<br><br><br>");
         out.print("<BR><BR><input type=submit  value = submit><BR>");
         out.print("</pre></form> </font>  </body> </html>");
         out.close();
   }
}
```

3.Stwo.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Stwo extends HttpServlet
{
   public void doGet (
         HttpServletRequest     request,
         HttpServletResponse  response
   ) throws ServletException, IOException
   {
         PrintWriter                out;
         // set content type and other response header fields first
```

```
        response.setContentType("text/html");

            // then write the data of the response
            out = response.getWriter();
//get values submitted by the form
String income = request.getParameterValues("income")[0];
String tax = request.getParameterValues("tax")[0];
// here we can use jdbc to store the values in database


            out.print("<html>  <head>    <title>Income tax details</title>");
            out.print("  </head>  <body bgcolor=gangadher>");
            out.print("<b>Thanks for submitting income tax form<br>");
            out.print(" following information is stored in our database");
            out.print(income+"<br>"+ tax +"<br>");
            Cookie[] c= request.getCookies();
            if(c!=null){
            for(int i =0 ;i<c.length;i++)
                    out.println(c[i].getName() +"....."+c[i].getValue());
            }
            out.print(" <BR><BR>Like this we can store the state of a client on client by using
Cookies");
            out.print("</b></body> </html>");
            out.close();
    }
}
```

* The User after filling the values for **Name** & **Father Name** Clicks on **Next** button the browser will now send the **Third** request as part of the **Third** request the browser will send the values of **Four** fields, This request will be send to **Stwo & Stwo** Can capture the values of **Four** fields & Store the data inside the **Data Base**. In this Example the **End User** gets filling that application is able to remember what he has given as part of **CTax1.html** when he has submitted the values of form two , In this technique more amount of data will be transferred between the Client & Server.
* We can use **Cookies** in place of **Hidden fields** to implement the same application.

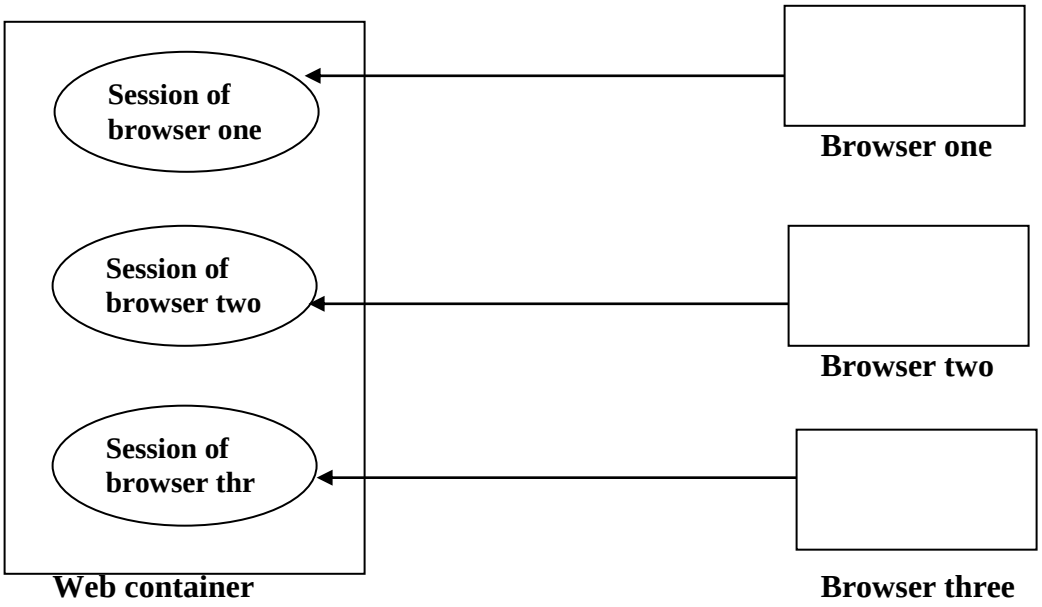
* **Cookie** is a small piece of information send by the server on the client.

* The First parameter passed to the constructor of the cookie called as **none of the cookie** & second parameter is called as **value of the cookie**.
* When we send the request to the above Servlet it will send two cookies (Name, Father Name) to the browser , The browser keeps this two cookies in the memory and these two cookies will be lost if we close the browser .
* The Browser will **SendBack** The Cookies that are set by a Server.
* In order to receive cookies as part of servlet we can write the code as shown above.
* If the Cookies are set by a servlet running in the web container on www.hotmail.com the browser will send back these cookies only to the resources that are available on www.yahoo.com .
* There is a limit on no of cookies that are allowed per domain.

* Similar to hidden fields in case of cookies more amount of data is transformed between the client & the server.
* We can use sessions to reduce the amount of data transferred between the client & the server by using **sessions** (instead of hidden fields & cookies).

Topic : Sessions

* We can use HttpSession object to hold anything ie **Specific to a Client**.



* As shown in the above diagram the web container is responsible for managing (creating , associating an object to the client ,removing) one HttpSession object for every client (Browser) ie accessing the web application.
* As part of our Servlet we can't create HttpSession object using the code shown below.

HttpSession session = new HttpSession();

(HttpSession is an interface so it is not possible to create object).

* To Access HttpSession object from a servlet we can write the code as shown below.

HttpSession session = request.getSession(true);

(or)

HttpSession session = request.getSession(false);

Ex : **SesOne.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SesOne extends HttpServlet
{
 public void service(HttpServletRequest request, HttpServletResponse response)throws ServletException,IOException
```

```
 {
  PrintWriter out=response.getWriter();
  HttpSession session = request.getSession(true);
  System.out.println("session-->"+session);
  System.out.println("Id-->"+session.getId());
  System.out.println("Is Newly Created -->"+session.isNew());
 }
}
```

2.**SesTwo.java**

Same as above code simple change it is shown below
HttpSession session = request.getSession(false);

** When a Servlet executes **HttpSession session =request.getSession(true)**;
the web container checks whether there is a session object on behalf of the client that has sent the request if the session object is not there the Web Container creates a new session object for that client, If the session object is already available on behalf of a client the Web Container **will not create a new session object**. It will return the old session object.
* If we use **HttpSession session = request.getSession(false);**
The Web Container checks whether there is a session object on behalf of the client that has sent to the request.

1. If there is no session object the Web Container returns a **null** (it will not create a new session object).
2. If the session object is already available the old object will be returned.

* Internally the Web Container gives an Id for every session object created by it, we can use **session.invalidate()** method to remove the session object. It is shown in below Example.

Ex: **SesTwo.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SesTwo extends HttpServlet
{
 public void service(HttpServletRequest request, HttpServletResponse response)throws ServletException,IOException
 {
  PrintWriter out=response.getWriter();
  HttpSession session = request.getSession(false);
  if(session!=null)
  {
   session.invalidate();
   System.out.println("session is destroyed");
   out.println("<b> Session is destroyed</b>");
  }
 }
}
```

* Similar to **request.setAttribute()** we have **session.setAttribute()**.

Ex: Sample projects using session.setAttribute() method to retrieve the fields

1.**login.html**

```
<html>
  <head>
    <title>Sudarshan login page!!</title>
  </head>
  <body bgcolor=yellow>
    <font face="helvetica">
<pre>
    <form action="s1" method=GET>
    User Name    <input type=text name=uname> <BR>
      Password   <input type=password name=pwd>

   <BR><BR>                    <input type=submit  value = login>
    </form> <pre>   </font></body> </html>
```

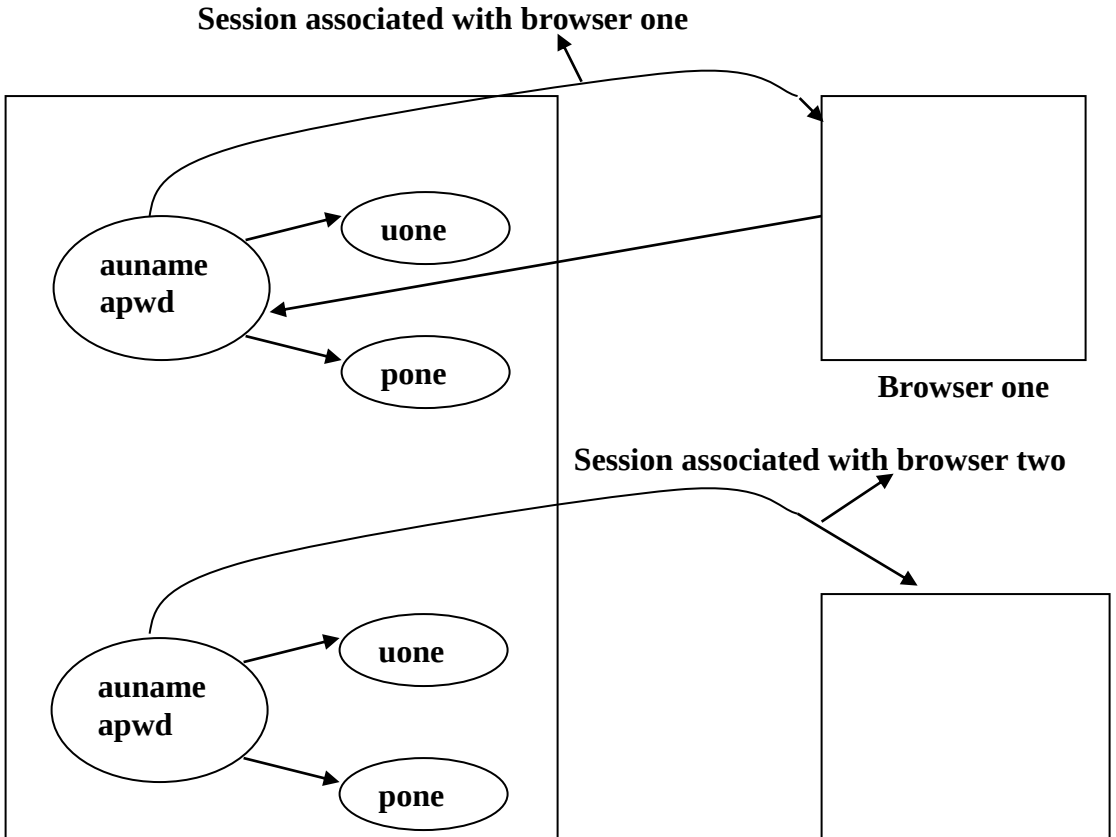
2.**SesOne1.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SesOne1 extends HttpServlet
{
 public void service(HttpServletRequest request, HttpServletResponse response)throws ServletException,IOException
 {
  PrintWriter out=response.getWriter();
  String vuname,vpwd;
  vuname = request.getParameter("uname");
  vpwd = request.getParameter("pwd");
  HttpSession session = request.getSession(true);
  System.out.println("Id-->"+session.getId());
  session.setAttribute("auname",vuname);
  session.setAttribute("apwd",vpwd);
  out.print("<html><head><title>Sessions Creations </title>");
  out.print("  </head>  <body bgcolor=gangadher>");
  out.print("<form action=\"s2\" method=get>");
  out.print("<pre>");
  out.println("fname    <input type=\"text\" name=\"fname\"><BR>");
  out.println("mname    <input type=\"text\" name=\"mname\"><BR>");
  out.println("           <input type=\"submit\" value=\"register\"><BR>");
  out.print("</form>");
  out.print("</head></html>");
  }
}
```

3.**SesTwo1.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SesTwo1 extends HttpServlet{
 public void service(HttpServletRequest request, HttpServletResponse response)throws ServletException,IOException {
  PrintWriter out=response.getWriter();
  String vuname,vpwd,vfname,vmname;
  vfname=request.getParameter("fname");
  vmname=request.getParameter("mname");
  HttpSession session=request.getSession(true);
  System.out.println("Id-->"+session.getId());
  vuname=(String)session.getAttribute("auname");
  vpwd=(String)session.getAttribute("apwd");
  out.println(vuname);
  out.println(vpwd);
  out.println(vfname);
  out.println(vmname);
 }
}
```

* In the above Example **SesOne1** captures the name , pwd & stores it inside the session objects associated with a specific client in this case SesOne1 is not sending the data to the client as hidden fields or as cookies.
* When the second form is submitted the **SesTwo1** captures fname ,mname & picks up the uname & pwd that are placed earlier by SesOne1 in the session object associated with a specific client.
* When ever we need to access the data specific to a client while processing multiple request ?
Ans: We must store the data in session object specific to the clients.



Browser two

Web Container

* When we develop an application we need to use **some data** like the name of the company the address of the company, the address of the company , the background color of the page , the foreground color of the pages , the image file that contains the ambulam of the company etc. has to be used as part of multiple servlets in a project. This data can be considered as global data as it has to be accessed by multiple servlets, this data can also be called as configuration settings (or) application preferences.
* A listener is an object that listener for a event to occur when the event occurs it carry's out some task

Ex: in AWT,JFC application ActionListener object waits for the user to click on a button , when the user clicks on a button ActionPerformed will be executed.

* As part of servlet (a) Creation of ServletContext
  (b) Removing of ServletContext
  (c) Creation of RequestObject
  (d) Removal of RequestObject
  (e) Creation of sessionObject
  (f) Removal of sessionObject etc are called as events.
* To execute the code when an event occurs we must implement the listeners.
* As part of servlet packages we have interfaces like the naming pattern xxxListener where xxx is ServletContext , ServletRequest, HttpSession etc.
* There will be **N** no of methods in the Listener interface these methods takes a parameters of type xxx event.
* As part of ServletContext listener we have

(a) ContextInitialized this method will be called after creating the ServletContext Object.

(b) ContextDestroyed this method will be called by the WebContainer before destroying the ServletContext object.

Ex: **MyList.java**

```
 import javax.servlet.*;
public class MyList implements ServletContextListener
{
 public void contextInitialized(ServletContextEvent e)
 {
  java.util.Date now=new java.util.Date();
  System.out.println("App started at....."+now);
```

```
 }
  public void contextDestroyed(ServletContextEvent e)
  {
   java.util.Date now = new java.util.Date();
   System.out.println("App stoped at....."+now);
  }
}
```

* In order to use the Listener we must copy the Listener class in WEB-INF \ classes directory & we must add the following information to web.xml

Add before <servlet> tag.

```
<listener>
        <listener-class> MyList </listener-class>
</listener>
```

* Most of the developer to day are preferring to store application preferences in the xml file.

Ex:

```
<hosp-app-conf>
<pref-info>
<pref-name>hospname</pref-name>
<pref-value>xyz hospital</pref-value>
</pref-info>
<pref-info>
<pref-name> hospaddr</pref-name>
<pref-value> someplace</pref-value>
</pref-info>
</hosp-app-conf>
```

* Earlyer developer uses to prefer normal text file (or) databases to store this kind of information.

Ex: sample project usin ServletContext we can retrive the fields of the programs.

1.**MyList1.java**

```
import javax.servlet.*;
public class MyList1 implements ServletContextListener
{
 public MyList1()
 {
  System.out.println("--MyList1 object is created");
 }
 public void contextInitialized(ServletContextEvent e)
 {
  ServletContext application = e.getServletContext();
  System.out.println("reading from xml file");
  application.setAttribute("username","Sudhi");
  application.setAttribute("wifename","soudhi");
```

```
  System.out.println("----Stored App pref in Sc");
 }
 public void contextDestroyed(ServletContextEvent e)
 {
  System.out.println("Context is destroyed");
 }
}
```

2.**retrive.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class retrive extends HttpServlet
{
 public void doGet(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
 {
  PrintWriter out = response.getWriter();
  ServletContext application=getServletContext();
  String vuname,wname;
  vuname=(String)application.getAttribute("username");
  wname=(String)application.getAttribute("wifename");
  System.out.println(vuname);
  System.out.println(wname);
  out.println("<b>user name is"+vuname);
  out.println("<b>wife name is"+wname);
 }
}
```

* A part from the name & value there will be path , domain , maxage .
* The default max age of a cookie is assume to be **-1** when a cookie is said with the default max age the browser holds the cookies till we close the browser.
* To remove the cookie we can set the max age to zero.
* While setting a cookie if we use cookie.setMaxAge(60) the browser stores the cookie in the disk & this cookie will be held for 60 seconds.

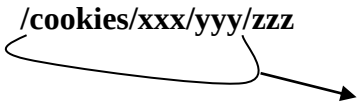
Ex: **ssone.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class ssone extends HttpServlet
{
 public void service(HttpServletRequest request , HttpServletResponse response)throws ServletException ,IOException
 {
  PrintWriter out=response.getWriter();
  Cookie c1,c2;
```

```
  c1=new Cookie("MyName","sudhi");
  c2=new Cookie("YourName","Soudhi");
  c1.setMaxAge(60*60*24*3);
  c1.setPath("/done/dtwo");
  response.addCookie(c1);
  response.addCookie(c2);
  out.println("<b> executed method</b>");
 }
}
```

* Assume that context path is **/cookies**.
* According to the above configuration to access **ssone** we need to use the following URL pattern **http://localhost:8080/cookies/xxx/yyy/zzz**.
* in the above code **MyName** cookie will not be stored in the disk.
* **YourName** cookie will be stored in the memory.
* As we have used setPath method for **MyName** cookie by passing **/done/dtwo**.
* As we are not setting the path of **YourName** cookie, the browser registers the **default Path**.

In this Example the servlet ie setting the cookie is accessed using **http://localhost:8080/cookies/xxx/yyy/zzz**. so the browser takes the path of this resource as the default path of the cookie.



Path of the resource.

* If the servlet mapping in web.xml is

```
<servlet-mapping>
        <servlet-name>s1</servlet-name>
        <url-pattern>/aaa/bbb/ccc/ddd/eee</url-pattern>
</servlet-mapping>
```

* We have to use /cookies/aaa/bbb/ccc/ddd/eee as a url to access the servlet (or) resource
* In this case the path of resource is /**cookies/aaa/bbb/ccc/ddd** .
* In this case the path **MyName** cookie is **/done/dtwo** and the path of **YourName** cookies /**cookies/aaa/bbb/ccc/ddd** why because we can't set the path of **YourName** so it will take the default resource ie /**cookies/aaa/bbb/ccc/ddd**.
* /**cookies/xyz/** is a sub path of **/cookies/** & it is also a sub path of **/**
* While sending cookies back the server sends the cookies only if
    1. The path of the resource is same as the path of the cookie.
    (or)
    2. The path of the resource is a sub path of the cookie .
* To remove the cookie we can set the MaxAge to zero.

* Some of the users set up there Browsers not to accept Cookies.
* To day most of the developers are using cookies to build the web application that can deliver personalized (according to the users requirement) content.

* Where do use make use of cookies?

Ans: Cookies are used to develop a personalized web applications.

* The Web Container uses two different techniques to manage the sessions
    1. By using Cookies.
    2. By using URL rewriting.

* When a session object is created the web container may assign an Id to the session object & this Id will be sent to the browser as a cookie with the name **jsessionId** this cookie will not be stored in the disk (this will be lost when we closed the Browser).

* After this when a request is sent to any of the resources of the web application the browser will **sendBack** the **jsessionId** cookie to the container using this Id, the container will identify the session object associated with the client (or) Browser.

*** When the browser is closed **jsessionId** cookie will be lost , The web container will not be knowing that the browser was closed by the user, so the web container will not be able to remove session object when we close the browser.
* Web container keeps track of the last accessed time of a session object if the current time minus last accessed timed exceeds session time out , the server will remove session object.

* After </servlet-mapping>

```
<session-config>
    <session-timeout>2</session-timeout>
</session-config>
```

* As part of **web.xml** wecan add the above lines to set session timeout and if we can't set the session timeout then it can take **30** minutes it is default time.

out.println("timeout→"+session.getMaxInactiveInterval());

* As part of our servlet we can use **session.getMaxInactiveInterval(60)** to set the session time out to **one** minute.
* **getMaxInactiveInterval(60)** is taken by seconds ie 60 means 60 seconds.

* Most of the web containers the default session timeout is 30 minutes.
* If the cookies are denied by the browser (or) the web container will not be able to associate the session object with the browser this is why most of the developer prefers URL rewriting technique.
* If user login in **mail.yahoo.com** but the end time he was not logout simply close the browser at that time in the server will running at that time default **getMaxInactive** method will be executed that means around 30 minutes.
* Some of the user setup the browser to denie setting cookies in this case the application that uses session object managed using cookies will fail.
* To overcome the problem most of the developers uses sessions(session object) managed by the server using URL rewriting technique.

Note : To use URL rewriting technique we must not generate the anchor tags and form tags using the URL's directly as shown below.

out.println("<form method = \"GET\" action=\"stwo\" >");

URL of another servlet.

* We must use **response.encodeURL** method.

out.println
(“<form method = \“GET\” action=\””+response.encodeURL(“stwo”)+”\”>”);

URL of another servlet is passed as parameter to response.encodeURL() here

* When we use **response.encodeURL( )** it will rewrite the URL by adding session id.
response.encodeURL(“stwo”);stwo;jsessionId=a123we546zz

* The application performance will be slightly reduce when we use URL rewriting technique.

* What are different ways used by web container to manage cookies?
Ans: **two** types **1. sessions using cookies.**
2. URL rewriting technique.
* can I access two user accounts from a single machine mail.yahoo.com?
Ans: yes only we open multiple times EnternetExplorer.exe icon,
if we use new window option then it is not possible to access.

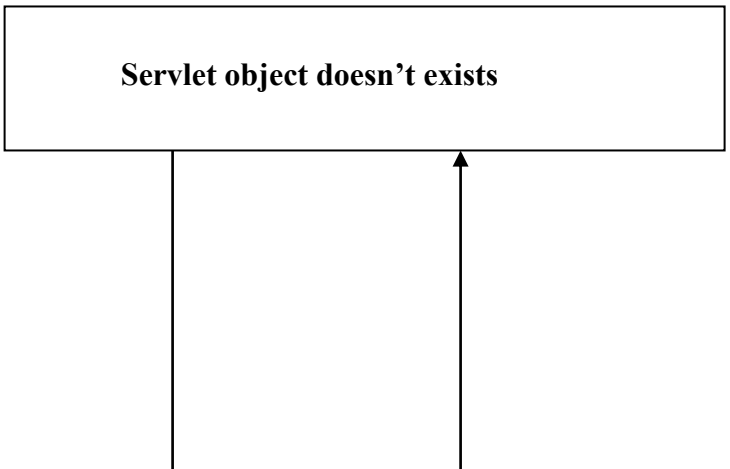
* In order to reuse the code as part of multiple of classes we can create a class say xxx with the code that is common for multiple classes & this class xxx can be used as the super class of all other classes.

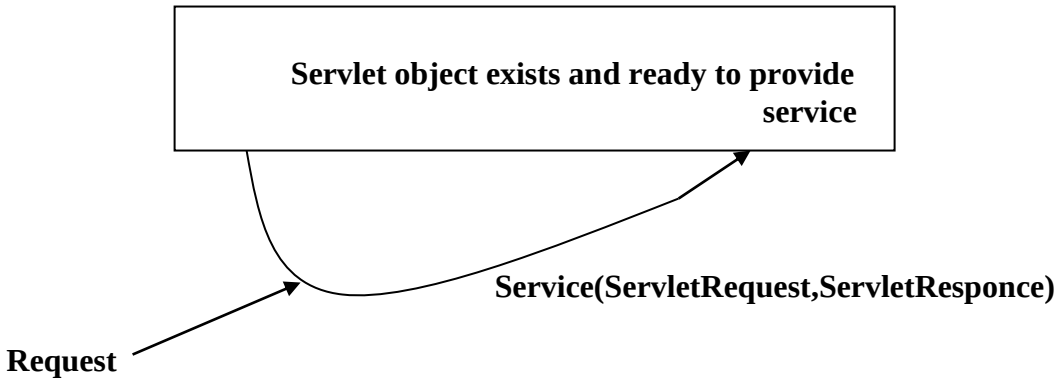
Topic : Servlets Life cycle.

* As part of servlet API java soft has defined the life cycle of servlet in this specification java soft has defined what the web container has to do at difference stages of the life of a servlet object.

Ex:

```
public static void  main()
{
    student s1;
     ------------        -------> here object doesn't exit.
     ------------
   S1=new student();-------------------------> object is created.
   // calling various methods on student object.
  S1=null;-----------------------------------> object is destroyed.
}
```





Ex:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SOne1 implements Servlet
{
 public SOne1()
 {
  System.out.println("Servlet objected is created");
 }
 private ServletConfig config;
 public void init(ServletConfig c)
        throws ServletException
 {
  System.out.println("---init method called----");
  config=c;
 }
  public ServletConfig getServletConfig()
  {
   return config;
  }
  public String getServletInfo()
  {
   return "Stores student in db";
  }
   public void destroy()
   {
    System.out.println("---Destroy method is called---");
   }
    public void service(ServletRequest request, ServletResponse response)
                                                  throws ServletException,IOException
    {
```

```
    System.out.println("---Service method called---");
   }
}
```

* We can implement the servlet interface directly as shown in the above example but in this case we need to implement all the **five** methods that are declared in the servlet interface.

1. init()
2. getServletConfig()
3. getServletInfo()
4. service()
5. destroy()

* According to the servlet specification it is the responsibility of the web container to decide about when to create a servlet object & when to remove the servlet object.
* When the servlet object is created the web container is responsible for creating Servlet Config object & pass it as a parameter to init method.
* Before removing the servlet object the web container is responsible for calling the **destroy** method.
* When ever a request is sent for a servlet, the web container is responsible for calling the **service()** method.
* **init(),service(), destroy()** are called as lifecycle methods of servlet.

* **callback()**

a method implemented by me & called another method is known as callback method.

* If we configure a servlet using **<load-on-startup>** the web container will create the servlet object when the web application is started.

In **web.xml**

```
<servlet>
        <servlet-name>SOne1</servlet-name>
        <servlet-class>sone<servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
```

* If we have two servlet one & two & if we want servlet two to be created first & servlet one created after words for servlet two we can give the value one as part of load on startup & two for servlet one.
* If load-on-startup is not specified generally the web container creates the servlet object when a first request is sent for a servlet.

* In web logic we see in **http://localhost:7001/console** in this**deployements /web applications** we have options **deploy or undeploy**.

* Some servers destroys the existing servlet object & creates a new servlet object if we modified the servlet class.
* Generally a web container destroys the servlet object when we stop the web application.
* The web container makes destroy a servlet object if there is no space to accomidate more objects & it can create the same object at a later point of time when a request is sent.

* **How many types of servlets ?**

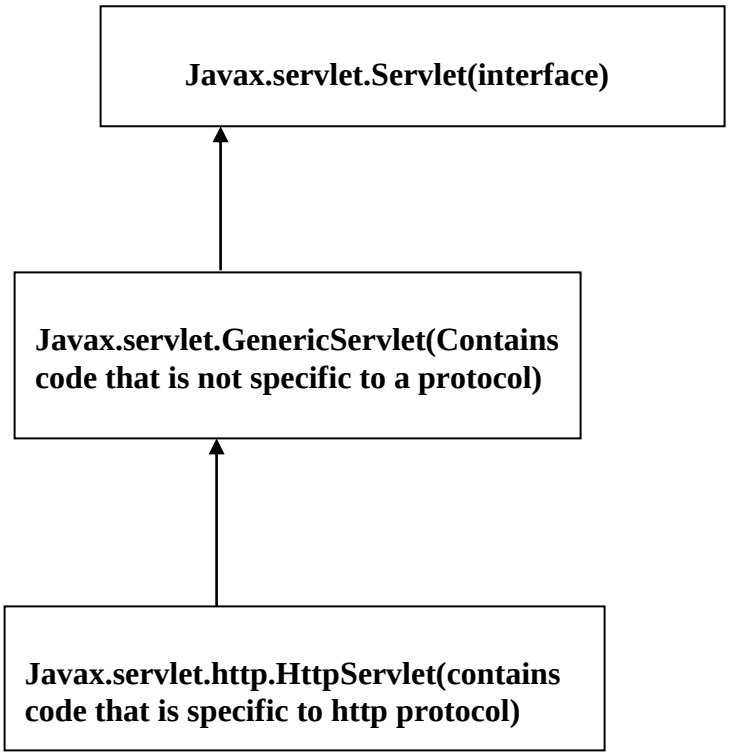
Ans: There are infinite types of servlets.

* Initially java soft has designed servlet technology keeping multiple types of servers in mind, to day the technology is implemented only as part of **HttpServers**.

* ie Servlet technology can be implemented as part of other servers also.

* a servlet running as part of **HttpServer** can be called as **HttpServlet** similarly if the servlet technology is implemented as part of **xxx server**, the servlets running inside **xxx server** can be called as **xxx servlets**.

* **What is the difference between Generic servlet & HttpServlet ?**



* **What are various ways of developing your servlets?**

Ans: N no of ways to implement a servlets.

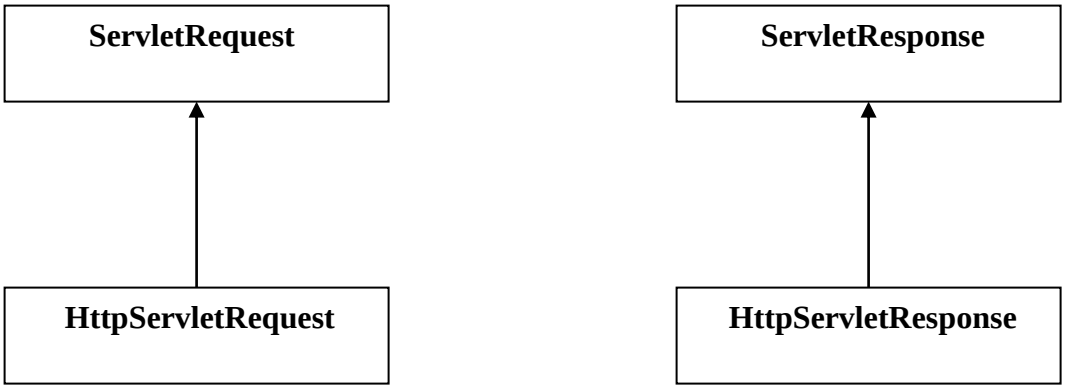
* There are multiple ways to implement our own servlets.

1. We can create a class directly implementing servlet interface.
2. We can create a sub class of GenericServlet class.

3. We can create a sub class of HttpServlet class.
4. A company can design a class like **xxxHttpServlet** & ask its developers to create there servlet as sub classes of **xxxHttpServlet**.

* **xxxHttpServlet** is a class it is directly implementing servlet interface. Like this there are **n** no of ways to implement the servlets.

* **What is the advantage of creating HttpServlet instead of GenericServlet ?**



* The above one is defined as servlet technology.
* ServletRequest ,ServletResponse interfaces are designed a Generic nature so there is not specific, & HttpServletRequest,HttpServletResponse are specific that's why we use ServletRequest interface & HttpServletRequest interface.
* As servlets are designed to be implemented as part of multiple types of servers, java soft has defined ServletRequest,ServletResponse providing the methods that are not specific to a protocol.

Ex: Using GenericServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Gsone extends GenericServlet
{
 public Gsone()
 {
  System.out.println("Gsone object created.....");
 }
 public void service(ServletRequest req,ServletResponse resp)throws
ServletException,IOException
 {
  HttpServletRequest request=(HttpServletRequest)req;
  HttpServletResponse response=(HttpServletResponse)resp;
  System.out.println("using Generic servlet.....");
  //some other code according to your requirment.
 }
}
```

Ex: Using HttpServlet

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Hsone extends HttpServlet
{
 public Hsone()
 {
  System.out.println("Hsone object created.....");
 }
 public void service(HttpServletRequest request,HttpServletResponse response)throws
ServletException,IOException
 {
  System.out.println("using Http servlet.....");
  //some other code according to your requirment.
 }
}
```

* As part of HttpServlet class java soft has provided the code similar to the one shown below.

```
Public abstract class HttpServlet extends GenericServlet
{
  Public void service(ServletRequest req,ServletResponse resp)throws ServletException,
                                                                          IOException
{
   HttpServletRequest request=(HttpServletRequest)req;
   HttpServletResponse response=(HttpServletResponse)resp;
   Service(request,response);
}
 Public void service(HttpServletRequest request,HttpServletResponse response)throws
                                                              ServletException,IOException
 {
        ----------------------------------------
        ----------------------------------------
 }
 }
```

* First **init** method → **init(ServletConfig)**
  Declared as part of servlet interface and implemented as part Generic Servlet.
* Second **init** method→ **init()**
  NOT declared as part of Servlet interface. This method is provided as part of GenericServlet class.

Note : The first init method of GenericServlet class internally calls the second init method.

Ex: MyServlet.java

```
public interface MyServlet
```

```
{
 void init(String s);
}
```

MyGServlet.java

```
public class MyGServlet implements MyServlet
{
 public void init(String s)
 {
  System.out.println("First init of MyGServlet");
  init();
 }
 public void init()
 {
  System.out.println("Second init of MyGServlet");
 }
}
```

oursrv.java

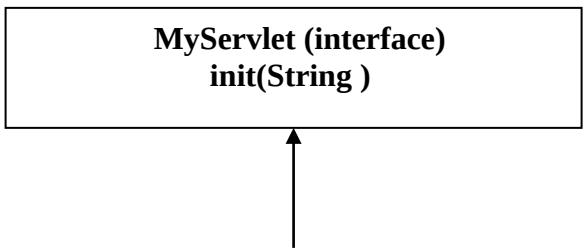
```
public class oursrv extends MyGServlet
{
 public void init()
 {
  System.out.println("Excuted sudhi");
 }
}
```

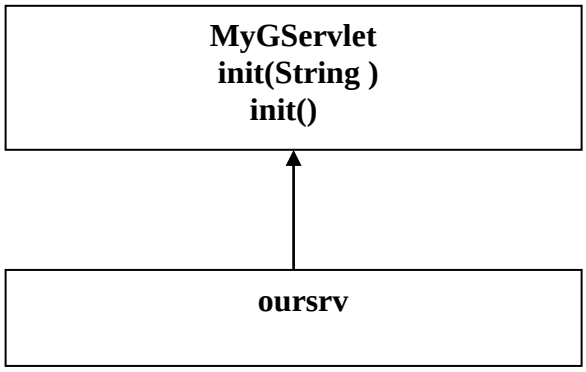
webcon.java

```
public class webcon
{
 public static void main(String args[])
 {
  MyServlet se=new oursrv();
  se.init("");
 }
}
```

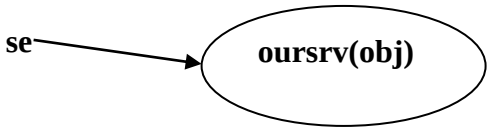
Output of above programs :

First init of MyGServlet
Excuted sudhi





*** The web container always calls the first init method that is declared as part of servlet interface.



* When ever an instance method is called the JVM
Step 1: Identified the class based on which the object is created.
Step 2: The JVM checks for the availability of the method in the class identified in **step 1** if the method is available the JVM will execute that method , if the method is not available it checks the method available in the super class then it will be executed(search in for the method will continue till the root class ie **java.lang.object**).

Ex: SOne2.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class SOne2 extends HttpServlet
{
 public void SOne2()
 {
  System.out.println("Executed SOne2");
 }
  public void init()
  {
   System.out.println("Second init");
  }
   public void service(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
   {
    System.out.println("Service method executed");
   }
}
```

* The first init() of the GenericServlet method has the code that holds the reference of ServletConfig in an instance variable after this the code calls the second init().

* What kind of code to provide in init() destroy() ?
* It is recommended to provide the code to acquire the resources (a resource may be data base…….) in the init method & de allocate the resources in the destroy method.

Ex: opening a file is called as acquiring a resource. Closing a file is called as deallocating a resource.

* Creating a java object can be considered as accruing a resource. un referencing the java object is de allocate resource.
* First service method→ service(ServletRequest,ServletResponse)
  Declared as part of servlet interface and implemented as part of HttpServlet.
* Second service()→ service(HttpServletRequest,HttpServletResponse)
  NOT declared as part of servlet interface,this method is provided as part of HttpServletClass.

Note: The first service method of HttpServlet class internally calls the second service method.

* The second service method checks the HttpRequest method used by the client (GET,POST,PUT etc) & calls doGet,doPost,doDelete etc methods.
* The web container will always call the first service method.

* A Filter is similar to servlets.
* A filter is an object that provides the implementation of **javax.servlet.filter** interface.
* As part Filter interface **Three** methods are provided.
  1. **init(FilterConfig)**
  2. **doFilter( )** this is like service method of servlet
  3. **destroy()**

Ex: Sone3.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Sone3 extends HttpServlet
{
 public void init()
 {
  System.out.println("--init method executed--");
 }
  public void service(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
  {
   System.out.println("Service method executed.....");
   PrintWriter out= response.getWriter();
```

```
  out.println("Our Sone3 Executed.....");
 }
  public void destroy()
  {
   System.out.println("......Destroy method executed.....");
  }
}
```

* in web.xml we write

```
        <servlet-mapping>
                <servlet-name>sone3</servlet-name>
                <url-pattern>*.sudhi</url-pattern>
        </servlet-mapping>
```

* With the above configuration the web container executes the service method of **Sone3** whenever we send the request using a URL that ends with **.sudhi**

* if we write the same program as different name,

Ex: Sone4.java

* then we write this servlet URL pattern is **/sone4**

```
<servlet-mapping>
                <servlet-name>sone3</servlet-name>
                <url-pattern>/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
                <servlet-name>sone4</servlet-name>
                <url-pattern>/sone4</url-pattern>
</servlet-mapping>
```

* With the above configuration **Except** for **/sone4** for all the other URLs the web container will execute **Sone3** servlet.

Ex: Fone.java

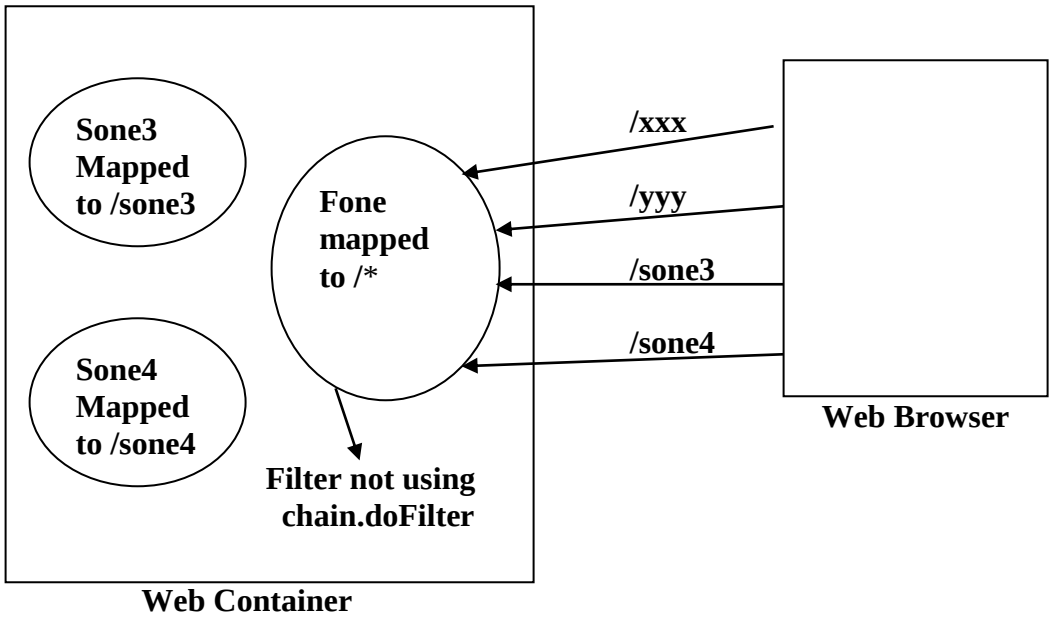
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Fone implements Filter
{
   private FilterConfig config=null;
   public void init(FilterConfig fc) throws ServletException
   {
    config=fc;
    System.out.println("init method of Fone....");
   }
   public void doFilter(ServletRequest req,ServletResponse resp,FilterChain chain) throws IOException,ServletException
   {
    HttpServletRequest request=(HttpServletRequest)req;
    HttpServletResponse response=(HttpServletResponse)resp;
    System.out.println("Started Fone.....");
```

```
    PrintWriter out=response.getWriter();
    out.println("O/p of Fone");
    System.out.println("End of Fone.....");
   }
   public void destroy()
   {
    System.out.println("Destroy of Fone.......");
    config=null;
}    }
```

* In the above program the we write **web.xml** is

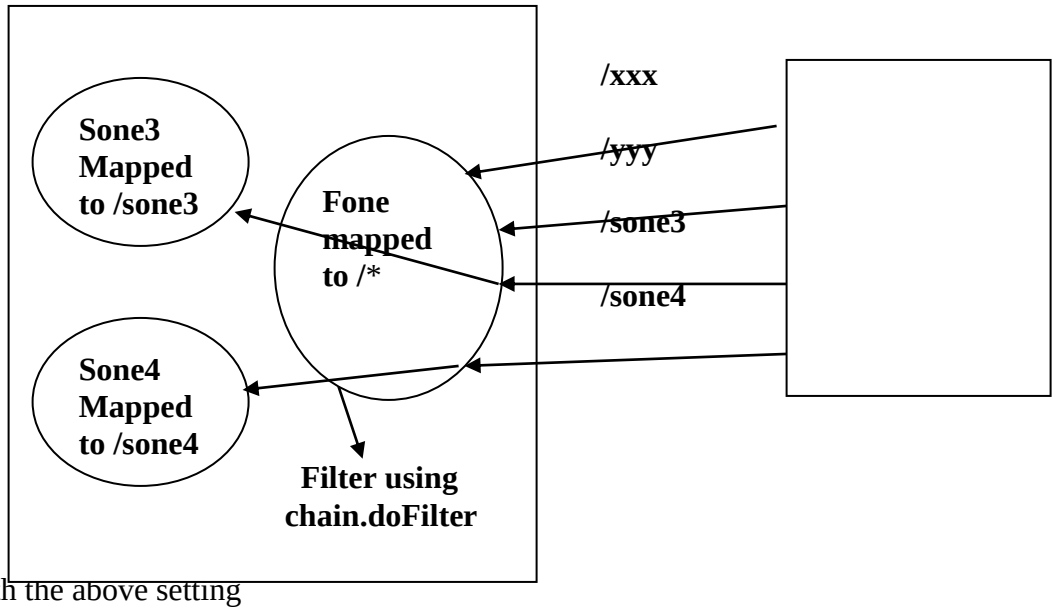
```
<filter>
        <filter-name>fone</filter-name>
        <filter-class>Fone</filter-class>
</filter>
<filter-mapping>
        <filter-name>fone</filter-name>
        <url-pattern>/*</url-pattern>
</filter-mapping>
```

* As part of the filters we will use **chain.doFilter(..)** in the above we have not use this method.



* In the above setup for every URL the web container will execute **Fone** filter even for **/sone3 & /sone4** the web container will execute **Fone** only.

* If we use **chain.doFilter(request,response)** in the above program then if we type any thing in the URL **fone** will execute in the server but Browser will give Error report, & if we type **/sone3** in the URL then **Sone3 & Fone** executed,same as if type **/sone4** then **Sone4 & Fone** will be executed.

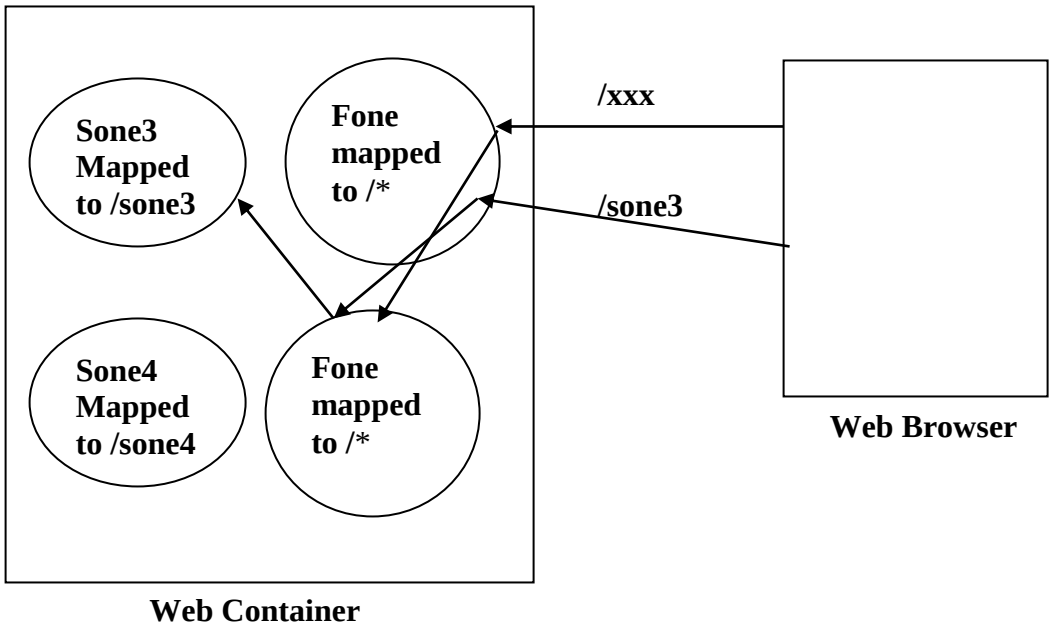


* With the above setting

Step 1: When we send the request using **/xxx** the Web container will start the execution of **fone**, When **fone** executes **chain.doFilter** the web container checks for the resource that is mapped to **/xxx** in the current setting as a resource is not available it will fail.

Step 2: If we send the request using **/sone3** the web container will execute the code in filter **fone** when **fone** executes chain.doFilter() ,the web container will execute servlet **sone3**.

* We can provide multiple filters to process a request



Note : fone & ftwo uses the chain.doFilter(..)

* Chain.doFilter causes then ex filter in the chain to be invoked ,or if the calling if the filter is last filter in the chain it invokes the resource.
* Filter can be used for performing the task like logging , gathering the performance related details, authentication , Data compression .
* It is always advisable to avoid using System.out.println statements for debugging instead of this we must use ServletContext.log()

ServletContext application= Config.getServletContext();
application.log("xxx……xxx");

* Every web container manages the log file as part our servlet as well as filters we can use **ServletContext.log()** to write the information in the log file maintain by the web container.

Application.log("recievecRequest→"+request.getRequestURL);

* It displayed in tomcat **logs** folder and file name is **localhost_log**.

* In web logic we have to see in **c:\bea\user-projects\sudarshan\myserver**.

* Authentication is the process of identifying is the user (Authentication is a process of checking whether the user is a valid user or not most of the system uses the username & passwords to authenticates the users.
* After authentication when a user tries to perform a task the system will check whether the user has permission to do the task or not this called as authorization.

* Any java application can carry out multiple tasks concurrently by using multiple threads, web container uses multiple threads to concurrently execute the service method of a servlet from multiple threads on behalf of multiple clients.

Ex:

```
 public class thread
{
 public static void main(String args[])throws Exception
 {
  System.out.println("Line one executed");
  Thread t=Thread.currentThread();
 System.out.println("thread-->"+t);
 t.sleep(3000);
  System.out.println("Line two executed");
 }
}
```

* To find out which thread is running the code we can use

Thread t=Thread.currentThread();
t1=sleep(3000);

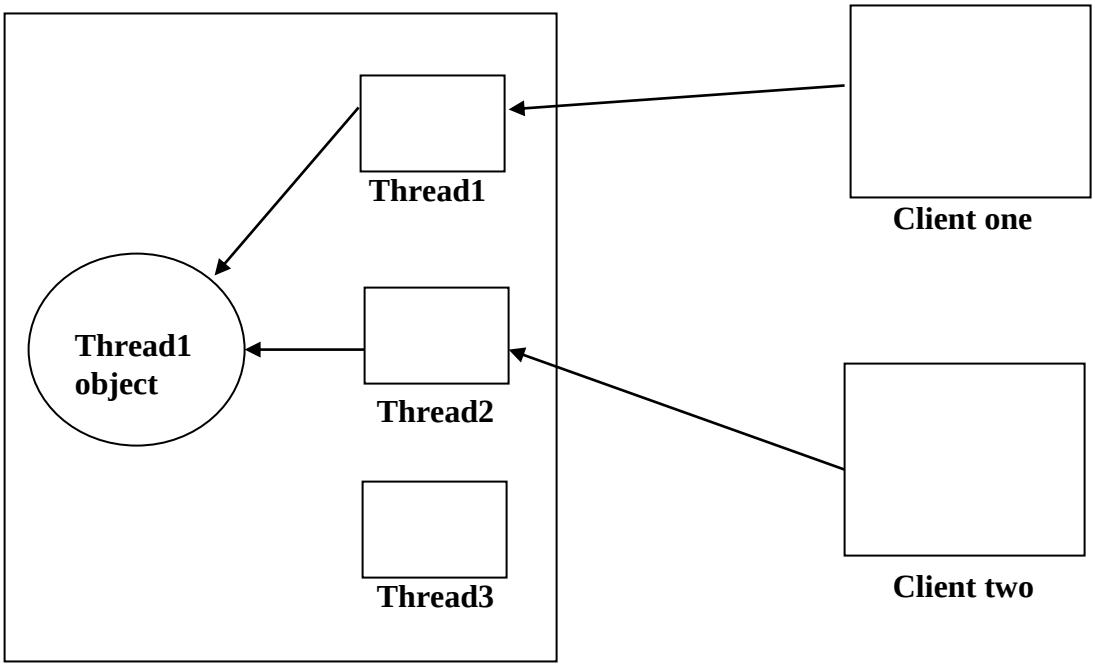
* Thread is a static method in Thread class.

Ex:

```
import javax.servlet.*;
import javax.servlet.http.*;
```

```
import java.io.*;
public class thread1 extends HttpServlet
{
  PrintWriter out;
  public void init()
  {
   System.out.println("Thread1 objected created.....");
  }
  public void service(HttpServletRequest request,HttpServletResponse response) throws
ServletException,IOException
  {
   out=response.getWriter();
   try
   {
    out.println("Line one thread1");
    System.out.println("Line one thread1");
    Thread t=Thread.currentThread();
    System.out.println("thread-->"+t);
    t.sleep(9000);
   }
    catch(Exception e)
    {
    }
     System.out.println("end of thread");
     out.println("end of thread");
 }
    public void destroy()
    {
     System.out.println("Thread1 destroyed.....");
    }
}
```

* Every Web container create a Threads Internally.



Web Container

* When a client sends a request the web container starts the execution of the service method from a thread, if two clients sends the request concurrently the web container will use two different threads & execute the service method from thread1 as well as from thread2 on the same servlet object.
* If 10 clients are requested concurrently in server but server side only one servlet object will be created & it will responsed & ikt creates multiple threads.(this point just for reference written by me).
* When ever an object is used concurrently from multiple threads to avoid problems we need to follow some guide lines.
* In the above servlet we declared **PrintWriter out** as a local variable, in this case there will be no problem even if multiple request are sernt concurrently to the servlet.

If we declare the **out** variable as **instant** variable the servlet may fail. If multiple request are sent to the servlets.

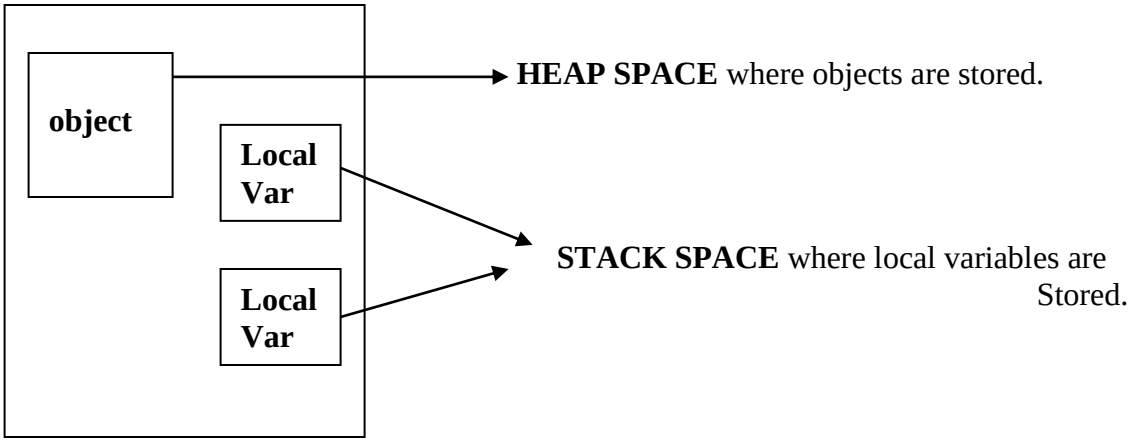
* It is better to avoid instance variables as part of servlets.

Ex:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class thread3 extends HttpServlet implements SingleThreadModel
{
  PrintWriter out;
  public void init()
  {
   System.out.println("Thread3 object created.....");
  }
  public void service(HttpServletRequest request,HttpServletResponse response) throws
ServletException,IOException
  {
   try
   {
    System.out.println("Line one thread1");
    out=response.getWriter();
    out.println("Line one thread1");
    Thread t=Thread.currentThread();
    System.out.println("thread-->"+t);
    t.sleep(9000);
   }
   catch(Exception e)
   {
   }
     System.out.println("Line two thread1");
     out.println("line two of thread1 ");
  }
```

```
    public void destroy()
    {
     System.out.println("Thread3 destroyed.....");
    }
}
```

* When SingleThreadModel interface is implemented the web container will use a **single thread** at any point of time to access a servlet object.
* If we implement **SingleThreadModel** interface the web container will create multiple servlet objects when two browsers sends the request to the servlet, the web container uses two different threads & executes the service method on two different servlet objects.
* Even though we can solve the problem with the instants variables using **SingleThreadModel** interface it is better not to use this interface When this interface used the server creates multiple objects which takes more amount of memory space.
* SingleThreadModel will not solve the problem with objects stored inside & servletContext.
* When we start running a java application the **jvm** allocates some amount of memory space for storing the java objects this space is called as **HEAP SPACE**.
* For every thread the **jvm** allocates some amount of memory space to store **the local** variables, this space is called as **STACK SPACE**.



Ex : student.java

```
public class student
{
 int age;
 String name;
 public void setAge(int a)
 {
  age=a;
 }
 public void setName(String s)
 {
  name=s;
 }
```

```
}
```

gstudent.java

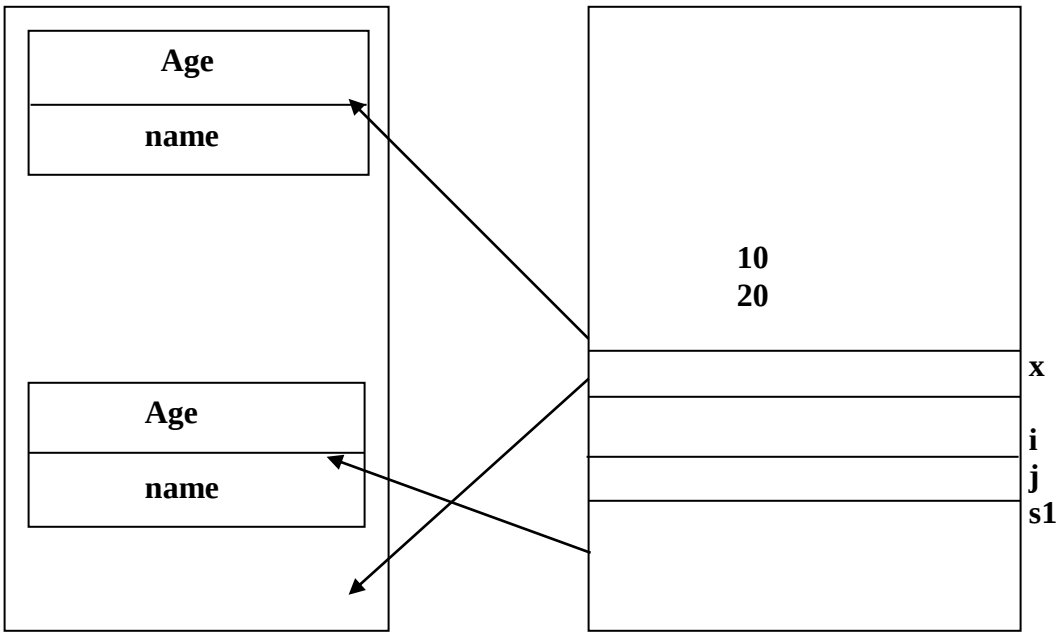
```
public class gstudent
{
 public static void main(String args[])throws Exception
 {
  int i=10;
  int j=20;
  student s1,s2,s3;
  s1=new student();
  s2=new student();
  s3=s1;
  String x=new String("sudhi");
  System.out.println(x);
  System.out.println(s2);
  System.out.println(s1);
  System.out.println(s3);
  }
}
```

Output is :

sudhi
student@253498
student@1fef6f
student@1fef6f

`

* Here only two objects are created that is **s1,s2**.

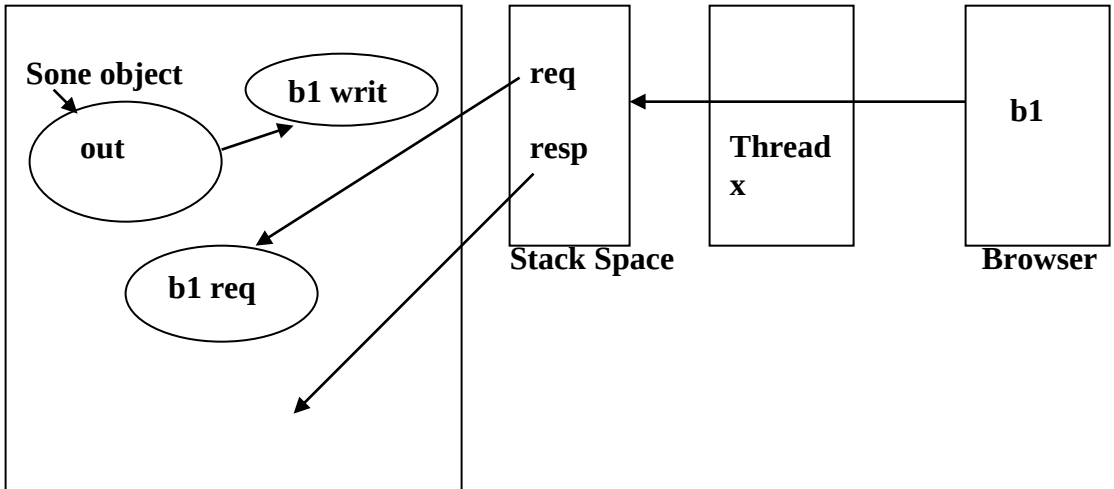


Ex:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class thread1 extends HttpServlet {
  PrintWriter out;
  public void init()  {
   System.out.println("Thread1 objected created.....");
  }
  public void service(HttpServletRequest request,HttpServletResponse response) throws
ServletException,IOException  {
   out=response.getWriter();
   try   {
    out.println("Line one thread1");
    System.out.println("Line one thread1");
    Thread t=Thread.currentThread();
    System.out.println("thread-->"+t);
    t.sleep(9000);
   }
    catch(Exception e)  {  }
     System.out.println("end of thread");
     out.println("end of thread");
 }
    public void destroy() {
     System.out.println("Thread1 destroyed.....");
    }}
```

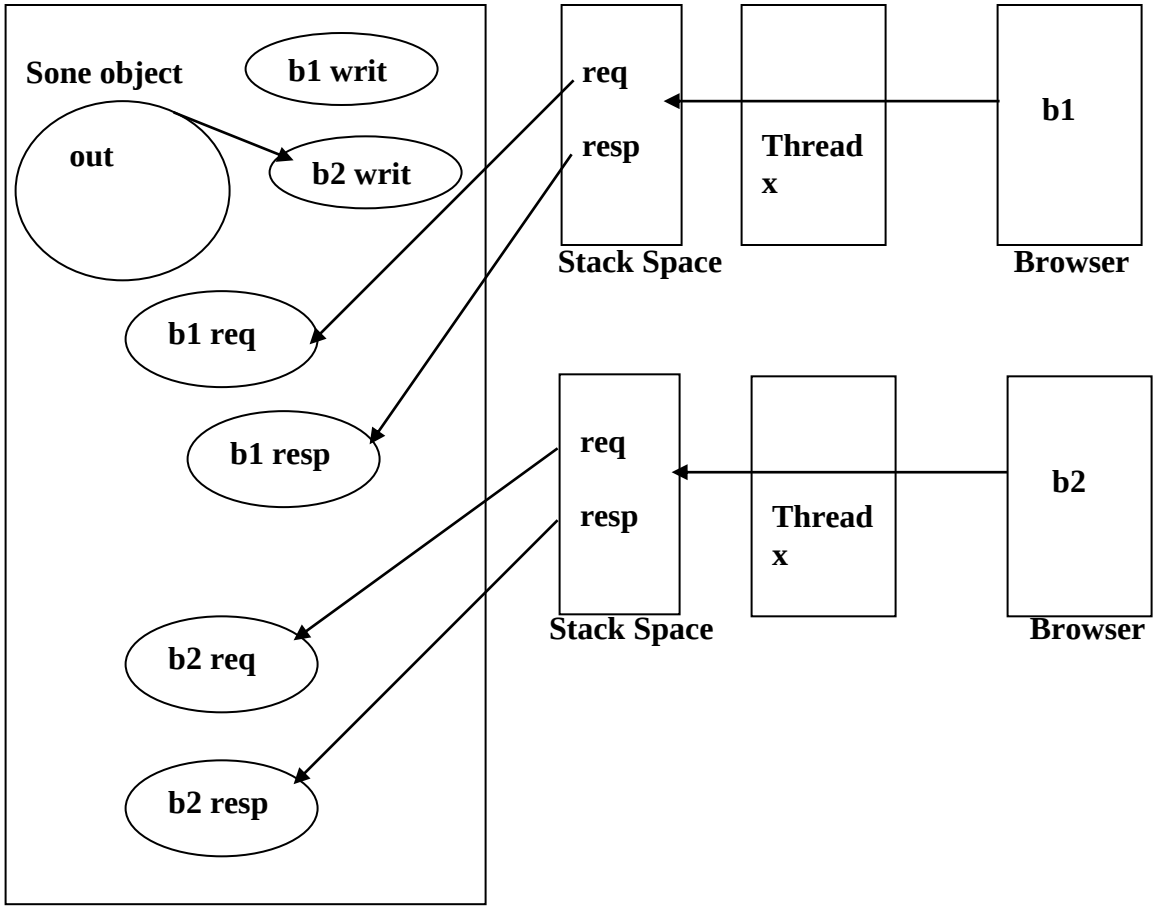
* I n the above Example we write **PrintWriter out** as a instant variable.
* If we use multiple browsers like **b1 , b2** First we send the request using **b1** then object create & thread may execute, like wise if we send request on **b2** then thread may execute, Here problem occur the problem is if we send request on **b1,b2** again **b1** then it display the two times **b2** answer on **b1** browser.

* If we send **b1** request then first time what happened it is shown in below diagram.



Heap Space

* If send the Request on second Browser then what happened it is shown in below diagram.



* In the above figure the we send second request on **b2** then **sone** object will be link to **b2 write** object and it will disconnect to the **b1 write** object, it will be collected to **Garbage values**.

* If we are force to use an instant variable we must used **synchronized** blocks to access the object pointed by the instance variable.

Ex :

```
Public class MySrv extends HttpServlet
{
  Student s;
  public void init()
  {
    s= new student();
  }
   public void destroy()
```

```
  {
   s=null;
  }
   public void service(…….)
   {
     ------------------------------
    ------------------------------
    ------------------------------
    synchronized(s)
    {
      Invoke the methods on obj pointed by s.
    }
  }
}
```

* It is always advisable to use synchronized blocks file accessing the objects stored inside **ServletContext & HttpSession** object.

Ex :

```
Student m = (student)application.getAttribute("sinfo");
Synchronized(m);
 {
  ------------------
  ------------------
 }
}
Public class stwo extends HttpServlet
{
  Student x=(student)application.getAttribute("sinfo");
  Synchronized(x);
  {
 --------------------------
--------------------------
 }
}
```

* Any java object which can be access from multiple threads without any side effect then we can say the object is thread safe.

* **To make a servlet as thread safe servlet we must follow the rules given below**

1. Avoid using instance variable ie prefer to using local variables.
2. If we have to use the instance variables we must access the object referencd by the instance variable by using synchronized blocks.
3. We must use synchronized blocks while accessing objects stored inside servlet context.

* Vector & array list provides similar kind of facilities but the methods in the vector are synchronized but the methods in array list are not synchronized.
* We can safely access a vector object multiple threads concurrently, but there will be small over head to execute a synchronized method. As a array list methods are not synchronized the over head of executing the synchronized method is eliminated , but we can't safely access array

list from multiple threads to access array list safely from multiple threads we must use synchronized blocks.

JNDI Connection pooling:

* We can obtain the connection directory using **DriverManager.getConnection()** but the performance of the web application will be effect it we get the connection directly to improve the performance we can use the connection from the connection pool.
* To access the directory server running in the weblogic server from a servlet we must first get the initial context for this we need not setup the properties we can directly get the initial context as shown below.

Context ic =new InitialContext();

Ex :
First method:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.*;
public class jndi1 extends HttpServlet
{
 public void service(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
 {
  PrintWriter out = response.getWriter();
  try
  {
   Context ic=new InitialContext();
   DataSource ds=(DataSource)ic.lookup("info.sudhi.orapool");
   Connection con=ds.getConnection();
   //execute sql stmts according to the rq.
   con.close();
  }
   catch(Exception e)
   {
    System.out.println("problem....."+e);
   }
 }
}
```

* The above Servlet instead of directly getting the connection using DriverManager.getConnection() it is getting the connection from the connection pool, this reduces the time to get the connection & improves the performance.
* In the above code we are always getting the Data source in the service method, we can obtain the Data Source from in the **init()**.

Ex :

Second method :

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import javax.naming.*;
import java.sql.*;
import javax.sql.*;
public class jndi2 extends HttpServlet
{
 DataSource ds=null;
 public void init()throws ServletException
 {
  try
  {
    Context ic=new InitialContext();
    DataSource ds=(DataSource)ic.lookup("info.sudhi.orapool");
  }
   catch(Exception e)
   {
    System.out.println("problem....."+e);
   }
  }
    public void destroy()
    {
     ds=null;
    }
    public void service(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
    {
     PrintWriter out = response.getWriter();
     try
     {
      synchronized(ds)
      {
       Connection con=ds.getConnection();
      //execute sql stmts according to the rq.
      con.close();
      }
     }
     catch(Exception e)
     {
      System.out.println("problem....."+e);
     }
   }
 }
```

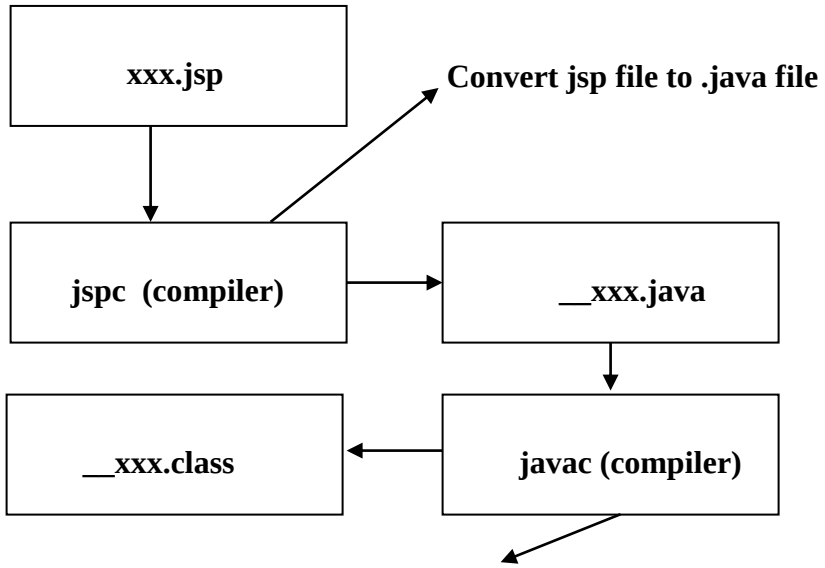
JSP (*Java Server Pages*)

* Java Server Pages is a design to simplify the development of a web application, performance wise an application develop using servlet will be slightly better then the application developed using **jsp**, but the amount of time that we need to spend in developing an application using servlet will be more then the time we need to spend in developing **jsp** application.
* **jsp** technology is based on **servlet** technology a developer can assume that the **jsp** is same as a servlet.
* As part of every container we get a **jsp** compiler.
* In case of weblogic we get a class **weblogic.jspc** which is called as jsp compiler.

Ex:

one.jsp

```
<html>
<head><title> welcome to Jsp</title>
<body bg color=gangadher>
  This is my first Jsp Page
</body></head>
</html>
```



* When we place a **jsp** file on a web container the web container take care of **jsp** compiles.

* If we take a manual compilation using Weblogic container we use following command.

C:\>java weblogic.jspc -keepgenerated one.jsp

* Then the **jsp** compiler first create **.java** file then java compiler create **.class** file on that folder.

* **-keepgenerated** if we use then we also seen **.java** file also.

* If we use **Tomcat server** so we find in .java & .class files it is in **Tomcat5**

C:\Program Files\Apache Group\Tomcat 5.25\work\catalina\localhost

(or)

* If use **Tomcat4.1** then we can see in

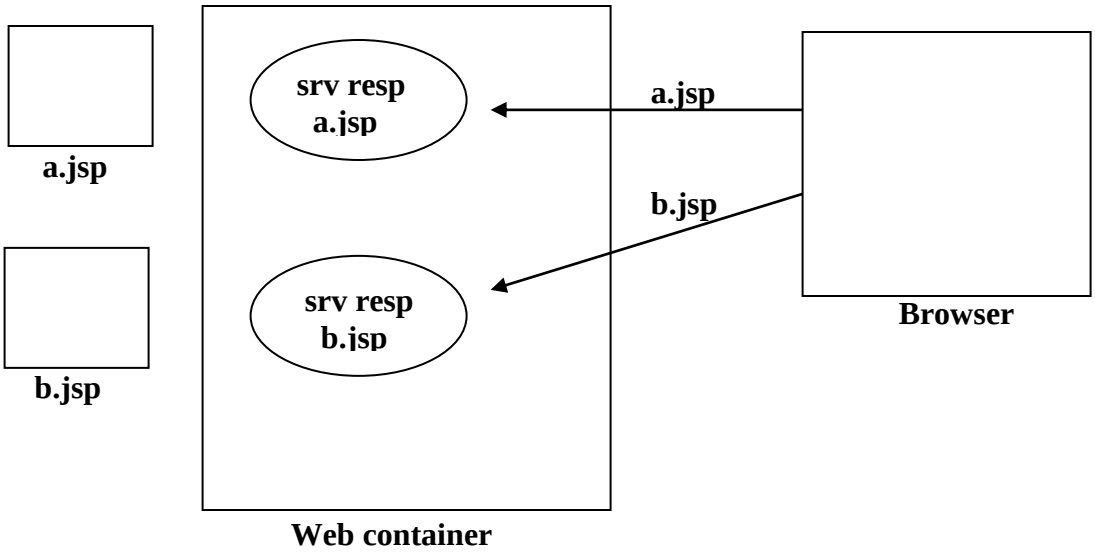
C:\Program Files\Apache Group\Tomcat 4.1\work\Standalone\localhost

* If we use **weblogic server** so we find in .java & .class files is

C:\bea\user_projects\sudarshan\myserver\.wlnotdelete_appsdir_jsp_dir_jsp_4536194\jsp_servlet

Note: if our program is successfully executed then only **.class** file will be appear or if our program not successfully executed then only **.java** file will be appear.

* When we send a request for a **.jsp** file, the web container (if required) runs the jsp compiler & generates **.java** file this **.java** file contains the code for a servlet class, this will be converted to a **.class** file by the java compiler, after this the web container creates a servlet object & calls the appropriate methods.
* Most of the web containers in the default setup regenerates the servlet after the **.jsp** file is modified.



**Line one
** ⟶ **temlate text**
<%
System.out.println(“sudhi”); ⟶ **SCRIPTLET (java code)**
%>

* We can use **html** tags ,**wml** (wireless markup languages) tags, **xml** tags or **plain** text etc as part of template text this portion will be send to the client as it is.
* As part of the **jsp** we can add code as shown below.

<%
Scriptlet code
%>
This is called scriptlet.

* Theoretically according to the **jsp** specification any language can be used to provide the code as part of the scriptlet but most of the servers supports only java language as part of scriptlet.
* We must declare a variable java before we use it.

```
<%
        String s1= "xxx",s2="yyy";
        System.out.println(s1.equals(s2));
        System.out.println(s3.equals(s2));
%>
```

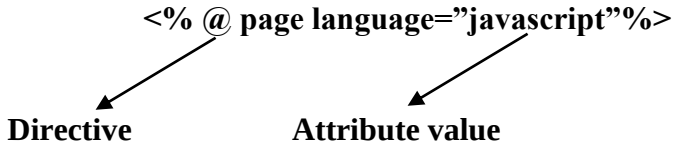
* The above **jsp** file will be converted as **.java** file but the java compiler can't convert it to the **.class** file as we are using **s3** with out declaring it.
* As part of **jsp** specification a set of variables are specified as **implicit** variables and implicit variable is a variable which can be used with out declaring in the java scriptlet, The implicit variables are out (**jsp writer)** equalent to servlet PrintWriter.
* In jsp writer class is similar to servlet PrintWriter.

request
response
config
application
session
page
pagecontext are implicit variables.

* Page is a variable which represent a current servlet object.

```
One<br>
<%
        out.println("----one<br>");
        out.println("----one<br>");
%>
```

* In the above scriptlet we are using out variable without declaring it.



* By writing this we are telling the jsp compiler that as part of javascript.

* As part of jsp files we can add various directives for **ex:**

```
<% @ page language="java" %>
<%
        out.println("can u see this<br>");
%>
```

* A directive is an instruction that will be used by the jsp compiler while generating the servlet.

Ex :

```
 <% @ page language="java"%>
<% @ page session="false"%>
<%
```

```
            String id=session.getId();
            out.println("sid→"+id);
    %>
```

* We can't use a session object after using the directive

```
    <% @ page session="false"%>
```

* As part of jsp's we can use various types of elements like template text, Directives , scriptlet, Declarative statement, jsp expressions, jsp action tags & jsp custome tags.

Ex:

A program which gets data from emp table

```
<%@ page import="java.sql.*"%>
<%@page import="java.io.*"%>
```

(or)

```
<%@page import="java.sql.*,java.io.*"%>
```

* As part of the jsp's if we need to use a class (or) an interface that is not part of java.lang.package we can use the page directive with import attribute.

Getdata.jsp

```
<%@ page import="java.sql.*"%>
<%@ page import="java.io.*"%>
<h1><b>Here is the Data from emp table</b></h1> <br>
<%
    Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
    Statement stmt=con.createStatement();
    ResultSet rs= stmt.executeQuery("select ename from emp");
    while(rs.next())
    {
        out.println(rs.getString("ename"));
        out.println("<br>");
    }
     con.close();
%>
```

* The out put is

Here is the Data from emp table

SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE

CLARK

* If we use **weblogic server** so we find in .java & .class files is

C:\bea\user_projects\sudarshan\myserver\.wlnotdelete_appsdir_jsp_dir_jsp_4536194\jsp_servlet

* In the above program is successfully executed then only **.class** file will be appear or if our program not successfully executed then only **.java** will be appear.

* If we use **Tomcat server** so we find in .java & .class files is

C:\Program Files\Apache Group\Tomcat 4.1\work\Standalone\localhost\jsp

* In the above program is successfully executed then **.java & .class** file will be appear or if our program not successfully executed then only **.java** will be appear.

***** It is not recomonded to develop jsp's using java scriptlet as in the above example.
* **The reasons for this are :**
1. A programmer who have no knowledge in java can't develop a jsp based application, if java code is used in jsp's.
2. JSP technology is designed to simplify the separation of business logic from the presentation logic , a developer tends to add business logic as part of jsp if he uses java scriptlets as part of jsp's.

Note : java soft strongly recommends using presentation logic only as part of jsp's.

3. In the above jsp we have hard coded jdbc driver ,URL etc. this can be eliminated using initialization parameters etc.

* As part of **web.xml** we can provide the initialization parameter in a jsp by configuring the jsp written in web.xml as shown below.

```
<web-app>
        <servlet>
                <servlet-name>MyJsp</servlet-name>
                <jsp-file>one.jsp</jsp-file>
                <init-param>
                        <param-name>pone</param-name>
                        <param-value>some value</param-value>
                </init-param>
        </servlet>
        <servlet-mapping>
                        <servlet-name>MyJsp</servlet-name>
                        <url-pattern>/jspurl</url-pattern>
        </servlet-mapping>
</web-app>
```

* Most of the developers will not add the information about the jsp's to **web.xml** as shown above.

Ex : using sessions

1.login.html

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>
  <head>
    <title>Sudarshan login page!!</title>
  </head>
  <body bgcolor=gangadher>
    <font face="helvetica">
<pre>
    <form action="Sone.jsp" method=GET>
    Student Name    <input type=text name=sname> <BR>
      Father Name   <input type=text name=fname>

   <BR><BR>                    <input type=submit  value = login>
    </form>
   <pre>
   </font>
  </body>
</html>
```

2. Sone.jsp

```
<%
        String vsname,vfname;
        vsname=request.getParameter("sname");
        vfname=request.getParameter("fname");
        session.setAttribute("asname",vsname);
        session.setAttribute("afname",vfname);
%>
<html>
  <head>
    <title>Sudarshan login page!!</title>
  </head>

  <body bgcolor=gangadher>
    <font face="helvetica">
<pre>
    <form action="Stwo.jsp" method=GET>
    Student Age       <input type=text name=sage> <BR>
      Address       <input type=text name=saddr>

   <BR><BR>                    <input type=submit  value = login>
    </form>
```

```
  <pre>
  </font>
 </body>
</html>
```

2. Stwo.jsp

```
<%
        String dsname,dfname,vsage,vsaddr;
        vsage=request.getParameter("sage");
        vsaddr=request.getParameter("saddr");
        dsname=(String)session.getAttribute("asname");
        dfname=(String)session.getAttribute("afname");
        System.out.println("Code to store data");
        out.println("<br>"+dsname);
        out.println("<br>"+dfname);
        out.println("<br>"+vsage);
        out.println("<br>"+vsaddr);
%>
```

* And another Ex :

1. Three.jsp

```
 <%     System.out.println("executin one");
        response.sendRedirect("four.jsp");
%>
```

2. Four.jsp

```
<%
        System.out.println("Executing two");
        out.println("Executing two");
%>
```

* When the Browser sends the request for **three.jsp** , the webcontainer executes **three.jsp** & sends a response to the browser indicating that browser must send another request for **four.jsp** (this is because of using **response.sendRedirect**) after this the browser sends the second request to the container for the resource **four.jsp** finally the browser gets the output ie generated **four.jsp**.

* **JSP expressions (or) java expressions :**

```
<%
        int i=10, j=20;
        out.println(i);
        out.println(j);
        out.println(i+j);
%>
```

* The above jsp can be rewritten using jsp expressions as shown below.

```
<%
        Int i=10, j=20;
%>
<%=i%>
<%=j%>
<%=i+j%>
```

out.println → <%

Ex:

```
        <%
                String s1="aaa";
                String s2="aaa";
                String s3="bbb";
        %>
        <%=s1.equals(s2)%> <br>
        <%=s1.equals(s3)%> <br>
        <%=new java.util.Date()%> <br>
        <%=session.getId()%> <br>
        <%=request.getHeader("User-Agent")%> <br>
```

* Most of the developer started using **EL**(expression language) expressions in place of jsp expressions the **EL** expressions are similar to the expressions used in most of the scripting languages & it is easy to learn these expressions for a developer having experience with scripting languages.

```
public class MySrv extends HttpServlet
{
  int add(int i, int j)
 {
    int res=i+j;
    return res;
}
--------service(--------)
{
  Int x=add(10,20);
}
}
```

* In the above servlet we have implemented add method which is used from the service method, in **jsp** also we can implement our own methods as part declarative statements as shown in the below example

```
<%!
        int add(int i,int j)
        {
         return i+j;
        }
%>
 Using declarative staements <br>
<%
        System.out.println("before calling add");
        int x=add(10,20);
        System.out.println("After calling add");
        out.println(x);
%>
```

* As part of declarative block a part from writing the methods we can declare the variables, these will be added as instant variables in the servlet, the variables will be added as static variables of the servlet , if we use static key board in the declaration variable.

The variable is declare in scriptlet will be added as **local variables** of the **servlets_jspService(…..)**.

* Declarative statements placed any where in the program either starting (or) ending.

Ex : gd.jsp

```
<%!
        int vone=22;
        void sm()
        {
                int vtwo=44;
                System.out.println("vone--->"+vone);
                System.out.println("vtwo--->"+vtwo);

        }
%>
 Using declarative staements <br>
<%
        int thr=44;
        System.out.println("vone--->"+vone);
        System.out.println("vthr--->"+thr);
        out.println("<h2><b>vone--->"+vone);
        out.println("<br>vthr--->"+thr);
        sm();
%>
```

* When the jsp compiler compiles the jsp page it add the scriptlet as part of **_jspService()** so when the below we added as a local variable of **_jspService(….)**.

* As part jsp file where we can add java code?
Ans: We can add java code in jsp **Declarative & scriptlet**.

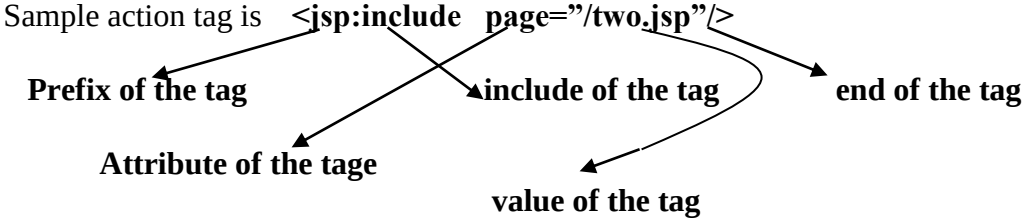
* According to the jsp specification the jsp compiler are responsible for adding the **implicit** variables **request,response** as parameters of **_jspService(…….)** the remaining implicit variables has to be added as local variables of service method.
* As part of Declarative statements we can add **jspInit()** & **jspDestroy()**.

Ex : service.jsp

```
<%!
        public void jspInit(){
                System.out.println("Executing jspInit()");}
        public void jspDestroy(){
                System.out.println("Executing jspDestroy()");}
%>
<%
        out.println("<h2><B>Executing  _jspService()");
%>
```

* To simplify the development of **jsp's**(to eliminate java code) as part of **jsp1.0**
jsp action tags are provided.

Sample action tag is **<jsp:include page="/two.jsp"/>**



* When we use **jspInclude** internally the jsp compiler generates the code that makes use of **RequestDispatcher.include()** in the above example when we send the request to **one.jsp** the web container executes the servlet corresponding to one one.jsp as well as the servlet corresponding to **two.jsp** .
* i.e the browser sends only one request but the web container will execute two servlets.
* When we use **jspForward** tag internally RequestDispatcher.forward() will be used.

* In the same we have another method also i.e **<%@ include file="/two.jsp"%>**

* What is diff between jspInclude tag **<jsp:include page="/two.jsp"/>** & include directive tag **<%@ include file="/two.jsp"%>** ?

```
Line one of one.jsp<br>
<%@ include file="/two.jsp"%>
Last line of one.jsp<br>
```

one.jsp

```
<br>o/p of two.jsp<br>
```

* When the jsp compiler compiles the above one.jsp it will replace the **include** directive with the content of two.jsp as shown below after this merging the jsp compiler will generate a single servlet.

two.jsp

```
Line one of one.jsp<br>
<br>o/p of two.jsp<br>
Last line of one.jsp<br>
```

* The performance of the application can be improved slightly by using include directive.

* In the above concepts

Ex : use jsp action tags:

1. done.jsp

```
<h2><b>line one of done.jsp</b></h2><br>
<jsp:include page="/dtwo.jsp"/>
<%
        out.println("<h2><b> output of done.jsp");
%>
```

2. dtwo.jsp

```
line one of dtwo.jsp<br>
<%
        out.println("<h2><b> output of dtwo.jsp");
%>
```

output is :

```
line one of done.jsp

line one of dtwo.jsp
output of dtwo.jsp
output of done.jsp
```

Ex: use Directive tag:

1.cone.java

```
<h2><b>line one of cone.jsp</b></h2><br>
<%@ include file="/ctwo.jsp"%>
<%
        out.println("<h2><b> output of cone.jsp");
%>
```

* the above one creates two **.java** files ie done.java,dtwo.java & two **.class** ie done.class, dtwo.class

2.ctwo.java

```
<h2><b>line one of ctwo.jsp</b></h2><br>
<%
        out.println("<h2><b> output of ctwo.jsp");
%>
```

output is :
line one of cone.jsp
line one of ctwo.jsp
output of ctwo.jsp
output of cone.jsp
* the above creates only one .java file and one .class file because of it is merged to that files.

* Is **thread safe** can be used as as attribute for page directive the default value of this is **true**.

<%@ page isThreadSafe="false"%>

* If we specify the above one the servlet generated by **jsp** compiler will implement the interface single thread model.
* When we specify the page directive with **info** attribute

<%@ page info="this jsp is for…."%>

* The jsp compiler generates the servlet with **getServletInfo** method(public String **getServletInfo()**).
* We can develop a jsp exclusively for handling the errors these kind of pages are known as error pages.

Ex : ep.jsp

```
<%@ page isErrorPage="true"%>
<%
 System.out.println("Executed ep.jsp");
%>
<h2><b>Due to some broblem.....<br>
<%=exception%>
Please send a mail to our admin.....
```

* **exception** is an implicit variable it is only to give an exception.
* We can the implicit variable exception only in the error pages will be executed, only there is an some error to run the jsp pages.
* The error pages will be executed when there is a problem in executing the other **jsp's**

Ex : Error page display

err.html

```
<html>
<head><title>ErrorPage</title>
</head>
<body bgcolor=gangadher>
<form method=GET action="at.jsp">
```

```
<pre>
Student              <input type=text name=sname>
Father         <input type=text name=fname>
<BR><BR>                    <input type=submit  value = login>
</pre>
</body>
</html>
```

l.at.jsp

```
<%@ page isErrorPage="false"%>
<%@ page errorPage="ep.jsp"%>
<%
        System.out.println("Exceuting at.jsp.....");
        int i=10,j=0,k;
        k=i/j;
        System.out.println("---------------");
%>
```

2. ep.jsp

```
<%@ page isErrorPage="true"%>
<%
 System.out.println("Executed ep.jsp");
%>
<h2><b>Due to some broblem.....<br>
<%=exception%>
Please send a mail to our admin.....
```

Ans is : *Due to some broblem.....*
java.lang.ArithmeticException: / by zero Please send a mail to our admin.....

* When the client send the request to **at.jsp** the web container will start executed successfully the output generated by **at.jsp** will be send to the client but if the execution of **at.jsp** fail due to an Exception the web container discards the output generated by **at.jsp** and starts executing **ep.jsp** .
* jsp's uses jspWriter instead of PrintWriter to send the content to the browser.

* **Difference between PrintWriter & jspWriter ?**

* If we develop our own servlet then we use **PrintWriter**.
* The methods print ,println,write of PrintWriter class are not implemented to throw an Exception(ie When we call this methods no exception will be thrown even if the method fails due to some problem) but the methods in **jspWriter** are implemented to throw IOException when they failed.

* jspWriter supports buffering.

Ex : buf.jsp

```
<%@ page buffer="2kb"%>
<%
        for(int i=0;i<1000;i++)
        {
                out.println("aaaaaaaaaa<br>");
        }
%>
```

* Then above jsp generate experimentally 14000 bytes.
* When we send the request to the above jsp internally **2kb** of memory will be allocated, this area is called as buffer area the output ie generated by the jsp will not be sent directly to the client it will be first stored in the buffer.
* In the above we set the buffer is **2kb** so it is stored the maximum capacity **2kb** then if **2kb** is full then it will send to the browser after that it will store next **2kb of data** it is full send to the browser it is continues process finally buffer is **7** times is full in the above program.
* When the loop is executed for about **146** times approximately **2kb** of data will be generated & it will be stored in the buffer as more amount of data can't be accommodated the buffer will flushed (what ever generated till that point of time will be sent to the client & the buffer will be free now) .
* **What is Buffer over flow how to solve in jsp?**
* If we specify **autoFlush="false"** the buffer will no be flushed after it is full at this stage if more output is generated an Exception will be thrown indicating buffer over flow.

<%@ page autoFlush="false"%>

* Buffer over flow can be solved by either increasing the buffer size or setting

<%@ page autoFlush="true"%>

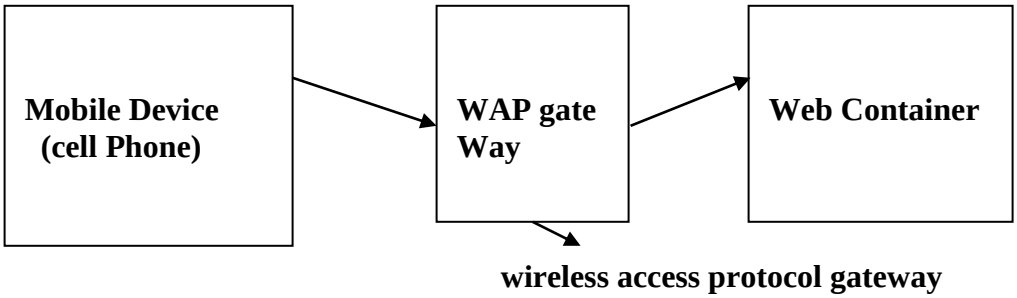
* content type can be used to specify the type of content that is generated by the jsp.

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

<%@ page contentType="text/xml"%>

* In mobile technology we use

<%@ page contentType="text/wml"%>



* Here we using wml tags

* We implement any thing in jsp so we have to use suppose we implement **xxx** to use

<%@ page extends="xxx"%>

* If autoFlush is set to true there will be no problem like buffer overflow in some cases we need to set **autoFlush=false** and a high value for buffer.

Ex : buff.jsp

```
<%@ page buffer="2kb"%>
<%@ page errorPage="ep.jsp"%>
<%
        for(int i=0;i<1000;i++){
                out.println("<h2>aaaaaaaaaa</h2><br>");
        }
                int z=10,m=0,k;
                k=z/m;
                out.println("<h2>bbbbbb<br>");

%>
```

* In the above page if we reduce the buffer size we will get partial output generated by **buff.jsp & ** the output of **ep.jsp**, if **buff.jsp** fails to see only the output generated by **ep.jsp** when **buff.jsp** fails we have set the buffer size to 14kb.
* As part of jsp1.1 taglib directive is provided this can be used as part of a jsp to use custom tag libraries.
* Several physical products like computer , fans tube lights etc are manufactured by taking various parts(computers) and assembling them by using the tools, the components used in a product need not be manufactured by a single company even through various components are manufacture by various companies we are able to assemble them together as they are develop the manufacture according to set of guidelines (or) standards.
* absorving the manufacturing of physical products company's like javasoft , microsoft etc has designed the specifications (guidelines) to develop a piece of software (component) which can be used (reused) along with other components and assemble a software application.
* There are two different specifications from javasoft which can be used to develop the component.
    1. javabean
    2. enterprise javabean
* COM is specification from Microsoft .

Ex: of company one develops a piece of software (component) using with javabean specification & company two develops another component using javabean specification we can develop our own application from these two companys.

* Java soft has develop the classes like jframe ,jbutton,jlabel etc according to the javabean specification so we can call jbutton etc as javabean.
* javabean's can be used to develop GUI as well as nonGUI components.
* A javabean can support a set of properties, events and some other additional methods.
* A javabean can support a property **xxx** by provi ding by methods **getXxx** & **setXxx**
* We need not provide both the methods to support a property.

Ex :

```
public class MyBean
{
```

```
        private String n;
        private int a;
public void setName(String n)
{
        N=x;
}
public String getName()
{
        return n;
}
public void setAge(int a)
{
        return a;
}
}
```

* In the above class is a javabean supporting two properties
    1. name
    2. age
* Name property is a type of String & age property is of type int.

```
public class YourBean
{
        private String empName;
        public void setEmpName(String empName)
        {
                this.empName=empName;
        }
        public void String getEmpName()
        {
                return empName;
        }
}
```

* The above class is a javabean supporting the property empName & it is type of string.
* If a property is Boolean property it is better to provide **isXxx()** instead of **getXxx()** as shown in the below example.

Ex :

```
public class Student
{
        private Boolean male;
public void setMale(String male)
{
        This.male=male;
}
public boolean isMale()
{
        return male;
}
}
```

* If the javabean has to support an event call actionEvent we most provide the methods addActionListener()

* We can use the tags jsp:useBean , jsp:setProperty , jsp:getProperty to deal with java beans from jsp's.

Ex : Student.java

```
package info.inetsolv;
public class Student
{
 private int age;
 private String name;
 private String address;
 public Student()
 {
  System.out.println("Student object created.....");
 }
 public void setName(String n)
 {
  this.name=n;
 }
  public void setAge(int a)
  {
   this.age=a;
  }
   public void setAddress(String b)
   {
    this.address=b;
   }
    public String getName()
    {
     return name;
    }
    public int getAge()
    {
     return age;
    }
    public String getAddress()
    {
     return address;
    }
    public void storeStudentData()
    {
     //code to store student info in DB.
    }
}
```

* The above program is package it is provided methods.

* The above java bean support Three propertys
    1. name
    2. age
    3.address

* To compile the package the command is **javac -d . Student.java**

2. **ub.jsp**

```
<%
        System.out.println("-------Step one------");
%>
<jsp:useBean id="varone" class="info.inetsolv.Student"/>
<%
        System.out.println("Package is imported.....");
%>
<jsp:setProperty name="varone" property="name" value="Sudhi"/>
<%
        System.out.println("Name property is set....");
%>
<jsp:getProperty name="varone" property="name"/>
<%
        System.out.println("Name value get.....");
%>
```

* Here in javabeans we import a package so we have to use
    <jsp:useBean id="varone" class="info.inetsolv.Student"/>

3.**store.html**

```
<html>
<head><title>jspbeans</title>
</head>
<body bgcolor=gangadher>
<form method=GET action="ub.jsp">
<pre>
Student name        <input type=text name=sname>
Student age         <input type=text name=sage>
Student address     <input type=text name=saddress>
<BR><BR>                        <input type=submit  value = store>
</pre>
</body>
</html>
```

* useBean tag when used as part of jsp creates an object based on the class (if required)
* setProperty tag when used as part of jsp internally calls the appropriate setter & get property tag internally calls the appropriate getter & the result will be send to browser (or) client.
* A non java programmer can also learn the tags shown in the above example as there is no java code used as part of jsp action tags.

Note : names of the fields are same as the names of properties of **info.inetsolv.student java bean**

* To retrieve the fields in jsp so we have use different technique as shown in the below.

1.

```
<%
        System.out.println("-------Step one------");
%>
<jsp:useBean id="varone" class="info.inetsolv.Student"/>
<%
        System.out.println("Package is imported.......");
%>
<jsp:setProperty name="varone" property="name"
        value='<%=request.getParameter("sname")%>'/>
<%
        out.println("sname value can be retrived......");
%>
<jsp:setProperty name="varone" property="age"
        value='<%=Integer.parseInt(request.getParameter("sage"))%>'/>
<jsp:setProperty name="varone" property="address"
        value='<%=request.getParameter("saddress")%>'/>
```

* Using jsp expression we are capturing the data and by using **setProperty** tag we are storing this data in the tags as shown below.

```
<%
        System.out.println("-------Step one------");
%>
<jsp:useBean id="varone" class="info.inetsolv.Student"/>
<%
        System.out.println("Package is imported.......");
%>
<jsp:setProperty name="varone" property="name" />

<%
        out.println("sname value can be retrived......");
%>
<jsp:setProperty name="varone" property="age" />

<jsp:setProperty name="varone" property="address" />
```

* The above code can be further simplify writing the code as shown below.

```
<%
        System.out.println("-------Step one------");
%>
<jsp:useBean id="varone" class="info.inetsolv.Student"/>
<%
        System.out.println("Package is imported.......");
%>
```

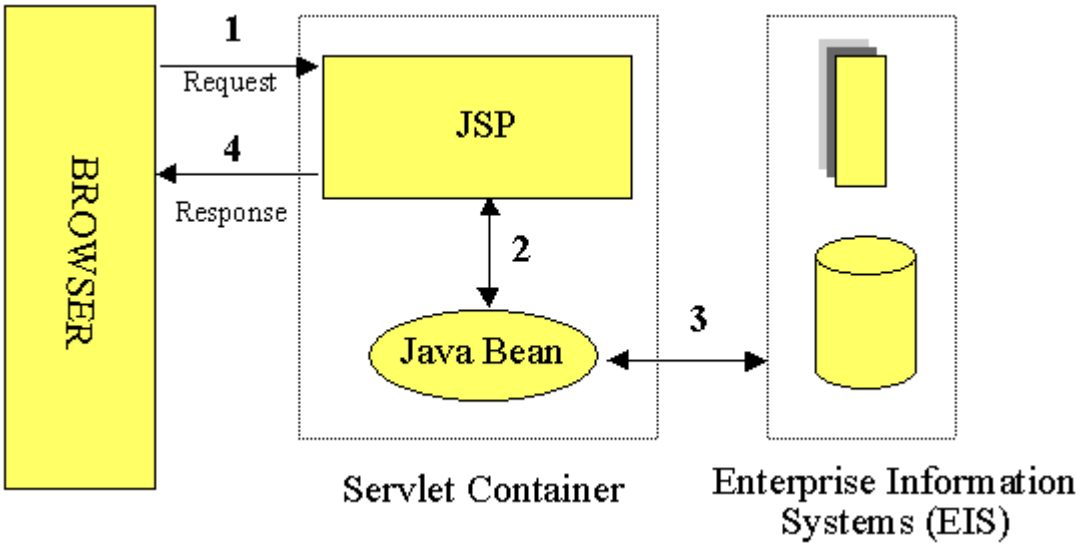
<jsp:setProperty name="varone" property="*" />

* When we use * as a property name all the properties whose names matches with field names in the form will be set.

* When we use * as a property name it will set the values of all the properties of the bean **that matches** the names of fields in the form.

* Java soft has provided two different architecture for the development of jsp based projects. These architectures (design patterns) are called as **jsp model-1** (java soft calls this as **MVC-1**) also for developing small to medium size projects
For medium size to large projects an architecture called as **jsp model-2** (also called as **MVC-2**)

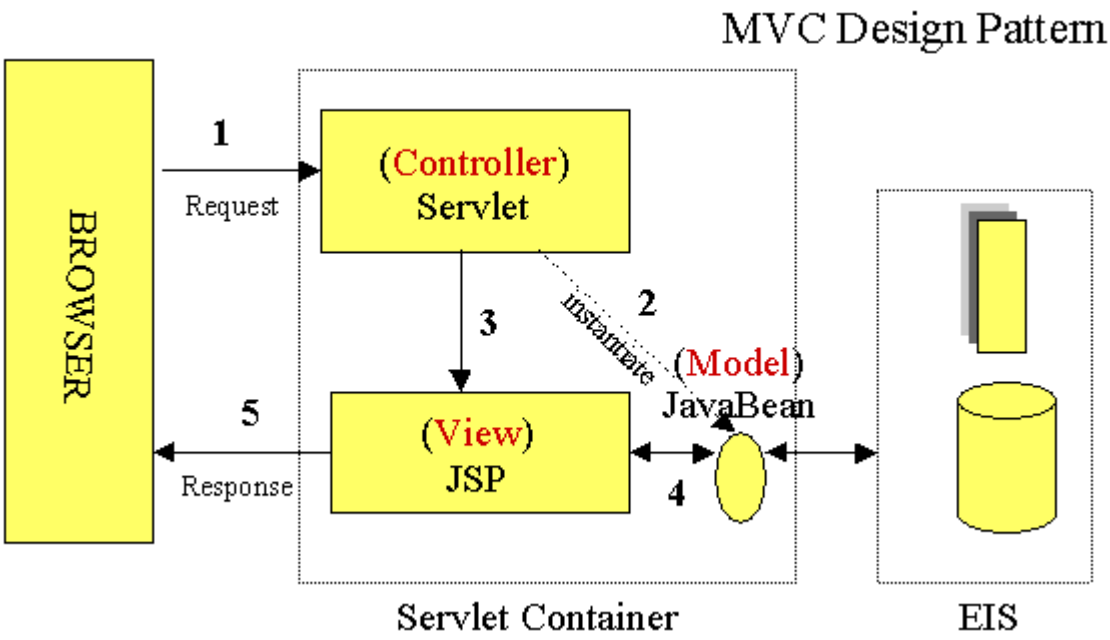
jsp model-1 (or) MVC-1



* If we use this architecture we can separate the business logic from the presentation logic in this model we provide presentation logic as part of jsp & we provide the business logic as part of a java bean.

* jsp model-2 is more complex then jsp model-1 but it offers more flexibility then **model-1**

Jsp model-2 (or) MVC-2



* Structs is design according to the above model ie if we develop a project using structs we can say that we have develop the project using the above model.

Ex : Sample project.

* In data base we create a **stud** table with name , age , address fields.

1. store.html

```
<html>
<head><title>jspbeans</title>
</head>
<body bgcolor=gangadher>
<form method=GET action="ub.jsp">
<pre>
Student name        <input type=text name=sname>
Student age         <input type=text name=sage>
Student address     <input type=text name=saddress>
<BR><BR>                        <input type=submit  value = store>
</pre>
</body>
</html>
```

2.create package in that we have name, age, address set methods & get methods & finally it is stored in data base.

```
package info.inetsolv;
import java.sql.*;

public class Student
{
 private int age;
 private String name;
 private String address;
 public Student()
 {
  System.out.println("Student object created.....");
 }
 public void setName(String n)
 {
  this.name=n;
 }
  public void setAge(int a)
  {
   this.age=a;
  }
   public void setAddress(String b)
   {
    this.address=b;
   }
    public String getName()
    {
      System.out.println("Student name is "+name);
      return this.name;

    }
    public int getAge()
    {
     System.out.println("Student age is "+age);
     return this.age;
    }
    public String getAddress()
    {
     System.out.println("Student address is "+address);
     return this.address;
     }
    public void storeStudentData() throws Exception
    {
     //code to store student info in DB.

          Class.forName("oracle.jdbc.driver.OracleDriver");
       Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
     String sqlstmt="insert into stud values('"+name+"','"+age+"','"+address+"')";
     Statement stmt=con.createStatement();
```

```
    System.out.println(sqlstmt);
    stmt.executeUpdate(sqlstmt);
    con.close();
   }
}
```

* The above method contains the business logic that take cares of storing the student data in the data base.

In this example we provided the business logic as part of a java bean as specified in **jsp model-1**

2. **ub.jsp**

```
<%
        System.out.println("-------Step one------");
%>
<jsp:useBean id="varone" class="info.inetsolv.Student"/>
<%
        System.out.println("Package is imported.....");
%>
<jsp:setProperty name="varone" property="*" />
<h2> Student Name is :
<jsp:getProperty name="varone" property="name"/>
<br>Student Age is :
<jsp:getProperty name="varone" property="age"/>
<br>Student Address is :
<jsp:getProperty name="varone" property="address"/></h2>
<%
        //code to invoke business logic
        varone.storeStudentData();
%>
```

* If the above jsp we have used the **scriptlet** code we can eliminate the scriptlet code by choosing appropriate method names .

*If suppose In the above javabean we can provide **setStudentData(String dummy)** in place of **storeStudentData()**

in this case we can say our javabean supports **Four** properties

1. name
2. age
3. address
4. studentData

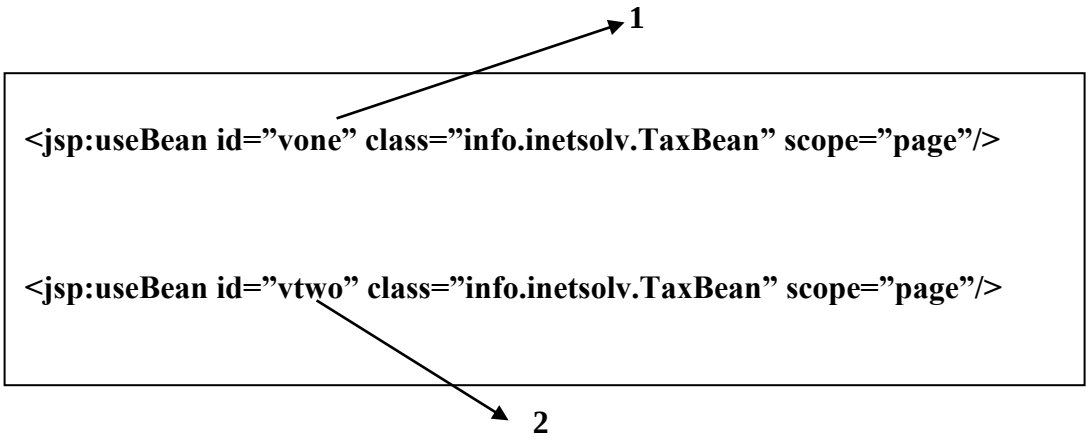
* if we use in **ub.jsp**

<jsp: setProperty name=”varone” property=”studentData” value=”usless value”/>

* By changing the name of the method we are able to eliminate the **scriptlet** from the jsp.
* If the above model is strictly followed a java developer can takes care of implementing the business logic as part of javaBean & even a non-java programmer can implement the presentation logic using jsp.

* As part of the servlets generated by jsp compiler code will be added for creating page context object, this object will be created as part of **_jspService()** & reference of this object will be stored in a local variable i.e every time the jsp is executed a **page context object** will be created & this object will be removed after the execution of jsp.

* **<jsp:useBean id="varone" class="info.inetsolv.TaxBean" scope="page"/>** is equalent to **<jsp:useBean id="varone" class="info.inetsolv.TaxBean" />** i.e the default value for the scope attribute is page .



* When we send the request to the above page the page context object will be created.
* When the first tag is evaluated a java object will be created and it will be placed inside page context object using the attribute name **vone**
* When the second tag is evaluated a java object will be created & it will be store in the page context using the attribute name **vtwo**

The page context object will be removed after the jsp page is executed when it is removed the objects stored inside it will also be removed.

* As a value for scope attribute we can also specify **request , session , application**.

<jsp:useBean id="varone" class="info.inetsolv.TaxBean" scope="request"/>

* If the scope is set as **request** , a **java bean** object will created & it will be stored inside the **request** object. If already the object is there in the request object a new object will not be created.

* What is difference between **scope="page" & scope="request" ?**

Ans:

Ex:

Five.jsp

```
<%
        System.out.println("----------1---------");
%>
<jsp:useBean id="vone" class="info.inetsolv.TaxBean.TaxBean" scope="request"/>

<%
```

```
        System.out.println("----------2----------");
%>
<jsp:include page="six.jsp"/>
```

Six.jsp

```
<%
        System.out.println("----------3---------");
%>

<jsp:useBean id="vone" class="info.inetsolv.TaxBean.TaxBean" scope="request"/>
<%
        System.out.println("----------4---------");
%>
```

* When the client sends the request to **five,jsp** the web container will execute **five.jsp** & **six.jsp** that is for one request two servlets will be executed in the container.
* When we send request to **five.jsp** while executing **five.jsp** a bean object will be created & it will be stored in the pageContext of **five.jsp** as we have used include tag the web container will execute **six.jsp** while executing **six.jsp** a page context object will be created for **six.jsp** & java bean will be created & it will be stored in pageContext of **six.jsp**
* In the above two jsp's if we use **scope="request"** instead of **scope="page"**
* When we send the request to **five.jsp** a request object will be created & the web container will start the execution of **five.jsp** When useBean will be created & it will be stored inside the request object with attribute name **vone**.
* When jsp include is evaluated the web container starts the execution of **six.jsp** by passing the same request object , when useBean tag is evaluated it will not create a new javaBean object.
* If we need to use a javaBean only during the execution of a jsp we must use **scope="page"**
* If multiple jsp's involved in processing a single request has to use the same object then we must use **scope="request"**
* When we specify **scope="session"** the javaBean object will stored in session object we must use the session scope when anything related to client has to be used while processing multiple request.
* To store global data which is not specific to a client we can specify **scope="application"** in this case the javaBean will be stored in servletContext.

* If we use scope=request then we send request every time it will create one useBean object only.
* If we use scope=session then we send request first time only it create useBean object after that it uses that object only.

* As part of jsp1.1 jsp custom tag library facility is provided using this facility any one can provide a tag library.
* A tag library is a set of tags to use jsp action tags we need not use any additional jar files but to use custome tag library we must use
    a. jar files
    b. TLD files (tag library) descriptor.

* After introducing tag libraries as part of jsp's java soft has designed a set of **standard** tag libraries, These are called as Jsp Standard Tag Libraries(JSTL).
* JSTL is implementers of a tag library provides two types of files
    1. JAR files: that contains a set of classes known as tag handlers.
    2. TLD files contain the description of various ags that are part of the tag library.

* As part of this (ctld) tag library the tag that are frequency used (most essential) are provided
Ex: if, foreach, out, set, remove etc.

2. SQL : the tags that can be used to deal with data bases are provided as part of SQL tag library.
3. FMT: this tag library contains the tag that can be used in developing 118n application.
4. XML contains the tags that can be used to deal with XML.
* Every tag library is identified with a unique id (URI).
* As part of the TLD files we can find the URI that is used to identify the tag library uniquely.

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:set var="vone" value="10" scope="page"/>
<c:out value="${vone}"/>
```

Internally some java code will be executed ie **pageContext.setAttribute("vone","10");**

* When we use the above statement internally the code equalent to **pageContext.setAttribute("vone","10")** will be executed.
ie **vone** will be stored as a attribute in the pageContext & here **vone** is called as a **scopped variable** with the scope page.

```
<c:remove var="vone" scope="page"/>
<c:out value="${vone}"/>
```

* When remove tag is evaluated internally **pageContext.removeAttribute("vone");** will be executed .
* scoped variable is nothing but an attribute that is store in a specific scope.

Ex:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:set var="vone" value="10" scope="page"/>
<c:set var="vone" value="20" scope="request"/>
<c:set var="vone" value="30" scope="session"/>
<c:set var="vone" value="40" scope="application"/>

<c:out value="${vone}"/>
```

Output is : 10

* When we use a scoped variable in the **EL** expression it will first check for the variable in the **page scope**, it is available it evaluated but if it is not available that it will be check in **request scope,** it is available it is evaluated and it is not available then it will check **session scope** it is available it is evaluated and it is not available then it will check **application scope**.
* As part of **EL** expressions we can use the **implicit** variables like **param, param values, Header values , page scope, request scope, session scope, application scope**.

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:set var="vone" value="10" scope="page"/>
<c:set var="vone" value="20" scope="request"/>
<c:set var="vone" value="30" scope="session"/>
<c:set var="vone" value="40" scope="application"/>
<c:out value="${pageScope.vone}"/>
<c:out value="${requestScope.vone}"/>
<c:out value="${sessionScope.vone}"/>
<c:out value="${applicationScope.vone}"/>
```

* By using **EL** expression we can perform **arithmetic, logical & relational** operators.

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:set var="vone" value="10" />
<c:set var="vtwo" value="20" />
<c:set var="vthr" value="30" />
<c:set var="vfour" value="40" />
<c:out value="${vone+vfour}"/><br>
<c:out value="${vone < vtwo && vfour < vthr}"/><br>
<c:out value="${10 < 20 && 30 < 40}"/> <br>
```

Output is :

50
false
true

* **Empty** operator returns **true** if there is no value for variable & it returns false if there is variable with a value.

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:set var="vone" value="10" />
<c:set var="vtwo" value="20" />
<c:out value="${vone}"/><br>
<c:out value="${empty(vone)}"/><br>
<c:out value="${empty(vthr)}"/><br>
```

Output is :

10
false
true

* If there is a field with the name age in a form we can get the value of the age field by using **"${param.age}"**.
* If can be used to decide weather a specific portion of **jsp** must be evaluated (or) not.

Ex :
1

```
<c:if test="${10<20}"/>
10 is less then 20
</c:if>
```

2

```
<c:if test="${param.one<param.two}"/>
10 is less then 20
</c:if>
```

* If the test expression returns true then the body of it will be evaluated other wise the body will skip.
* We can access the properties of the java beans using **EL** expressions.

Ex :
1 A java bean object can be used as a property as another java bean object.

1. **Address.java**

```
package info.inetsolv;
public class Address
{
 public Address()
 {
  System.out.println("Address objected created.....");
 }
 String street,city,state;
 public void setStreet(String s)
 {
  this.street=s;
 }
 public void setCity(String s)
 {
  this.city=s;
 }
```

```
 public void setState(String s)
 {
  this.state=s;
 }
  public String getStreet()
  {
   return street;
  }
  public String getCity()
  {
   return city;
  }
  public String getState()
  {
   return state;
  }
}
```

2.**Student.java**

```
package info.inetsolv;
import info.inetsolv.Address;
public class Student
{
 private int age;
 private String name;
 private Address address;
 public Student()
 {
  System.out.println("Student object created.....");
 }
 public void setName(String n)
 {
  this.name=n;
 }
  public void setAge(int a)
  {
   this.age=a;
  }
   public void setAddr(Address a)
   {
    this.address=a;
   }
    public String getName()
    {
      System.out.println("Student name is "+name);
      return name;

    }
```

```
   public int getAge()
   {
    System.out.println("Student age is "+age);
    return age;
   }
   public Address getAddr()
   {
    System.out.println("Student address is "+address);
    return address;
    }

}
```

* The above two java bean objects are used in one jsp program.

1 **he.jsp**

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%
        info.inetsolv.Student s;
        info.inetsolv.Address a;
        a=new info.inetsolv.Address();
        a.setStreet("E.C.Nagar");
        a.setCity("Hyd");
        a.setState("A.P");

        s=new info.inetsolv.Student();
        s.setName("Sudarshan");
        s.setAge(25);
        s.setAddr(a);
        pageContext.setAttribute("svar",s);
%>
<c:out value="${svar}"/><br>
<c:out value="${svar.name}"/><br>
<c:out value="${svar.age}"/><br>
<c:out value="${svar.addr}"/><br>
<c:out value="${svar.addr.street}"/><br>
<c:out value="${svar.addr.city}"/><br>
<c:out value="${svar.addr.state}"/><br>
```

Output is:

```
 info.inetsolv.Student@cc5436
Sudarshan
25
info.inetsolv.Address@acbd3a
E.C.Nagar
Hyd
A.P
```

* In the above code we have stored the student object using the scoped variable name **svar**.

<c:out value="${svar.addr}"/>

* The above one property is addr but it data type is **Address** so it is internally called **toString** method in browser **Hashcode**.

So we can specify the below types ie

```
<c:out value="${svar.addr.street}"/>
<c:out value="${svar.addr.city}"/>
<c:out value="${svar.addr.state}"/>
```

* **ForTokens :** forTokens is a tag that can be used to split a string in to multiple tokens.

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:forTokens var="tok" items="123,ramesh,2000" delims=",">
<c:out value="${tok}"/><br>
</c:forTokens>
```

Output is :

123
sudarshan
9000

* in the above we have given **, (comma)** so that's why we use **delims=","** & if we use **:** then we have to use **delims=":"** and so on……

* If **, (comma)** is the delimeter instead of using **forTokens** tag we can use **forEach** tag as shown below.

Ex:

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:forEach var="tok" items="123,sudarshan,9000">
<c:out value="${tok}"/><br>
</c:forEach>
```

Output is :

123
sudarshan
9000

* We can write code which is similar to the following loop using **forEach**

```
for(int i=0;i<10;i++)
{
  System.out.println(i);
}
```

(or)

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
```

```
<c:forEach var="i" begin="0" end="10" step="1">
<c:out value="${i}"/><br>
</c:forEach>
```

Output is:

0
1
2
3
4
5
6
7
8
9
10

* We can access the elements of an array using **forEach** tag

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%
        int i[]=new int[5];
           i[0]=10;
           i[1]=20;
           i[2]=30;
           i[3]=40;
           i[4]=50;
        pageContext.setAttribute("svar",i);
%>
<c:forEach var="i" items="${svar}">
<c:out value="${i}"/><br>
</c:forEach>
```

Output is :

10
20
30
40
50

* We can access the above Address javabean object

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<%
        info.inetsolv.Address a[]=new info.inetsolv.Address[2];

        a[0]=new info.inetsolv.Address();
```

```
        a[0].setStreet("E.C.Nagar");
        a[0].setCity("Hyd");
        a[0].setState("A.p");


        a[1]=new info.inetsolv.Address();
        a[1].setStreet("H.M.T.Nagar");
        a[1].setCity("sec-bad");
        a[1].setState("A.p");
        pageContext.setAttribute("svar",a);
%>
<c:forEach var="itm" items="${svar}">
<c:out value="${itm.street}"/><br>
<c:out value="${itm.city}"/><br>
<c:out value="${itm.state}"/><br><br>
</c:forEach>
```

Output is :

E.C.Nagar
Hyd
A.p

H.M.T.Nagar
sec-bad
A.p

* The above code creates as array of type address whose size is **2** & array is pointed by the scoped variable **svar**
* We can access the elements stored in various types of collection object (vector,array list,hash table etc).

JSP/JSTL

* **Procedure to configure the web application to use tag libraries**.

Step 1: copy the jar files that are provided to the tag libraries implementer under **WEB-INF\lib** directory.
Step 2: Copy the **TLD** files under **WEB-INF** directory.
Note : Can be copied under a subdirectory **WEB-INF** also.
Step 3: To tell the container that we want to use a tag library as part of our project we must add **taglib** tag as part of **web.xml**

Web.xml

```
<web-app>
        <display-name>JSP Tags Example </display-name>
   <description>
```

```
        Shows how to use custom jsp tags in our apps.
    </description>
        <taglib>
            <taglib-uri>http://java.sun.com/jstl/core</taglib-uri>
            <taglib-location>
             /WEB-INF/tlib/c.tld
            </taglib-location>
        </taglib>
        <taglib>
            <taglib-uri>http://java.sun.com/jstl/sql</taglib-uri>
            <taglib-location>
            /WEB-INF/tlib/sql.tld
            </taglib-location>
        </taglib>
</web-app>
```

* As part of web.xml we must add the information about all the tag libraries that has to be used in the project.
* As part of jsp file to use a tag libraries we must import the tag library, using **taglib** directive.

Ex :

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<c:out value="Hi Suarshan"/><br>
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
<xxx:sometag/>
<c:out value="Hi Sudha"/>
```

* When the above page is compiled by the jsp compiler , the jsp compiler takes the information provided in the tag lib directive and internally as shown below

URI	Prefix
http://java.sun.com/jstl/core	**c**
http://java.sun.com/jstl/sql	**sql**

* When The jsp compiler sees**<xxx:sometag/>**, the compiler takes the **prefix(xxx)** and checks for the prefix in the table. In this case there is no prefix(xxx) in the table. So the compiler will not treated it as jsp **custom tag** ie, this line will be treated as **template text**.
* When the **jsp compiler** sees **<c:out value="Hi Sudarshan"/>** it picks up the **prefix(c)** and checks for the prefix in the table. In this case it is available the compiler now picks up the uri corresponding to the prefix from the table (**http://java.sun.com/jstl/core)**. this URI will be used by the jsp compiler to get the TLD files information from **web.xml** .
* The jsp compiler reads the information provided in TLD file to generate java code.
* There are two different types of JSTL tag libraries ie
    - **1 → c.tld , sql.tld , x.tld ,fmd.tld** supports EL expressions.
    - **2 → c-rt.tld , fmt-rt.tld , sql-rt.tld , x-rt.tld** supports the jsp.

* As part of **sql** tag library **setDataSource** tag which takes care of creating the data base connection.

```
<sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
    url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
<sql:update dataSource="${ds}">
        insert into txtest value('111','222')
</sql:update>
<sql:update dataSource="${ds}">
        insert into txtest value(?,?)
        <sql:param value="${param.cone}"/>
        <sql:param.ctwo}"/>
        </sql:update>
<sql:transaction dataSource="${ds}">
        <sql:update>
                update table one.....
        </sql:update>
        <sql:update>
                update table two.....
        </sql:update>
</sql:transaction>
```

Ex: to get the data in data base using jsp/jstl

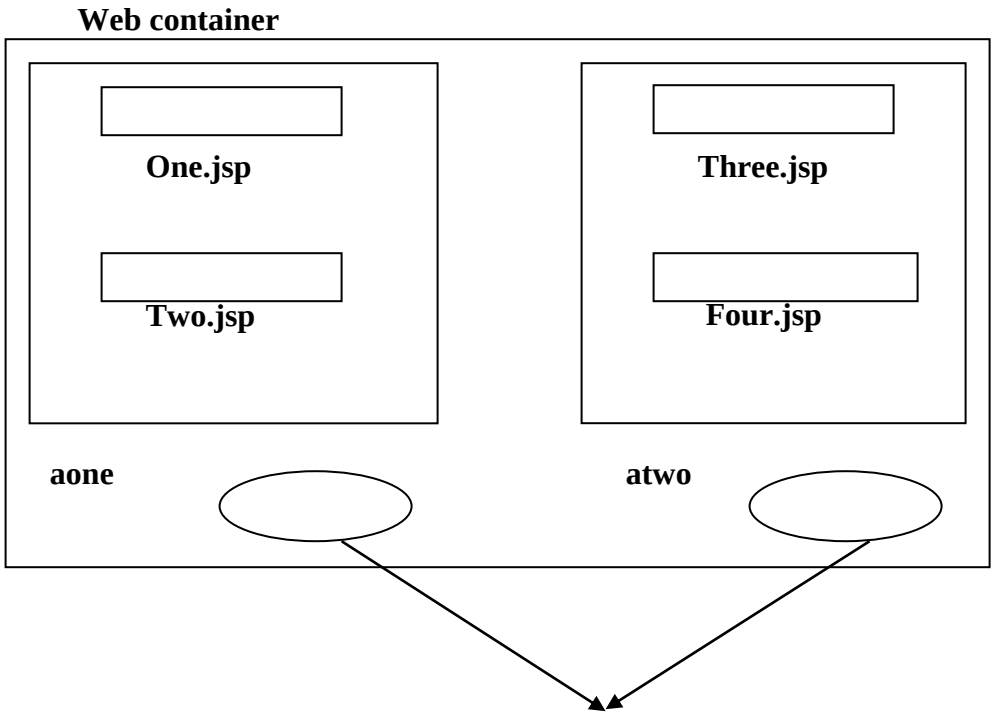
```
<%@ taglib uri="http://java.sun.com/jstl/sql" prefix="sql" %>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<sql:setDataSource var="ds" driver="oracle.jdbc.driver.OracleDriver"
                url="jdbc:oracle:thin:@localhost:1521:orcl" user="scott" password="tiger" />
<sql:query var="res_set" dataSource="${ds}"  sql="select * FROM dept" />
<c:forEach var="row" items="${res_set.rows}">
<c:out value="${row.deptno}"/> <br>
<c:out value="${row.dname}"/> <br>
<c:out value="${row.loc}"/> <br>
</c:forEach>
```

Output is:

```
10
ACCOUNTING
NEWYORK
20
RESEARCH
DALLAS
30
SALES
CHICAGO
40
OPERATIONS
BOSTON
```

* **${res_set_rows}** is a attribute which is responsible to get the data in data base.
* If a java bean has to be used by one jsp while processing a requesat we can use **scope=page**.

* If multiple jsp's involve in processing a request has to else the same object we must specify **scope=request**.
* If anything specific to a client has to be used while processing multiple request then we must store the java bean in session object.
* If a java bean holds something that is not specific to a client & in that has to be used while processing multiple request we must use **scope=application**.



For every application there will be a servletContext.

* As shown in the above diagram for every web application there will be one servlet context object.

Ex:

```
<%
        ServletContext app=config.getServletContext();
        System.out.println(app);
%>
```

Output is:

org.apache.catalina.core.ApplicationContextFacade@29ea31

* ServletContext.getContext() can be used to get the reference of a ServletContext object of another web application ie we can write

Ex:

```
<%
        ServletContext anotherApp=application.getContext("/atwo");
        RequestDispatcher rd=anotherApp.getRequestDispatcher("/athr");
        rd.forward(request,response);
%>
```

* The above code works fine in the default setup on weblogic but on tomcat ServletContext.getContext() returns **null** if we have not to addes the following information **server.xml** file.

* What is static include & Dynamic include ?

Ans: Using **include Directive** is called as static include & using **JSP include action tag** is called as Dynamic include.

STRUTS

* There are so many frame works available for the developers to develop the projects a frame work is a piece of software designed according to a good architecture, the frame works are designed by the people who has experienced in developing the projects based their previous experienced they identify the task that are carried out as part of various projects & they implement the code to carry out these tasks.
* A frame work provides a procedure for implementing the project.
* We can reduce the total amount of time required to develop a project using a frame work.

* Struts is develop using **servlet/jsp technology,** it is not designed keeping a particular web container in mind if we develop a project based on struts we can deploy that projects on any of web container.
* We download struts frame work in **Jakarta.apache.org** we get a set of **JAR** files & set of **TLD** files.
* As part of **struts.jar** several classes are provided some of these classes are related to tag libraries of struts as part of struts a set of classes like **ActionErrors, ActionError, Action , ActionForm** etc are provided these classes are used in developing our own code. Apart form this a servlet class (**org.apache.struts.action.ActionServlet)** this sertvlet is called as **controller**.
* **struts-html , struts-bean , struts-logic , struts-tiles , struts-template , struts-nested** are the tag libraries that are provided as part of struts frame work.

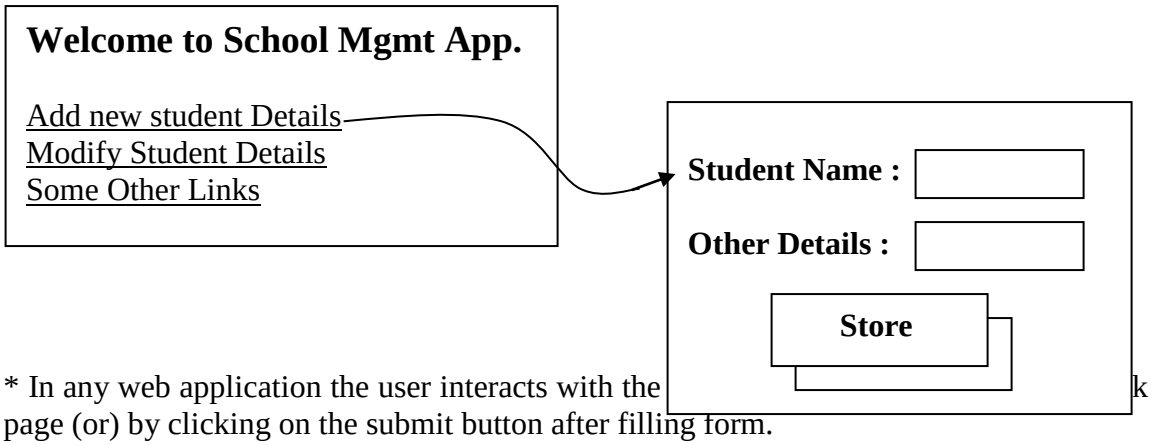
→ In i-net solve cd →cd1→SIREXAMPLES→j2eec→struts→struex→struts-blank.war

* We can use struts-blank.war as a starting point for the development of a struts based web application.
* To start with a struts based project

Step 1: create a directry
Ex:**f:\saone**
Step 2: copy struts-blank.war to the above directory
Step 3: move to the above directory
Step 4: run the command **jar xvf struts-blank.war**

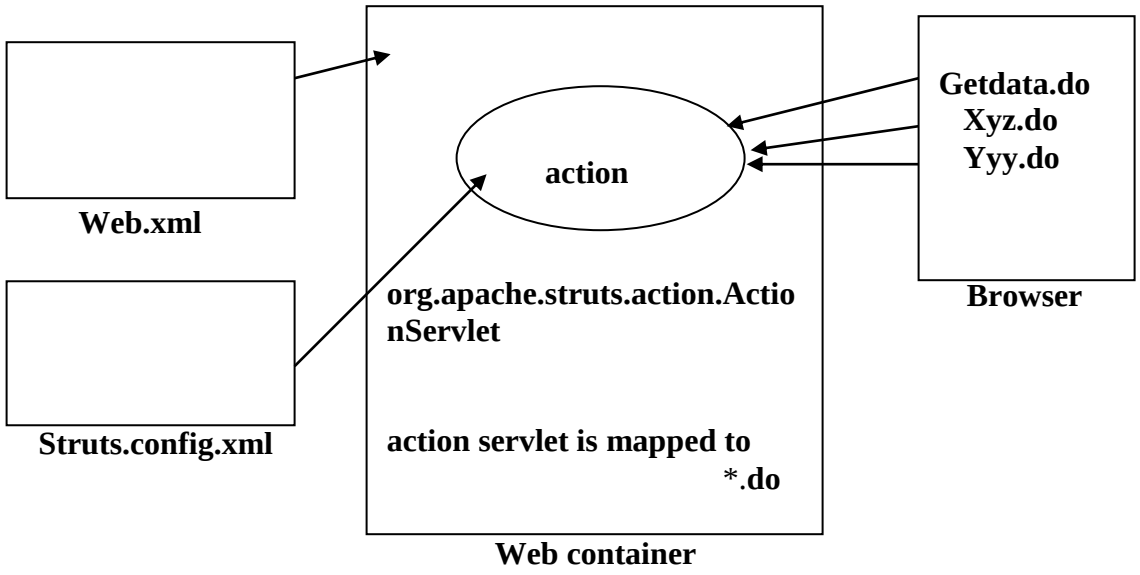
* The above command extract the content of the war file & places the content under **saone**.
* When we start running a struts based web application the web container will first create a servlet object based on **org.apache.struts.action.ActionServlet**.
* In the default setup action servlet is mapped to ***.do**
* When we send the request using the URL's like **one.do , two.do , xya.do** etc the service method of ActionServlet method will be executed.

Ex :



* In any web application the user interacts with the k a web page (or) by clicking on the submit button after filling form.
* When the user clicks on a link (or) clicks on the submit button the web application as to carry out some work according to the application requirement, in **struts** frame work this work is called as **action**.
* As part of **web.xml** of a struts based application the information about the ActionServlet must be added as shown below (we need not add this information to web.xml). if we are using struts-blank application add the below code.

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
<init-param>
        <param-name >conifg</param-name>
        <param- value>/WEB-INF/struts-config.xml</param-value>
</init-param>
        <load-on-startup>2</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
</servlet-mapping>
```



* When we start a web application using this above configuration the web container reads **web.xml** as we have specified **load-on-startup** for ActionServlet the web container will create a servlet object based on **org.apache.struts.action.ActionServlet**.

The **init()** of ActionServlet gets the value of the init-param config (in our case the value is) **/WEB-INF/struts-config.xml** The code in the **init()** reads the information available in **struts-config.xml** file.

* According to out configuration the ActionServlet is mapped to ***.do**
* As shown in the above diagram when we use any **url** that ends with **.do** the web container will execute the **service()** of ActionServlet.

Ex :

* create a folder in that copy **struts.jar & servlet-api.jar** files & setting the path.
* if servlet-api.jar is not there so we can use **weblogic.jar** file also.

C:\> set CLASSPATH=struts.jar;servlet-api.jar;

Or

C:\> set CLASSPATH=struts.jar;weblogic.jar;

Struts.jar available in **saone** folder →WEB-INF→lib.
Servlet-api.jar available in → C:\Program Files\Apache Group\Tomcat 4.1\common\lib.
Weblogic.jar available in → **C:\bea\weblogic700\server\lib**

* As part struts based projects we split of the entire application into various actions & we implement the code to carry out an action as part of Action classes.
* Action class we implement as part of our struts project must be a subclass of **org.apache.struts.action.Action**.

Ex: To run the ActionOne Servlet In struts
* first create one folder name any thing sample is struts in that copy in
struts.jar,servlet-api.jar (or) struts.jar,weblogic.jar
and set the path.after that create below programs compiled it
1 To create a application and save it .java **ActionOne.java**
2 To create a application and save it .java **ActionTwo.java**
* these **.class** files copy in **tomcat** or **weblogic** requirement folders.
* finally changes in **struts-config.xml** ie write the resouses names

Ie
```
<action-mappings>
<action path="/xxx" type="ActionOne">
</action>
<action path="/yyy" type="ActionTwo">
</action>
</action-mappings>
```

1. ActionOne.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class ActionOne extends Action
{
 public ActionOne()
 {
  System.out.println("ActionOne Objected created.....");
 }
 public ActionForward execute(ActionMapping mapping,ActionForm

form,HttpServletRequest request,HttpServletResponse

response)throws Exception {
  System.out.println("returning ActionOne.....");
  return null;
 }
}
```

2.ActionTwo.java
Same program as above.
* We must provide the information about the struts actions in the **struts-config.xml** as part of <action-mappings> as shown below.

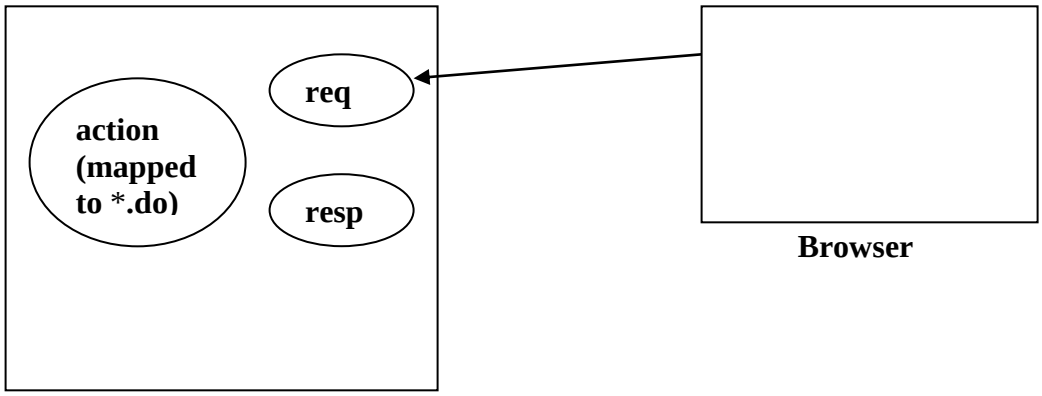
```
<action-mappings>
<action  path=”/xxx” type=”ActionOne”/>
<action  path=”/yyy” type=”ActionTwo”/>
</action-mappings>
```

Information about the actions in a project there will be several actions

* In the path we should not use **.do** the above information will be read by the action servlet when we startup the web application.

* When we type the following URL in the browser **http://localhost:8080/saone/aaa.do**
Setp 1: The browser sends the request to the web container.
Step 2: The web container creates the request object & the response object.
Step 3: The web container calls the service() of the ActionServlet by passing the request & response objects (As the URL ends with **.do** the web container calls the service() of ActionServlet).
* The ActionServlet checks for the path **/aaa** in the action mappings. In this example there is no action mapped to **/aaa** so the action servlet sends a error message to the browser.

* If we type the following URL **http://localhost:8080/saone/xxx.do**



Web container

Step 4: The ActionServlet checks for the information about the action mapped to the path **/xxx**.
Step 5: In this example the information about **/xxx** is available from this information it takes the name of the Action class (**ActionOne**).
Step 6: The struts frame work calls the execute() on action object if the object is not available the struts frame work creates the object.
Note : As the code in the **execute()** returns **null** the frame work will not do anything after this step.
* The information that is provided as part of **<action path="/xxx" type="ActionOne"/>** can be accessed using the object ActionMapping that is passed as a parameter to the **execute()**.
* As part of ActionMapping we can specify one (or) more than one forwards as shown below.

```
<action path="/xxx" type="ActionOne">
        <forward name="fone" path="/aaa.jsp"/>
        <forward name="ftwo" path="/bbb.jsp"/>
</action>
```

* In the ActionMapping we have used two forwards the names of these forwards are **fone & ftwo**.
* As part of our Action code we can use **findForward()** to get the information about the forwards.

1.ActionOne.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class ActionOne extends Action
{
 public ActionOne()
 {
  System.out.println("ActionOne Objected Created.....");
 }
public ActionForward execute
                        (ActionMapping mapping,ActionForm form,HttpServletRequest
request,HttpServletResponse  response) throws Exception
 {
  System.out.println("returning fone.....");
 return mapping.findForward("fone");
 }
}
```

2.ActionTwo.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class ActionTwo extends Action
{
 public ActionTwo()
 {
  System.out.println("ActionTwo Objected created.....");
 }
 public ActionForward execute
                              (ActionMapping mapping,ActionForm form,
 HttpServletRequest request,HttpServletResponse  response)throws Exception
 {
  System.out.println("returning ftwo.....");
  return mapping.findForward("ftwo");
 }
}
```

3 aaa.jsp

Hi sudarshan

4. bbb.jsp

Hi sudha

* When we type **http://localhost:8080/saone/xxx.do**
* In that the above steps **step 1 to step 6** are same……

Step 7: In this case the execute() is retrieving the forward **fone** (the path of **fone**) is defined as **aaa.jsp**.

Step 8: This forward will be taken by the frame work & the frame work will forward the request to **bbb.jsp**

* In this case the request is processed by two servlets

1.ActionServlet
2.one.jsp servlet

* As part of struts based projects there will be several actions in **struts-config.xml** as part of every action we can defined the forwards as shown below these forwards are called as **local forwards**.

```
<action-mappings>
        <action path="/xxx" type="ActionOne">
                <forward name="fone" path="/one.jsp"/>
         </action>
        <action path="/yyy" type="ActionTwo">
                <forward name="ftwo" path="/two.jsp"/>
        </action>
 </action-mappings>
```

* If multiple actions uses the same forward then instead of defining the forwards multiple times we can use the **global forwards**.

```
 <global-forwards>
        <forward name="fone" path="/one.jsp"/>
 </global-forwards>
 <action-mappings>
        <action path="/xxx" type="ActionOne">
         </action>
        <action path="/yyy" type="ActionTwo">
        </action>
 </action-mappings>
```

Ex : Using STRUTS a simple project to get the values in data base.

1. Creating a package **info.inetsolv**

Dept.java

```
package info.inetsolv;
public class Dept
{
 private String DeptNo;
 private String Dname;
 private String Loc;
 public Dept()
 {
  System.out.println("Dept object created.....");
```

```
 }

  public Dept(String deptno,String dname,String loc)
  {
    this.DeptNo=deptno;
    this.Dname=dname;
    this.Loc=loc;
  }
   public String getDeptNo()
   {
          return DeptNo;
   }
    public String getDname()
    {
     return Dname;
    }
    public String getLoc()
    {
     return Loc;
    }
}
```

* The above java bean supports three properties
1 DeptNo
2 Dname
3 Loc.

2.**GetDataAction.java**

```
import info.inetsolv.Dept;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class GetDataAction extends Action
{
 public ActionForward execute
(ActionMapping mapping,ActionForm form,
                                HttpServletRequest request,HttpServletResponse  response)
{
 try
 {
   //business logic is to get the data in this case.
   Class.forName("oracle.jdbc.driver.OracleDriver");
   Connection con=
   DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
   Statement stmt=con.createStatement();
   String sqlstmt= "select * from dept where deptno=10";
```

```
   ResultSet rs=stmt.executeQuery(sqlstmt);
   //go to first row..
   rs.next();
 info.inetsolv.Dept d=
new info.inetsolv.Dept(rs.getString("deptno"), rs.getString("dname"),rs.getString("loc"));
    request.setAttribute("deptdata",d);
   System.out.println(" return data sucess.....");
   return mapping.findForward("success");
 }
  catch(Exception e)
  {
   System.out.println("return Failure.....");
   return mapping.findForward("failure");
  }
 }
}
```

Note : If the try block is executed successfully **success** will be excuted as a forward if id fails to execute the try block then catch box will be executed that means **failure** as a forward.

3.**dd.jsp**

```
<%
        info.inetsolv.Dept x=(info.inetsolv.Dept)request.getAttribute("deptdata");
        out.println(x.getDeptNo());
        out.println(x.getDname());
        out.println(x.getLoc());
%>
```

Finally we change the content **struts-config.xml**

```
<action-mappings>
        <action path="/gd" type="GetDataAction">
        <forward name="success" path="/dd.jsp"/>
        <forward name="failure" path="/df.jsp"/>
        </action>
 </action-mappings>
```

The output is :

10 ACCOUNTING NEW YORK

* In struts we provide the presentation logic in jsp.
* In the above example we provide the business logic as part of action class but in most of the struts based projects the business logic will be implement as part of **model classes**

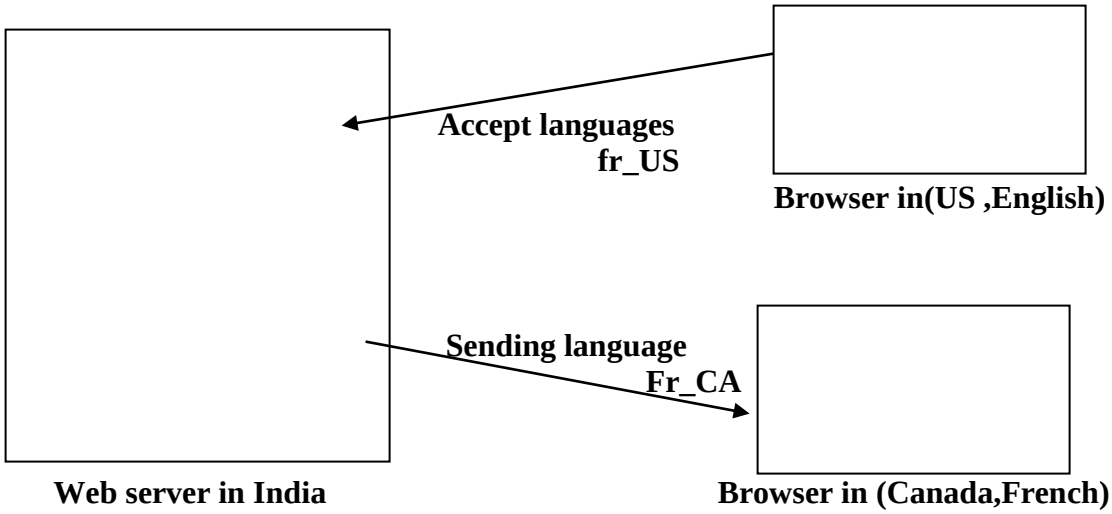
Note : In struts based projects we carefully separate presentation logic from business logic, This simplifies the maintenance of the project.

* The process of retrieving data from db & storing the object is called loading.

* The execute() of Action object will be called from the code of ActionServlet, The execute() in this example gets the data & loads the data in dept object that dept object will be stored in request object with the attribute name **deptdata** the reason for storing it is this data has to be a accessed by the **dd.jsp**

* **What is internationalization(i18n) how is supported in STRUTS?**

* When we install an operating system (or) at later point of time we can specify the locale settings (Regional Settings) these settings provide the information like the country code like the language & the currency symbol that has to be used the format in which date has to be displayed the time zone etc.
* In system go to control panel → Regional settings.
* When we develop an application targeting the customers living in various parts of the world we must take care of the issues like the date format , the language in which the information is presented if we develop an application to take care of these issues we can call our application as **i18n** application.
* STRUTS provide as easy way to build **i18n** applications.



* A browser as part of every request sends a header with the name Accept language in this header it will provide the information about the language code & the country code of the machine on which the browser is running.
* As part of our servlets we can use **request.getHeader("Accept_Language")** to get the locale setting of the browser.
* To support various locale we can develop multiple jsp's for multiple regions as shown below.
* If we follow this approach to support 10 different locale we must provide 10 jsp file that is if we need 100 jsp's in a project for supporting one locale for 10 locales we must maintain 100*10 jsp's.
* Instead of developing the application like this we can use FMT tag library of JSTL (or) the tag library's that are provided as part of JSTL.

* We can create the property files with a set of **resources**

application_en_us.properties

base name **language code** **Country name**

Note : In this file we must place the resources in **en_us**.

page.title = welcome

value of the resource

key of the resource

page.body = hi how r u
some.res = value of the resource

* In the above file there are multiple resources a set of resources combined together is called as resource **bundled**, In the above file we have provided the values of resources in English language, if we have to support **French** language we must provide a file with the name **app_fr_CA.propertys** in this file as the values we must use the strings in French language but the keys must be same.
* We can access the values of the message resources by using message tag ie available as part of struts bean tag library.

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<html>
        <head>
                <title>
                        <bean:message key="page.title"/>
                </title>
        </head>
        <body>
                <bean:message key="page.body"/>
        </body>
</html>
```

* When Bean messahe tag is evaluated it checks the locale of the browser is **fr_CA** the values will be picked up from **xxx_fr_CA.properties**
* AS we can't support every locale we must provide a default properties file if (a default properties file is a file with the name basename.properties)

* If we use struts-blank.war application to develop our web application we can add all the resources in application.properties which is configured by providing the message resource tag as shown below.

<message-resource parameter="resources.application"/>

* We must copy the properties file in **classes → resources**

<message-resource parameter="resources.application"/>

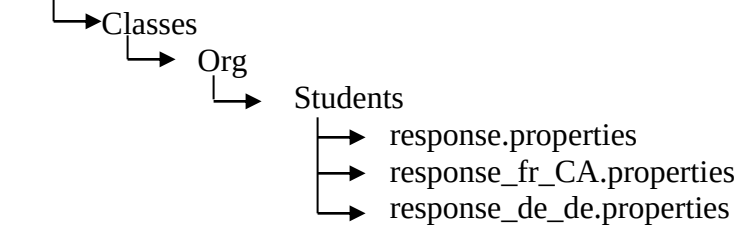
Name of the subdirectry under which we must Copy the properties file

Base name of the properties file

* resources directry must be created under the WEB-INF/classes directry.
* If we place the following tag in **struts-config.xml**

<message-resources parameter="org.student.responce" />

* We must create the following directry structure to place the resource files.



Note : We can add **N** no.of **resone_xx_yy.properties** to support **various locales**.

* During the development we can configure the message resources as shown below.

<message-resources parameter="org.student.resone" null="false"/>

Default value of **null** is **true**, if **null** is set to **false** even if the resource with a particular **key is not available Exception** will not be thrown.
Ex: If we use a resource key **zzz** and if it is not available we will get the value like

???_xx_yy_zzz.???

Local properties

* As part of a project we will use several resources instead of placing all the resources in a single properties file we can use multiple properties files.

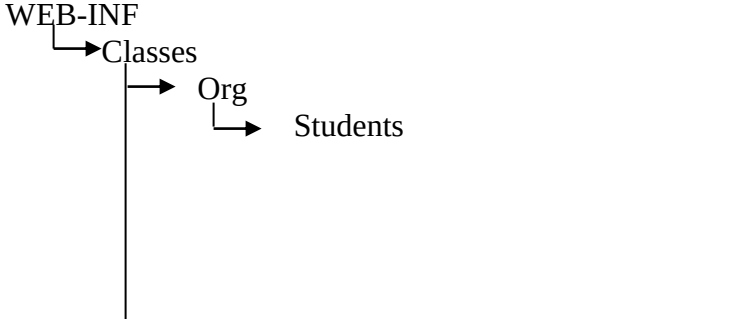
Procedure for using multiple resource bundles

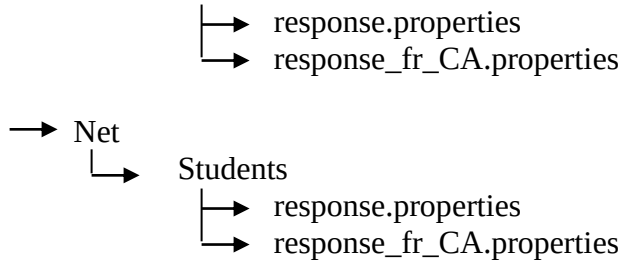
Step 1: Add message resources tag as shown below in the **struts-config.xml** file.

```
<message-resources parameter="net.student.restwo" null="false"
                                                    key="SOMEKEY"/>

<message-resources parameter="org.student.resone" null="false"/>
```

Note : If key is not specified the default key (bundle name) **org.apache.struts.action.MESSAGE** will be used using one key we can load only one bundle.

Step 2: Create a directry structure as shown below & add the propertie file.





Step 3: To access the resources from **resone** we can use message tag as shown below

<bean:message key="res.one.one"/>

Step 4: To access the resources available in **restwo** we must specified the bundle attribute as shown below.

One.jsp

```
<@taglib uri="/tags/struts-logic" prefix="logic"/>
<bean:message key="res.one.one"/>
<bean:message key="res.one.one"  bundle="org.apache.struts.action.MESSAGE"/>
```

* When we start struts based application the **init()** of the **ActionServlet** reads StrutsConfig file in this config file it will read the message resources tag & loading (reading) the resources of a bundle & places the bundle as an attribute in ServletContext object.

Ex : **test.jsp**

```
<@taglib uri="/tags/struts-logic" prefix="logic"/>
<% request.setAttribute("souji","sudhi");
%>
<logic:present name="souji" scope="request">
        souji is available in the request
</logic:present>
<logic:present name="aaa" scope="request">
aaa is available
</logic:present>
```

* The body of a one will be evaluated if the attribute is available in a particular scope in the above exaple the body of the first tag will be evaluated.

If the body is not present it will be evaluated if the attribute is not there in a particular scope.

How to Handle Exceptions

* To Handle the Exception we can provide try catch blocks.
* In the we have seen previous example like

Ex:

```
public class GetDataAction extends Action
{
 public ActionForward execute(ActionMapping mapping,ActionForward forward,
                              HttpServletRequest request,HttpServletResponse response)
{
```

```
 try {
  // code to perform the execution……..
return mapping.findForward("success");
}
catch(Exception e)
{
 System.out.println("failure.....");
 return mapping.findForward("failure");
}}
```

* We can handle the Exceptions in struts by declaring how the exception to be handle in **struts-config.xml** this is called as declarative Exception Handling.

Ex: **MyAction.java**

```
import java.io.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class MyAction extends Action
{
 public ctionForward execute(ActionMapping mapping,ActionForward forward,
        HttpServletRequest request,HttpServletResponse response)throws Exception
 {
  System.out.println("LIne one");
  FileInputStream fis=new FileInputStream("xxx.aaa");
  System.out.println("line two");
  return mapping.findForward("success");
 }}
```

* In the above code we have not provided try catch block when the execution of the code in the **execute()** fails it throws an Exception.
* When the Exception is thrown the struts frame work checks for the information on how to handle the Exception in the configuration file.

If we provide the following information the frame work will forward the request to **eh.jsp**

* in **struts-config.xml**

```
<global-exceptions>
<exception type="java.lang.Exception" key="error.msg.one"   path="/eh.jsp"/>
</global-exceptions>


<action-mappins>
                <action path="/sudhi" type="MyAction">
                <forward name="success" path="/eh.jsp"/>
                </action>
</action-mappins>
```

* error.msg.one is the key to the error message that has to be send to the client so we must defined it in the properties file as shown below.

-- standard errors -- we have to add

Applications.properties file

error.msg.one=failed due to……..

* As part of jsp file **eh.jsp** we must provide the errors tag to generate the error message.

eh.jsp

```
<%@taglib uri="/tags/struts-html" prefix="html"%>
<html:errors/>
```

The out put is :

- faild due to.....

* The above one is thoe one way to handle the exceptions & another type of exception is handle is shown below.

* similar to global forwards & local forwards in struts we can use local Exception & global Exception.

* As part of struts we can represent an error using an object of type

Org.apache.struts.action.ActionErrors

If we have to represent 10 errors we must create 10 objects of type ActionError & there object must be added to an object of type it is showing below example.

Ex: MyAction1.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class MyAction1 extends Action
{
 public ActionForward execute(ActionMapping mapping,ActionForm form,
        HttpServletRequest request,HttpServletResponse  response) throws Exception
{
 System.out.println("line one executed..");
 ActionErrors aes=new ActionErrors();
 System.out.println("1----"+aes.size());
 ActionError ae1,ae2,ae3,ae4;
  ae1=new ActionError("error.one");
  ae2=new ActionError("error.two");
```

```
  ae3=new ActionError("error.thr");
  ae4=new ActionError("error.four");
  aes.add("propone",ae1);
  aes.add("propone",ae2);
  aes.add("proptwo",ae3);
  aes.add("proptwo",ae4);
  System.out.println("2----"+aes.size());
  saveErrors(request,aes);
  return mapping.findForward("success");
 }
}
```

2. **struts-config.xml** add in

```
<action-mappings>
        <action path="/soudhi" type="MyAction1">
        <forward name="success" path="/fone.jsp"/>
        </action>
 </action-mappings>
```

3. **application.properties** add in # -- standard errors --

```
error.one=message one
error.two=message two
error.thr=message thr
error.four=message four
```

4. **fone.jsp**

```
<%@taglib uri="/tags/struts-html" prefix="html"%>

<html:errors property="propone"/>
<html:errors property="proptwo"/>
```

(or)

```
<%@taglib uri="/tags/struts-html" prefix="html"%>

<html:errors/>
```

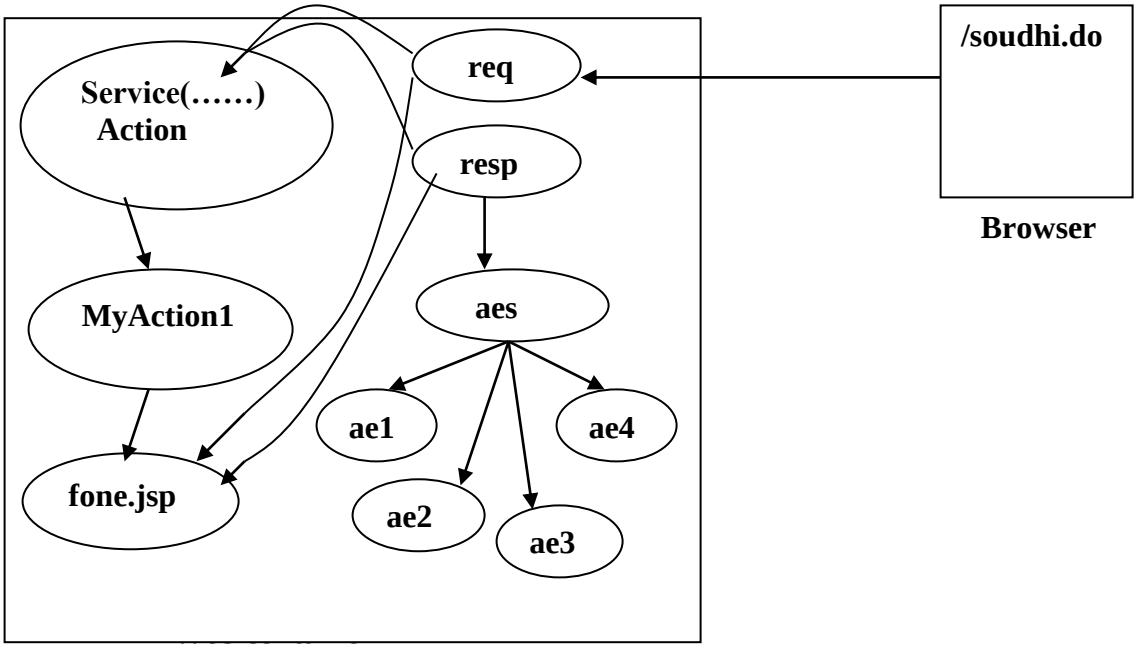
→ It generate error messages, it picks up ActionErrors objects & in the above application it picks up 4 errors.

Out put is :

- message one
- message two
- message three
- message four

* When we send the request using **soudhi.do** the web container creates the request object the response object & starts the execution of the **service()** of ActionServlet by passing the request & response object.

* If require an object based on MyAction1.class will be created & then the execute() will be invoked.
* The code in the execute() creates four ActionError objects & store this objects in ActionErrors object.



* Then **saveErrors** is executed the information about ActionErrors object will be stored as an Attribute in **request** object.

* In our code we are returning the **forward("fone")** in thiscase the struts frame work forward the request to **fone.jsp** by giving the same request object & response object to **fone.jsp**

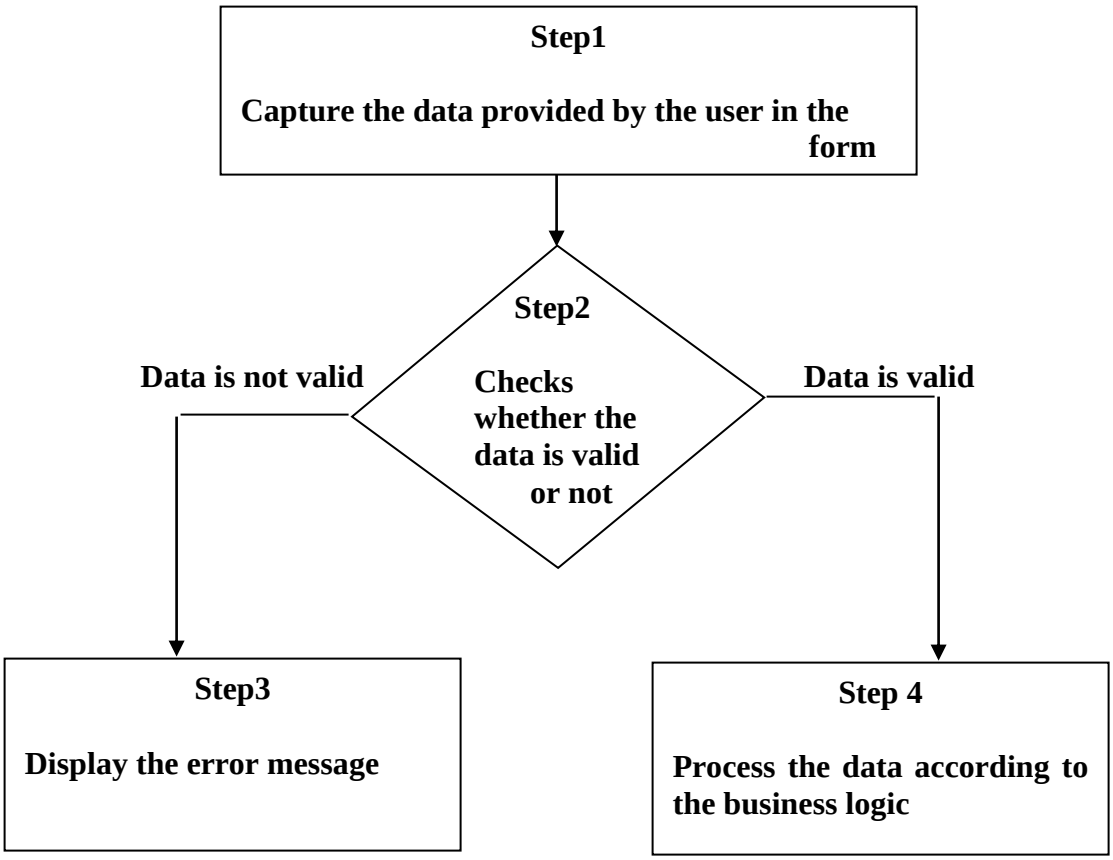
* When we use **<html:errors property="propone"/>** the tag generates the errore that are stored using propone as a property name , if we directly use **<html:errors />** it will generate all the error messages.

* While adding the error messages to ActionError we can group the error messages using property name.

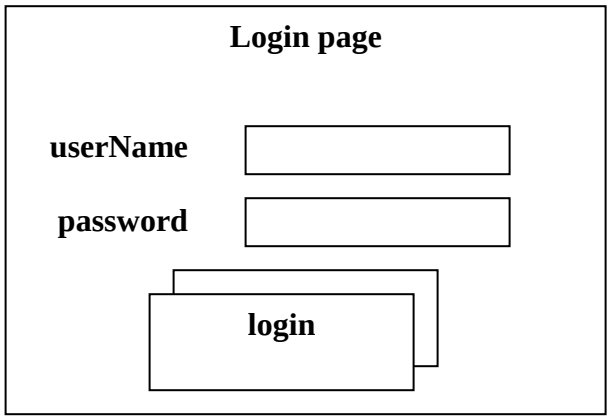
* As part of jsp1.1
ActionMessages (similar to ActionErrors)
ActionMessage (similar to ActionError)
saveMessages (similar to saveErrors)
to access the ActionMessages we must use messages tag which is slightly different from html errors tag.

* To handle the forms in struts we needed to follow the procedure given below.

Struts frame work deals with the forms following a standard approach shown in the following flowchart.



* In struts we create the login page so we have to follow some steps



Step1 : Deside about the name of the fields & give a name for the form & also deside about the validation rules that have to be applied on every field in the form.

Validation rules:

1 userName is required
2 password required

Note : username & password must be null blank string.

Step2 : Develop a **form bean** class (the data capture by the struts frame work will be stored in the form bean object).

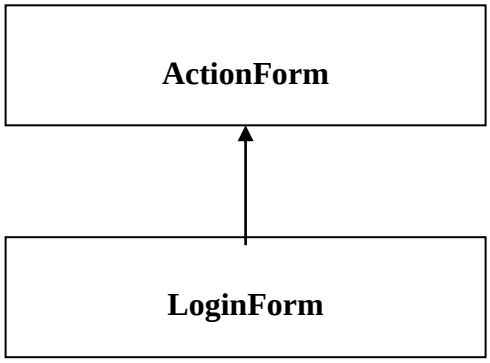
LoginFB.java

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class LoginFB extends ActionForm
{
 private String userName;
 private String password;
 public LoginFB()
 {
  System.out.println("LoginFB objected created.....");
 }
  public void setUserName(String un)
  {
   userName=un;
  }
  public String getUserName()
  {
   return userName;
  }
  public void setPassword(String pwd)
  {
   password=pwd;
  }
  public String getPassword()
  {
   return password;
  }
   public void reset(ActionMapping am,HttpServletRequest request)
   {
    System.out.println("-------reset---called");
    userName="sudhi";
    password=null;
   }
    public ActionErrors validate(ActionMapping

actionMapping,HttpServletRequest request)    {
     System.out.println("---------validate called----");
```

```
    ActionErrors aes=new ActionErrors();
    if(userName==null || userName.equals(" "));     {
     ActionError e1=new ActionError("un.required");
     aes.add("userName",e1);
    }
     if(password==null || password.equals(" "))       {
       ActionError e2=new ActionError("pwd.required");
      }
       System.out.println("-------size---"+aes.size());
          return aes;
     }
}
```

* It is responsible of the struts frame work to capture the data & store that data in the form beanobject by calling setXxx() methods.

Step3 : properties an Action class to perform the appropriate action according to the business requirement.



ActionForm form=new LoginForm();

2. LoginAction.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class LoginAction extends Action
{
 public ActionForward execute(ActionMapping mapping,ActionForm

form,HttpServletRequest request,HttpServletResponse  response)

throws Exception
 {
  LoginFB fb=(LoginFB)form;
  System.out.println("getting userName"+fb.getUserName());
  System.out.println("getting password"+fb.getPassword());
  return mapping.findForward("success");
```

```
 }
}
```

Step5 : Provide a jsp file that generates the form using the tags that are provided as part of struts html tag library.

3.**lform.jsp**

```
<%@ taglib uri="/tag/struts-html" prefix="html"%>
<html:html>
<head>
        <title> Login-form</title>
</head>
<body>
        <html:form action="/log.do">
                <html:errors/>
                <b>UserName:</b>
                <html:text property="userName"/>
                <b>Passowrd:</b>
                <html:password property="password"/>
                <html:submit property="login"/>
        </html:form>
</body>
</html:html>
```

* **The following information is below**
    1. Name of the form : **LoginForm**
    2. Name of the FB class : **LoginFB**
    3. Name of Action class : LoginAction
    4. Name of forwards used in Action : **success → start.jsp**
    5. Name of the jsp file that generates the form : **form.jsp**
    6. Path used in the action of html : form tag in the jsp file : **/log.do**

* Using the above information configure struts-config.xml

Struts-config.xml

```
<form-beans>
        <form-bean name="LoginForm" type="LoginFB"/>
 </form-beans>


<action-mappings>
 <action name="LoginForm" input="/lform.jsp"  path="/log"
                                type="LoginAction" scope="session" validate="true">
      <forward name="success" path="/start.jsp"/>
        </action>
```

```
</action-mappings>
```

Application.properties

```
un.required=user name required.....
pwd.required=password required.....
```

4 test.jsp

```
<%@ taglib uri="/tags/struts-html" prefix="html"%>
        <html:html locale="true">
        <html:form action="/log.do">
                <b>UserName:</b>
                <html:text property="userName"/><br>
                <b>Passowrd:</b>
                <html:password property="password"/><br>
                <html:submit property="login"/>
        </html:form>
        </html:html>
```

* When **<html:html>** is evaluated it generates html tag and sends it to the browser.
* When **<html:form action="/log.do>** is evaluated its check for the path **/log** provided in action tag in the **struts-config.xml** , if there is no action tag with in the path **/log** it throws an Exception indicating it can't retriving mapping for **/log** ,so that if it is available it will pickup the name of the form(**LoginForm**) then it will search for the form in form-bean tag from this tag it will pickup the name of the form-bean class (**LoginFB**) .

If required the form-bean object will be created , **reset()** method will be called and the object will be stored either in request scope or in session scope.

* When **<html:text property="username"/>** is evaluated it call **getUserName()** &then it will generate input tag as shown below.

<input type="text" name="username" value="xxx">

* When **<html:password property="password" />** it evaluated it call **getPassword()** method and then it will generate input tag as shown below.

<input type="password" name="password" value=""/>

Struts frame work be called the user bean object will be created.

* When the user press the submit button the browser sends a request using **log.do** as a URL the web container will start executing the **service()** method of **ActionServlet**.
* The frame work captures the data and calls the setter methods on the form bean object to store the data in the form bean object (it require the frame will create).
* If we specify **validate=true** in the action-mapping the jframe work will call the **validate()** method.
* If the validate() method returns **ActionErrors** object with one or more than one ActionError objects in it the frame work will forward the request to the resource specified the input attribute of action mapping (input="lform.jsp").

* If no errors are written by the validate() method the frame work will call the execute() method of Action object.

* If we use html errors tag as shown below it will generate all the errors if we need to display errors relaed to a field text to the field then we can use the property attribute as part of errors tag

<html:errors property="username"/>

* <**html:errors/>** tag internally uses the following resources

errors.header=<UL>
errors.prefix=<LI>
errors.sufix=</LI>
errors.footer=</UL>

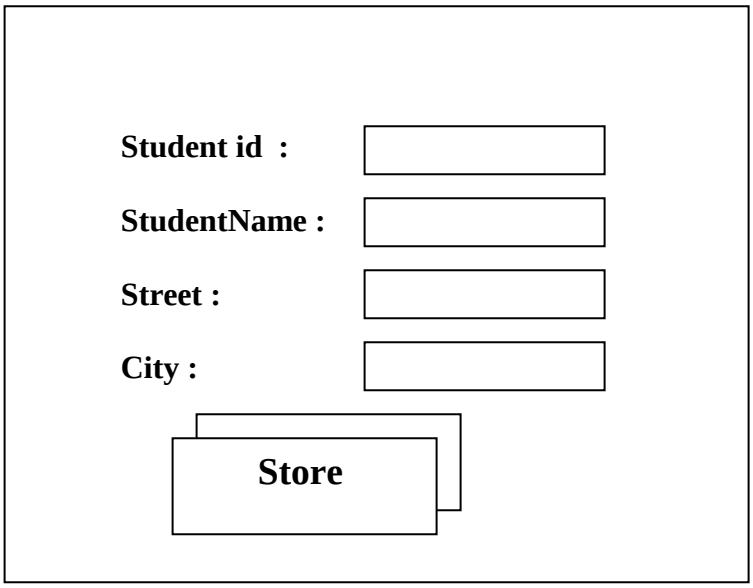
The values of these resources will be used in generating the list of errors.

* In this approach we are forced to add html tags as part of the resource files this can be eliminated by making use messages tag in place of errors tag **lform.jsp**

```
<%@taglib uri="struts-bean" prefix-"bean" %>
 <head>
<title> loginForm</title>
</head>
        <body>
        <html:message id="em">
        <bean:write name="em"/><br>
        <html:messages>
       </body>
```

* Body of messages will be evaluated for n no-of attributes where n is the no-of messages every time one message will be placed in **em**.

* The advantage of using messages tag is we will have more control over what has to be generated and we need not add html tags as part of resource file.



Form Name : New StudForm

Validation Rules :

1. stud id is required
2. stud name is required
3. min size of stud name must be 5 characters.

1. **NewStudFB.java**

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class NewStudFB extends ActionForm
{
 private int studentId;
 private String studentName;
 private String street;
 private String city;
 public NewStudFB()
 {
  System.out.println("NewStudFB object created.....");
 }
 public void reset(ActionMapping actionMapping,HttpServletRequest

request)
 {
  System.out.println("reset method called.....");
  studentId=0;
  studentName=null;
  street="E.C.Nagar";
  city="Hyderabad";
 }
  public void setStudentId(int id)
  {
   studentId=id;
  }
  public void setStudentName(String name)
  {
   studentName=name;
  }
  public void setStreet(String str)
  {
   street=str;
  }
  public void setCity(String cty)
  {
   city=cty;
  }
  public int getStudentId()
  {
   return studentId;
```

```
  }
  public String getStudentName()
  {
   return studentName;
  }
  public String getStreet()
  {
   return street;
  }
  public String getCity()
  {
   return city;
  }
  public ActionErrors validate(ActionMapping actionMapping,
HttpServletRequest request)
  {
   ActionErrors aes=new ActionErrors();
   if(studentId==0)
   {
    ActionError e1=new ActionError("id.req");
    aes.add("studentId",e1);
   }
  if(studentName==null || studentName.equals(""))
   {
    ActionError e2=new ActionError("name.req");
    aes.add("studentName",e2);
   }
   else
   {
    if(studentName.length()<5)
     {
      ActionError e3=new ActionError("name.min");
     aes.add("studentName",e3);
    }
   }
    System.out.println("Sucessfully executed.....");
    return aes;
 }
}
```

* In the above code we have use three error messages with that keys
    1 **id.req**
    2 **name.req**
    3. **name.min**

* The above 3 resources must be added to property files
In **application.propertys** we added
    id.req= student id is required
    name.req= student name is required

name.min = student name must be 5 characters are there

2. **NewStudAction.java**

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import org.apache.struts.action.*;
public class NewStudAction extends Action
{
 public ActionForward execute(ActionMapping mapping, ActionForm

form, HttpServletRequest request, HttpServletResponse response)

throws Exception
{
 System.out.println("executing bus logic.....");
 NewStudFB fb=(NewStudFB)form;
 Class.forName("oracle.jdbc.driver.OracleDriver");
 Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:
orcl","scott","tiger");
Statement stmt=con.createStatement();
String sqlstmt="insert into student

values("+fb.getStudentId()+",'"+fb.getStudentName()+"','"+fb.getSt

reet()+"','"+fb.getCity()+"')";
System.out.println("values inputed..."+sqlstmt);
stmt.executeUpdate(sqlstmt);
con.close();
return mapping.findForward("data-stored");
}
}
```

* In the above class as part of the **execute()** method the business logic is implemented in a big project it is better to implement the business logic as part of MODEL.

3. **sform.jsp**

```
<%@ taglib uri="/tags/struts-html" prefix="html"%>
<html:html>
<head>
        <title> New Student Form</title>
</head>
<body>
        <html:form action="/student.do">
```

```
<html:errors/>
            <b>studentId:</b>
            <html:text property="studentId"/><br>
            <b>studentName:</b>
            <html:text property="studentName"/><br>
            <b>street:</b>
            <html:text property="street"/><br>
            <b>city:</b>
            <html:text property="city"/><br><br>
            <html:submit property="login"/>
        </html:form>
</body>
</html:html>
```

Struts-config.xml

In form bean

```
<form-beans>
        <form-bean name=”NewStudForm” type=”NewStudFB”>
        </form-bean>
</form-beans>
```

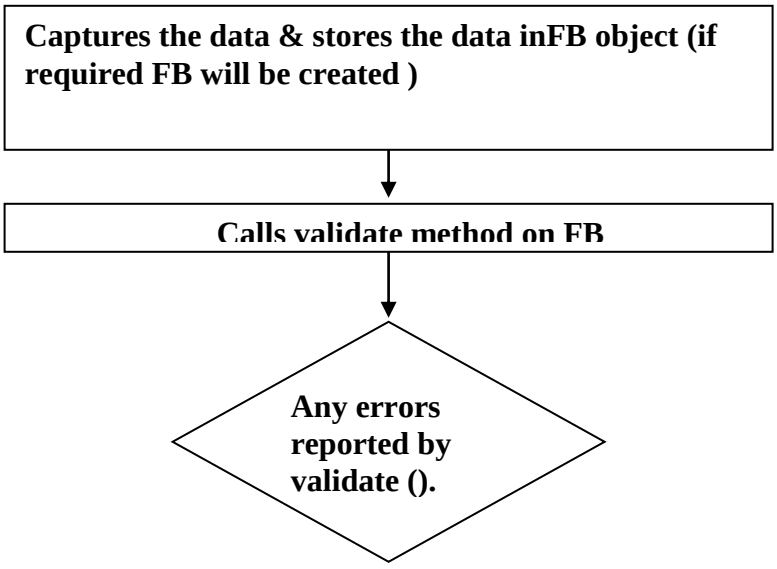
In Action mapping

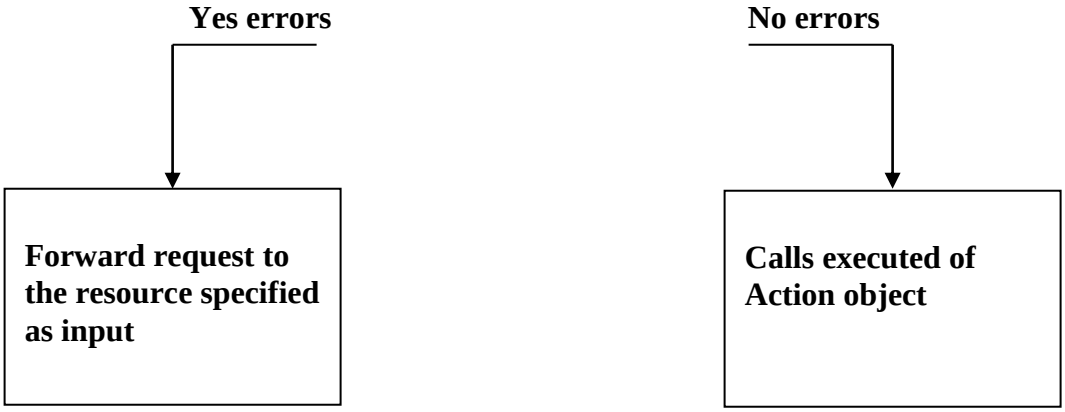
```
<action-mappings>
        <action name=”NewStudForm” path=”/student input=”/sform.jsp”
        validate=”true” scope=”request” type=”NewStudAction”>
        <forward name=”data-stored” path=”/ls.jsp”/>
        </action>
</action-mappings>
```

In the above project we have done steps:

1. name of the form: NewStudForm
2. name of the FB class: NewStudFB
3. name of the Action class: NewStudAction
4. forwards datastored : ls.jsp
5. name of the jsp: sform.jsp
6. the path used in the action tag: /student.do

struts flow chart in the above project



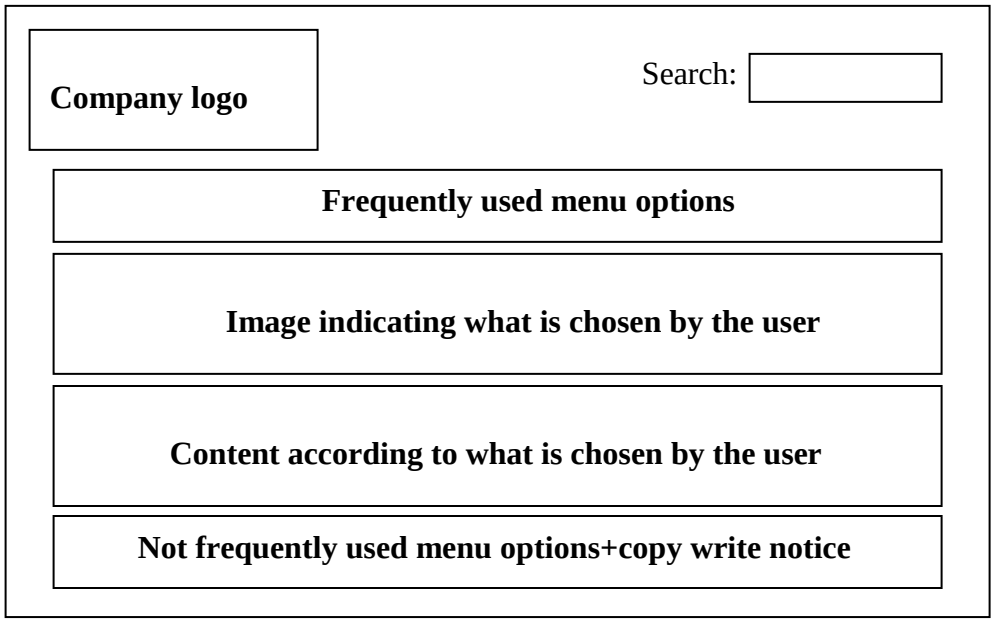


STRUTS TILES PLUGIN :

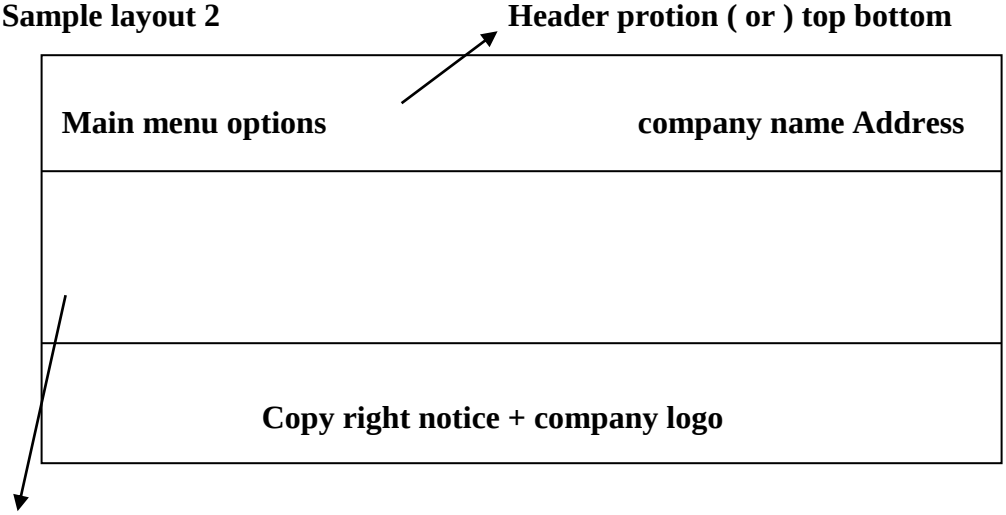
* As part of struts1.1 two additional softwares (PLUGINS) are provided
  1. **Validator plugin**(this is responsible for taking care of validations automatically).
  2. **TILES plugin** (this can be used to develop the applications with consistent look).

* In order to use tiles as part of a web application we must first decide about layout of the page.

Ex: sample layout 1



Ex: Sample layout 2



body portion(or) middle bond **footer portion (or) bottom band**

Ex : if we take a simple project that is even every page open the web page may be same look .

home.jsp

copy notice here image here

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>

<tiles:insert page="/layout1.jsp" flush="true">
  <tiles:put name="title"  value="This is first example on Tiles" />
  <tiles:put name="topband" value="/tb.jsp" />
  <tiles:put name="middleband" value="/cc.jsp" />
  <tiles:put name="bottomband"   value="/bb.jsp" />
</tiles:insert>
```

tb.jsp

```
        <head>
                <title> sudarshan company</title>
        </head>
        <a href="home1.jsp">home</a>

        <a href="products.jsp">products</a>

        <a href="service.jsp">services</a>

        <a href="contact.jsp">contacts</a>
```

layout.jsp

```
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<html>
        <head>
                <title:getAsString name="title"></title>
        </head>
        <body>
          <table>
                <tr><td><tiles:insert attribute="topband" />                    </td></tr>
                <tr><td><tiles:insert attribute="middleband"/>
        </td></tr>
                <tr><td><tiles:insert attribute="bottomband"/>
        </td></tr>
```

```
    </table>
  </body>
```

* layout.jsp provides the information about how the page has to be generated.
* we must use layout.jsp for generating all the pages.

hc.jsp

welcome to our home page

bb.jsp

copy notice here image here

cc.jsp

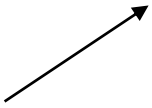
here are outer contact details.

products.jsp

```
    <%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<title>our products information.....</title>
<tiles:insert page="/layout1.jsp" flush="true">
  <tiles:put name="title"  value="This is first example on Tiles" />
  <tiles:put name="topband" value="/tb.jsp" />
  <tiles:put name="middleband" value="/cc.jsp" />
  <tiles:put name="bottomband"   value="/bb.jsp" />
</tiles:insert>
</a>
```

Same as services.jsp & contact.jsp and so on……

The ou put is……..



head portion

home products services contacts

here are outer contact details.

Body

copy notice here image here

Bottom

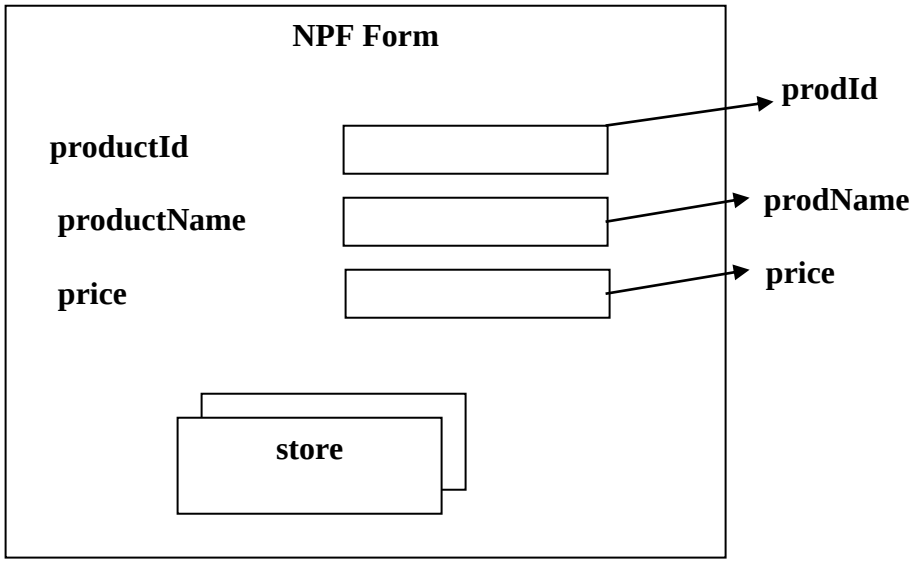
The out put is same as if we click in home link , products link, service link, contacts link.

DynaActionForm :

* **DynaActionForm** is a subclass of ActionForm class but this is more flexible then ActionForm class, we can use this class as super class of our **formBean** classes as shown below.

* As part of DynaActionForm **get & set** methods are provided these methods take care of setting the value of the property & returning the value of the property , the property can be of type **java.lang.String** (or) wrapper classes of java primitive data types (java.lang.INTEGER),(java.lang.FLOAT),(java.lang.BYTE).

Ex : sample project using DynaActionForm



Note : NPF Form

Validation Rules :

1. prodId is required.
2. prodName is required
3. price is required.

* In that we have set the class path is

set CLASSPATH=struts.jar;weblogic.jar;commons-beanutils.jar;.

1. Prob.java

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
public class ProbFB extends DynaActionForm
{
 //no instance variable for declaring...
 //no setters & getters methods.....
 public void reset(ActionMapping mapping,HttpServletRequest request)
 {
  set("prodId","11");
  System.out.println("reset method called.....");
```

```
 }
  public ActionErrors validate(ActionMapping mapping,HttpServletRequest request)
 {
  ActionErrors aes=new ActionErrors();
  String vpid=(String)get("prodId");
  if(vpid==null || vpid.equals(""))
  {
   ActionError e1=new ActionError("pid.req");
   aes.add("prodId",e1);
  }
  String vpName=(String)get("prodName");
  if(vpName==null || vpName.equals(""))
  {
   ActionError e2=new ActionError("pName.req");
   aes.add("prodName",e2);
  }
  String vprice=(String)get("price");
  if(vprice==null || vprice.equals("")){
     ActionError e3=new ActionError("pr.req");
   aes.add("price",e3);
  }
   System.out.println("Sucess fully values taken.....");
   return aes;
  }
}
```

2. NPAction.java

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
import java.io.*;
import java.sql.*;
public class NPAction extends Action
{
public ActionForward execute(ActionMapping mapping,ActionForm form,
HttpServletRequest request,HttpServletResponse response)throws Exception
 {
  ProbFB fb=(ProbFB)form;
  //we can get the values of proiperty using.
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  Statement stmt=con.createStatement();
  String sqlstmt="insert into products
values("+(String)fb.get("prodId")+","'+(String)fb.get("prodName")+"',"+(String)fb.get("price")
+")";
System.out.println("values inputed..."+sqlstmt);
stmt.executeUpdate(sqlstmt);
con.close();
```

```
return mapping.findForward("data-stored");
 }
}
```

3. npf.jsp

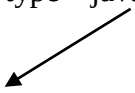
```
<%@taglib uri="/tags/struts-html" prefix="html"%>
<html:html>
<head>
        <title> New Product Form</title>
</head>
<body>
        <html:form action="/store.do">
                <b>Product Id:</b>
                <html:text property="prodId"/><br>
                <b>Product Name:</b>
                <html:text property="prodName"/><br>
                <b>price:</b>
                <html:text property="price"/><br>
                <br><html:submit property="store"/>
                <html:errors/>
        </html:form>
</body>
</html:html>
```

In Application.propertys

```
pid.req=Id required.....
pName.req= name required.....
pr.req= price required.....
```

In struts-config.xml

```
<form-beans>
        <form-bean name="NPForm" type="ProbFB">
      <form-property name="prodId" type="java.lang.String"/>
      <form-property name="prodName" type="java.lang.String"/>
      <form-property name="price" type="java.lang.String"/>
        </form-bean>
 </form-beans>
```



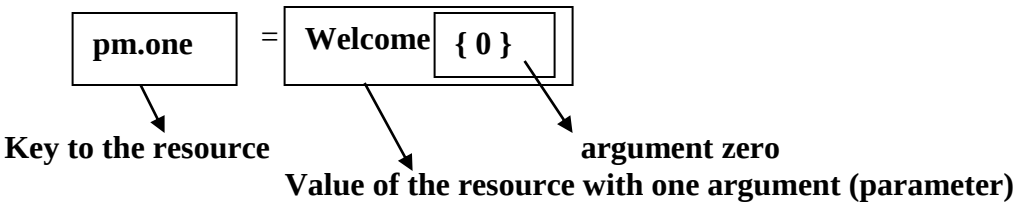
* In that What our project requirement we can do if suppose we use price is an intger then we use **type=”java.lang.INTEGER” ,** same as FLOA T and BYTE also.

```
<action-mappings>
        <action name="NPForm" path="/store" input="/npf.jsp" validate="true"
scope="request" type="NPAction">
        <forward name="data-stored" path="/npf1.jsp"/>
        </action>
```

</action-mappings>

* **DynaActionForm** can be configured according our requirements by using this we can reduce the total amount of code that has to be provided as part of our **FormBean** class.
* As part of struts1.1 validator plugin is provided the validator plugin software uses the information available on the files **validator-rules.xml,validatation.xml.**

* In **Application.properties**

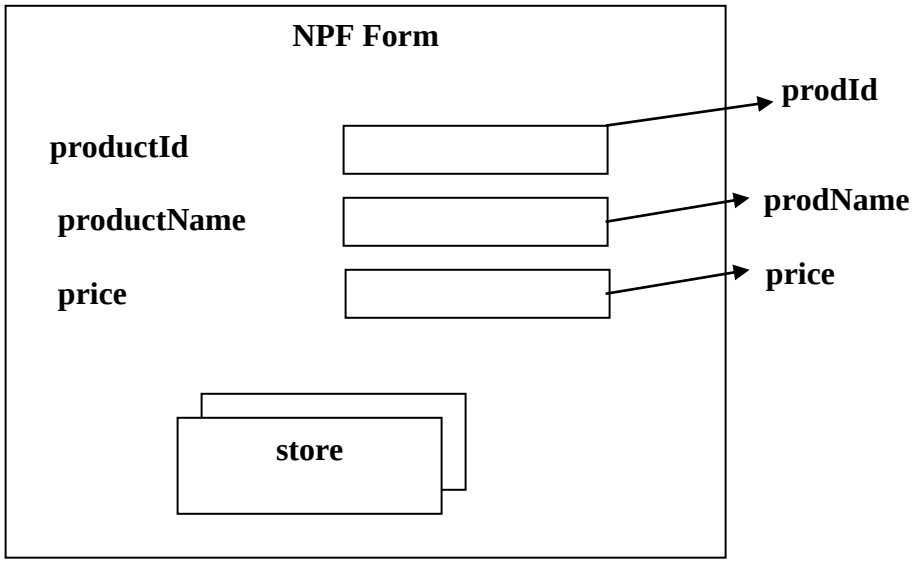


Ex: pm.jsp

```
<%@taglib uri="tags/struts-bean" prefix="bean"%>
<bean:message key="pm.one"/><br>
<bean:message key="pm.one" arg0="Mz.XYZ"/><br>
<bean:message key="pm.one" arg0="Mr.ABC"/>
```

* Struts support maximum size of arguments is 5 that means it supports only 5 arguments.
* As part of a message we can use a maximum of 5 parameters arg {0}------arg {4}.

Ex : simple project.



Note : NPV Form

Validation Rules :

4. prodId is required.
5. prodName is required

6. price is required.

* To use validator frame work for carrying out the validator form must be used as a super class of the **formBean** class.

1.**NPVForm.java**

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.apache.struts.validator.*;
public class NPVForm extends ValidatorForm
{
 private String prodId,prodName,price;
 public void setProdId(String p)
 {
  prodId=p;
 }
 public void setprodName(String pa)
 {
  prodName=pa;
 }
 public void setPrice(String ap)
 {
  price=ap;
 }
  public String getProdId()
  {
   return prodId;
  }
  public String getProdName()
  {
   return prodName;
  }
  public String getPrice()
  {
   return price;
  }
   public void reset(ActionMapping mapping,HttpServletRequest request)
 {
  prodId=null;
  prodName=null;
  price=null;
  System.out.println("reset method called.....");
 }
}
```

2. **NPVAction.java**

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
import java.io.*;
import java.sql.*;
public class NPVAction extends Action
{
 public ActionForward execute(ActionMapping mapping,ActionForm form,HttpServletRequest
request,HttpServletResponse response)throws Exception
 {
  NPVForm fb=(NPVForm)form;
  //we can get the values of proiperty using.
  Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","scott","tiger");
  Statement stmt=con.createStatement();
  String sqlstmt="insert into products values
("+(String)fb.getProdId()+","'+fb.getProdName()+"','+fb.getPrice()+")";
System.out.println("values inputed..."+sqlstmt);
stmt.executeUpdate(sqlstmt);
con.close();
return mapping.findForward("data-stored");
 }}
```

3. **npf.jsp**

```
<%@taglib uri="/tags/struts-html" prefix="html"%>
<html:html>
<head>
        <title> New Product Form</title>
</head>
<body>
   <html:form action="/store.do">
                    <b>Product Id:</b>
                    <html:text property="prodId"/><br>
                    <b>Product Name:</b>
                    <html:text property="prodName"/><br>
                    <b>price:</b>
                    <html:text property="price"/><br>
                    <br><html:submit property="store"/>
          </html:form>
                    <html:errors/>
</body>
</html:html>
```

* In **Struts.config.xml**

```
<form-beans>
        <form-bean name="NPVForm" type="NPVForm">
      </form-bean>
 </form-beans>
<action-mappings>
```

```
        <action name="NPVForm" path="/store" input="/npf.jsp" validate="true"
scope="request" type="NPVAction">
        <forward name="data-stored" path="/npf1.jsp"/>
        </action>
</action-mappings>
```

* if validate method is called on **prodId** object the JVM execute the validate method of ValidatorForm class.

* In **Validation.xml**

```
  <formset>                                          Name of the form
    <form name="NPVForm">                            Name of the field in the form
      <field  property="prodId"  depends="required">
                                                     Name of the validator
          <arg0 key="Product Id is" resource="false"/>
       </field>
     <field  property="prodName" depends="required">
       <arg0 key="Product Name is" resource="false"/>
     </field>
     <field  property="price" depends="required">
       <arg0 key="Price is" resource="false"/>
     </field>
    </form>
  </formset>
```

* **required** is called as a validator it checks whether the user has provided the value or not if the value is not provided it will be treated as an error.
* Integer validator checks whether the input is an integer or not if the input is not an integer it will be treated as an error similarly we have float ,long, short, double & Byte validators.

```
<form name="NPVForm">
        <field  property="prodId"  depends="required">
         <arg0 key="Product Id is" resource="false"/>
         </field>
```

* In our project we can set the validation in our login page ie in our prodId field we enter only integer other wise it give error message & if that field is empty then it will give error message ie as shown below in **validation.xml**

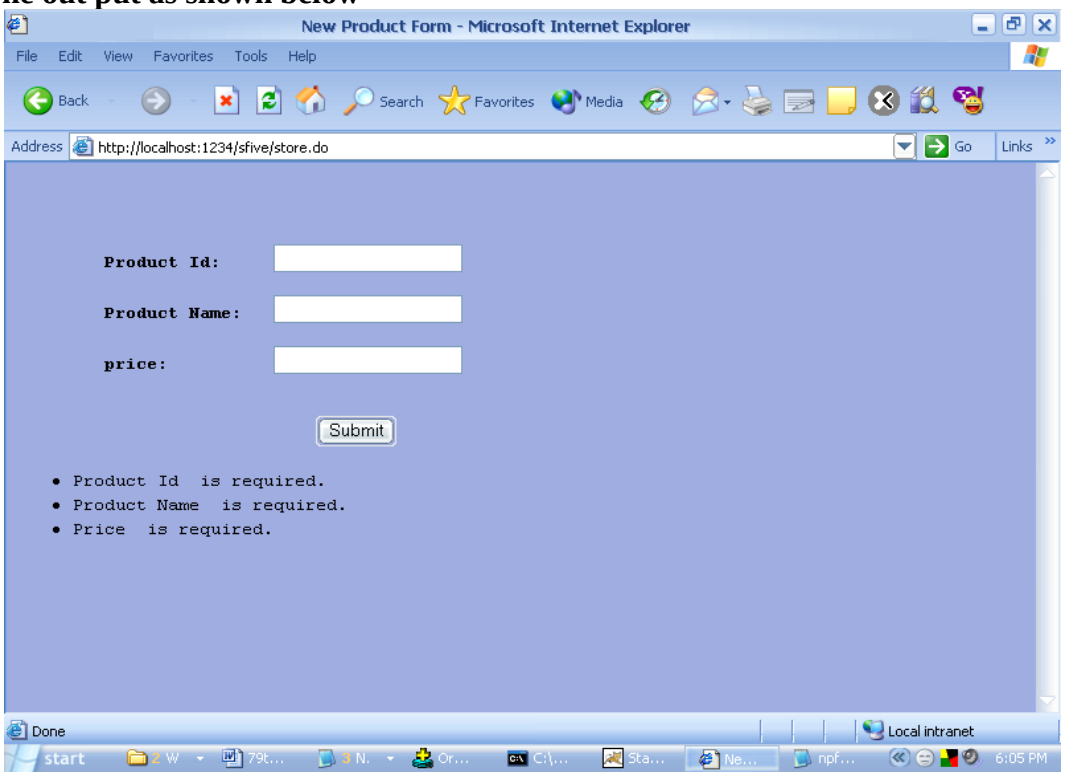
```
 <field  property="prodId" depends="required,integer">

  <arg0 key="Product Id " name="required" resource="false"/>
                                                     Used for errors.required

  <arg0 key="Product Id " name="integer" resource="false"/>
```

Used for errors.byte message

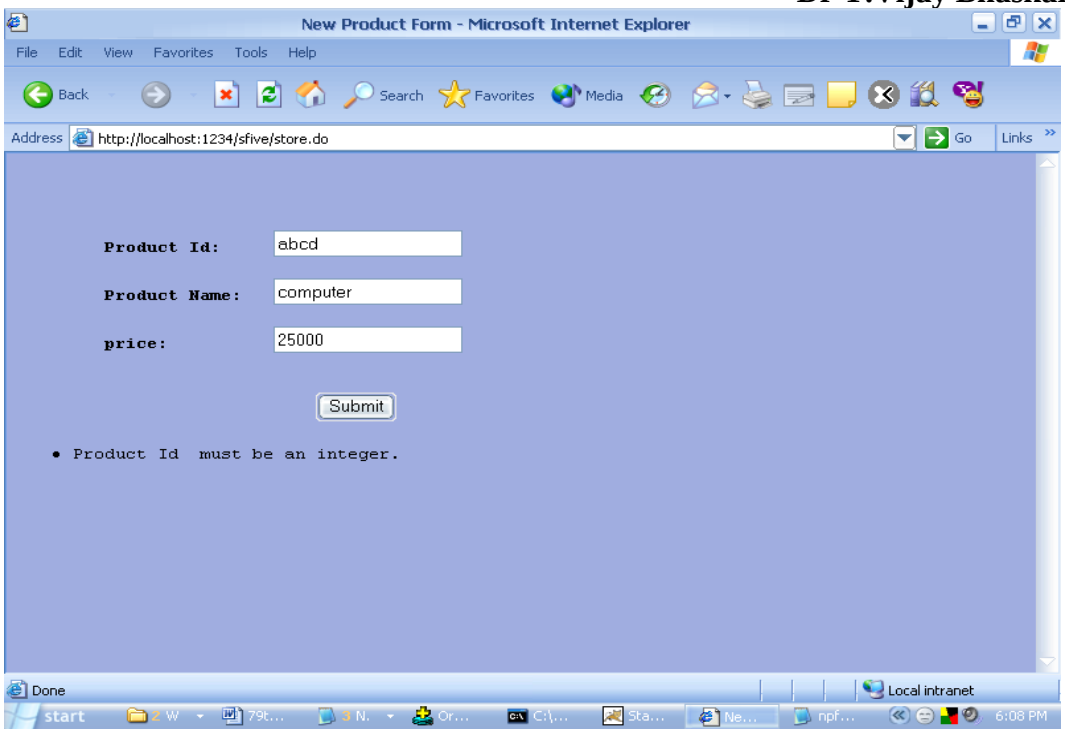
</field>

* And same thing we have to do in byte,float string date etc…ie our project requirement.

The out put as shown below



* In the above we doesn't fill the 3 fields so it will give 3 error messages.



* In the above we enter the **prodId id is characters** then it will show an error product Id must be an integer.
* We can use multiple validators on a single field by providing the configuration as shown below.
* If we use

```
<field property="prodId" depends="required,integer">
<arg0 key="Product Id " resource="false"/>
<arg0 key="Product Id " resource="false"/>
</field>
```

* The above we use same **arg0** will be used for **errors.required & errors.integer**

* If we write **resource="true"** then it will treated as a key then we mention the key in **Application.properties** files.ie as shown below

```
<field property="prodName"  depends="required">
<arg0 key="  " resource="true"/>
```

Then we will metion in **Application.properties** as shown below

errors.required={0} product Id is required.

Then the output will be is **product Id is required** will be displayed in if we don't give Product Name value.

* If we write **resource="false"** then it will give directly it is not required in **Application.properties**.

* If we need to check whether the value of an input field is between a particular a minimum & maximum value we can use **intRange** validator this validator takes min & max as the variables.

* If we take previous project then we need to check **Product Id** in the range of 1 to 1000 & also it checks the number is integer & the field must be filled so we have to write below changes.

Validation.xml

```
<field property="prodId" depends="required,intRange">
<var>
        <var-name>min</var-name>
        <var-value>1</var-value>
</var>
<var>
        <var-name>max</var-name>
        <var-value>1000</var-value>
</var>
<arg0 key="Product Id" resource="false"/>
<arg0 key="Product Id" resource="false"/>
<arg1 key="1" resource="false"/>
<arg2 key="1000" resource="false"/>
</field>
```

* **float** range can be used to specify a minimum floating point value & maximum floating point value ie allowed for an input field this validator uses the variables min & max.

We can change in
<field property="prodId" depends="required,floatRange">
And remaining same thing above mentioned.

* **Email validator:**

* **email validator** also we can use in this we can use one argument only.
ie

```
        <field
        property="prodName"
        depends="email">
        <arg0 key="Email Id " resource="false"/>
      </field>
```

* If we not enter the email id then it will give error is **Email Id is required** & email id should be **@** and **.com** compulsory if we give invalid id then the error is

- **`Email Id is an invalid e-mail address.`**

* **Date validator:**

* **Date validator** uses the variable **deletePatternStrict**

```
        <field property="price" depends="required,date">
        <var>
           <var-name>datePatternStrict</var-name>
           <var-value>dd/MM/yyyy</var-value>
        </var>
```

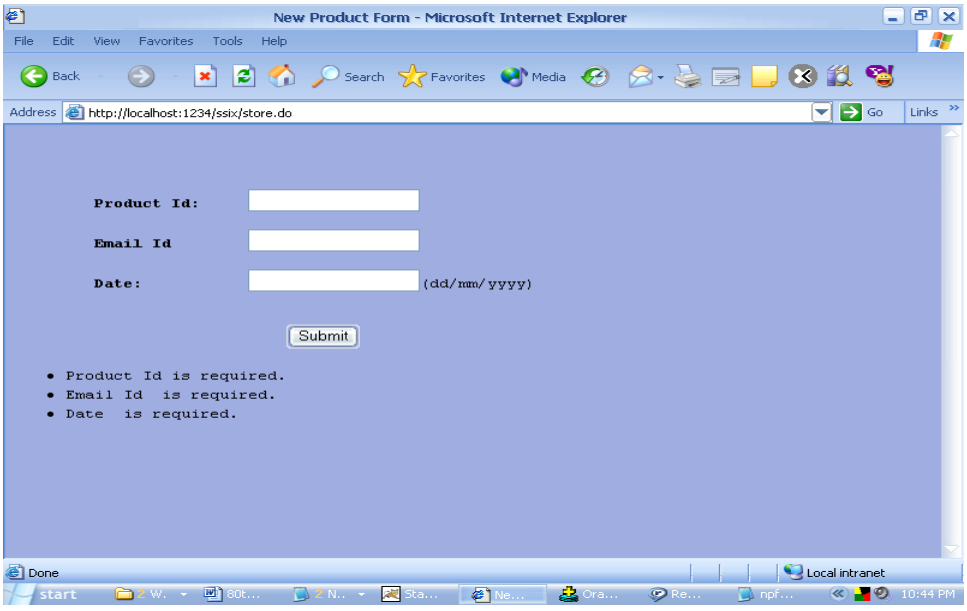
<arg0 key="Date " resource="false"/> </field>

* In the above example we have use the variable dataPatternStrict in this case for a Date like **February / fourth / 2005** the user must enter **04/02/2005** , If we need to accept the same date as **4/2/2005** instead of **datePatternStrict** we must use datePattern

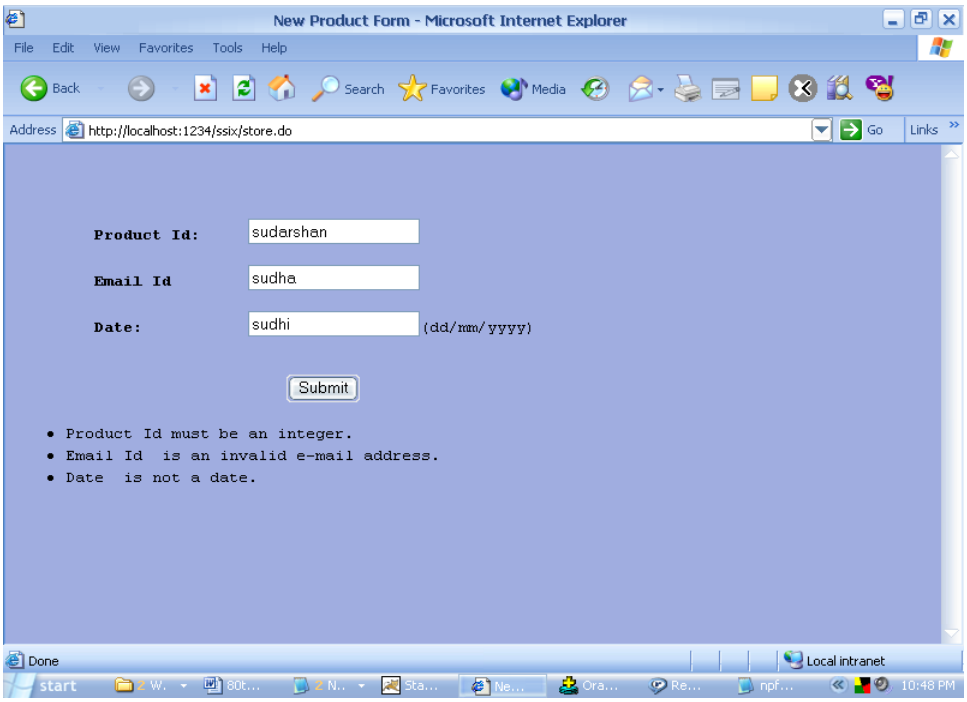
<var-name>datePattern</var-name>

<var-value>dd-mm-yyyy</var-value>

* As shown three fileds id,email,date formats then we show the below picture.
* If we doesn't input the values submitted the empty form then result will be shown in below figure.



* If we mention in wrong types in the fields then it will show the errors that is shown in the below figure……



* minlength is variable which is set to minimum length.

```
 <field property="prodName" depends="minlength,maxlength">
 <var>
      <var-name>minlength</var-name>
      <var-value>4</var-value>
</var>
<var>
      <var-name>maxlength</var-name>
      <var-value>8</var-value>
</var>
     <arg0 key="Product Name" name="minlength" resource="false"/>
    <arg1 key="4" name="minlength" resource="false"/>
    <arg0 key="Product Name" name="maxlength" resource="false"/>
     <arg1 key="8" name="maxlength" resource="false"/>
    </field>
      <field property="price" depends="required">
        <arg0 key="Price " resource="false"/>
       </field>
```

* Credit card validator is used to check whether the credit card no is valid enter by the user.
* The information about the validators are provided as part of validator-rules.xml.

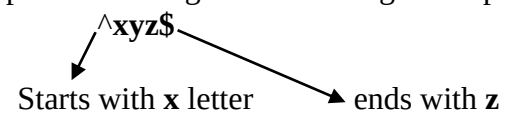
* **mask** is the name of the validator it uses the variable like mask.

```
 <field property="prodId" depends="mask">
 <var>
<var-name>mask</var-name>
<var-value>xyz</var-value>
</var>
<arg0 key="Product Id " resource="false"/>
 </field>
```

* In regular expression there is some special characters that type characters is called meta character ie ^

^xyz$ this starts with **x** character.

* As part of Regular Expressions we can use characters like ^ **(charat)** , **$** (**doller**) etc with special meaning if we use a regular expression as shown below as input only **xyz** characters.



^[a-f]yz$

It means a to f characters

* To allow the range of chars (or) range of numbers.
* We can use expressions like

[a-f] it allows any characters between **a** to **f**
[a-fx-z] it allows **a,b,c,d,e,f,x,y,z**
[1-3] it allows **1,2,3**
^**[1-3]** it allows except **1,2,3**
(abc)|(def) it allows **abc** or **def**.

* Ro accept a 5 digit telephone number we can use the following expression
^[0-9][0-9][0-9][0-9][0-9]$

* We can use **\d** to accept a digit so the above expression can be rewritten as **^\d\d\d\d\$**
Ie it allows Must be 4 numbers.

* We can use **\D** it allows anything other then digits. If we write how many D digits then it accept that many no-of chars.

x* allows **0** (or) **more then 0** x characters.

^**ax?as$** in that **x?** allows **0** (or) **1** x character.

^**ax+a$** in that **x+** allows **1** (or) **more** characters.

* → **0** (or) **more**
? → **0** (or) **1**
+ → **0** (or) **more**

^**\dx{5}** it allows exactly 5 characters.

^**x{3,6}$** it allows minimum 3 and maximum 6 characters allowed.

^**x{3, }** it is minimum allowed 3 characters.

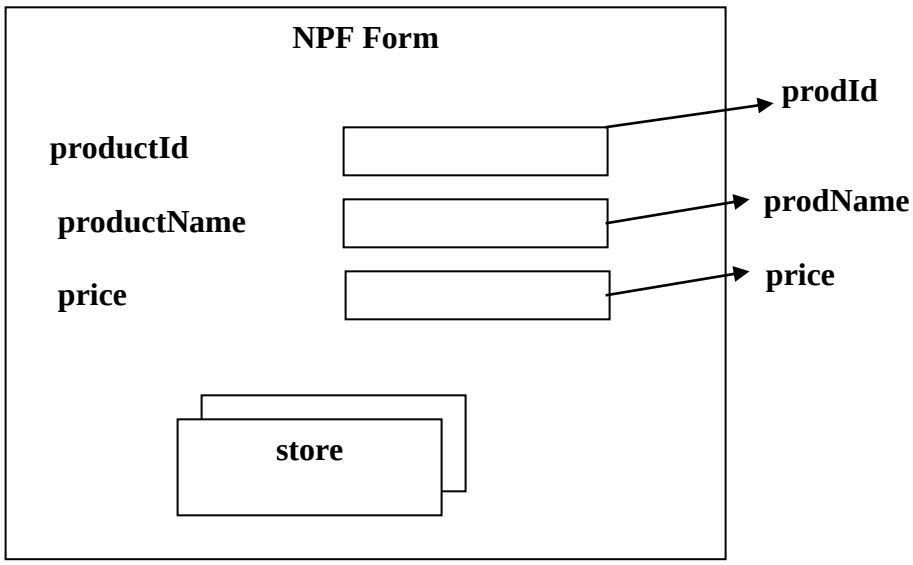
\d{3} (| -) d{3}$ it is an postal code like 500 001 (or) 500-001.

* **\s** can be used to allow white space character (tab or new line)

* **\S** can be used to accept anything except white spaces .
* **\w** it accepts a "word" character (alphanumeric plus "_") ie it allowed **a-z** , **A-Z**.

* **\W** it accept other then non word character.
* Some of the validator can't be carried out by the validator frame work for Ex : in the below form……
* A validation rule like the value of the price field can't exceed 5 times the value of **prodId** field.

Ex : simple project.



Note : NPV Form

* As part of our **formBean** class we must provide the validate method.

* We can use java script tag available in html tag library of struts as shown below this tag generates the java script with a function validate validateNPVForm for generating this function the information provided in validation.xml , validate-rules.xml will be used.
* The function generated must be called when the user submit the form for this as part of the jsp in the form tag we must on submit attribute as shown below
<html:form action="/store.do" onsubmit="return validateNPVForm(this);">

* must of the programmers prefer using **DynaValidatorForm** we can assume this class to be a combination of DynaActionForm & validator Form ie if we create a sub class of this class we need not provide getters & setters & we need not provide validate method if all the validation can be carried out automatically by the validator frame work.

* **MVC** (model-view-controller) is a very popular design pattern used for the development of business applications struts frame work is designed based on MVC.
* In this architecture the software is divided in to three different pieces.

1. **model :**
Is a piece of software ie responsible for managing the data (we implement business logic that manipulates the data according to the business requirement)

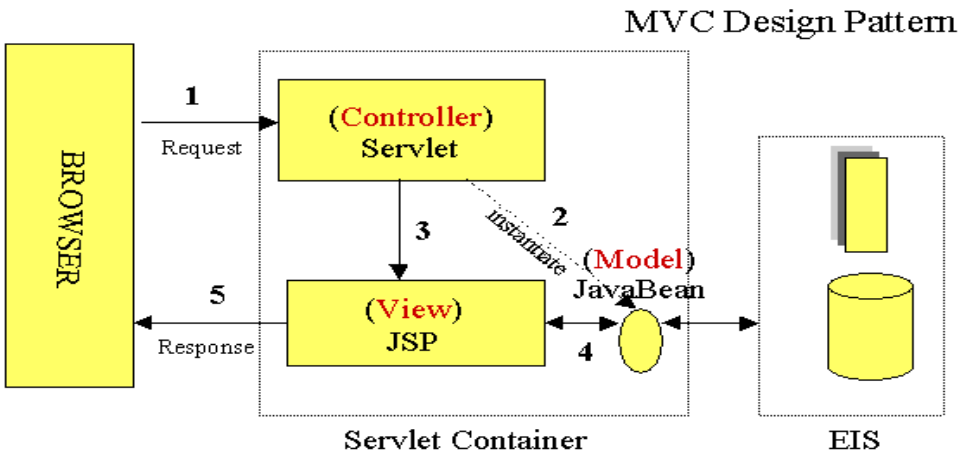
2. **view :**

Is a piece of software that is responsible for presenting the information to the user In struts based application we use java server pages to generate the view ie presented to the user by the browser.

3. **controller :**

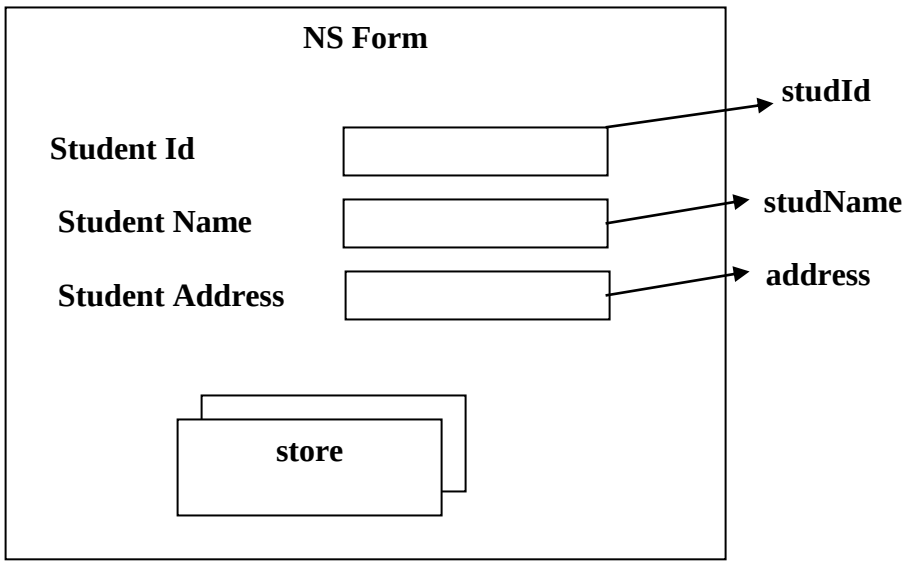
Is piece of software ie responsible for taking a decision on what as to be done by the application when the user interacts with the application.

Ex: As part of a web application the user interacts with the web application by clicking on various links by submitting the forms.



Ex : simple project :

* As part of struts the ActionServlet + FormBean + Action Code is called the controller



Note : NS Form

Validation Rules :

7. studId is required.
8. studName is required
9. address is required.

1.**StudModel.java**

```
import java.sql.*;
public class StudModel
{
 public void storeStudent(String studId,String
studName,String address) throws Exception
{
 System.out.println("Student busLogic.....");
   Class.forName("oracle.jdbc.driver.OracleDriver");
  Connection

con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:

1521:orcl","scott","tiger");
 Statement stmt=con.createStatement();
String sqlstmt="insert into Student1

values('"+studId+"','"+studName+"','"+address+"')";
System.out.println("sql stament is....."+sqlstmt);
stmt.executeUpdate(sqlstmt);
con.close();
 }
}
```

* The above classs (model) contain business logic this code can be use as part of struts based web application a GUI application developed using AWT (or) JFC.

If the business logic is implemented directly as part of formBean (or) in the Action classes we can use that code only as part of struts based application.

2. **StudFB.java**

```
import javax.servlet.http.*;
import org.apache.struts.action.*;
import org.apache.struts.validator.*;
public class StudFB extends DynaValidatorForm
{
 //no instance variable for declaring...
 //no setters & getters methods.....
 public void reset(ActionMapping mapping,HttpServletRequest

request)
 {

  System.out.println("reset method called.....");
 }
}
```

3.**NSAction.java**

```
import javax.servlet.http.*;
import java.io.*;
import org.apache.struts.action.*;
public class NSAction extends Action
{
 public ActionForward execute(ActionMapping mapping,ActionForm
form,HttpServletRequest request,HttpServletResponse response)
 {
  StudFB fb=(StudFB)form;
  try
   {
    StudModel sm=new StudModel();
sm.storeStudent((String)fb.get("studId"),(String)fb.get("studName"),(S
tring)fb.get("address"));
return mapping.findForward("sucess");
 }

   catch(Exception e)
   {
    return mapping.findForward("failure");
   }
  }
}
```

* The Action class code is response for invoking the business logic provided as part model & return a forward.

Struts-config.xml

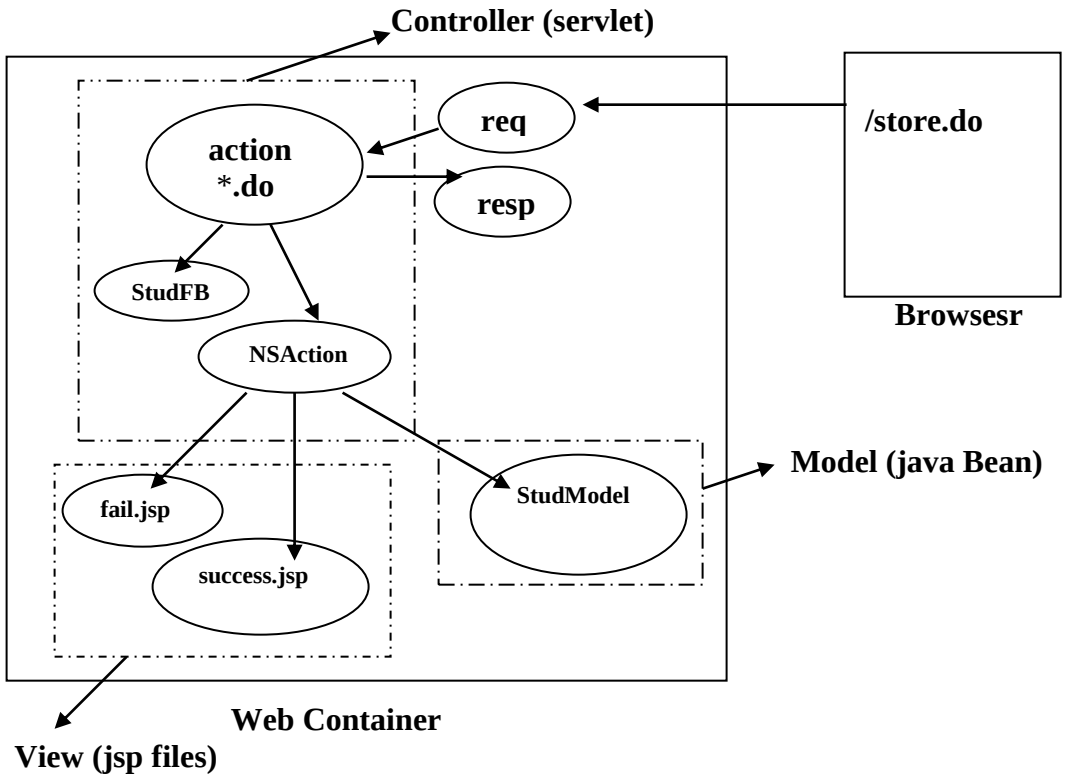
```
<form-beans>
        <form-bean name="NSForm" type="StudFB">
<form-property name="studId" type="java.lang.String"/>
<form-property name="studName" type="java.lang.String"/>
<form-property name="address" type="java.lang.String"/>
        </form-bean>
 </form-beans>

<action-mappings>
        <action name="NSForm" path="/student" input="/sform.jsp" validate="true"
scope="request" type="NSAction">
        <forward name="sucess" path="/sform1.jsp"/>
      <forward name="failure" path="/sform2.jsp"/>
        </action>
</action-mappings>
```

Validation.xml

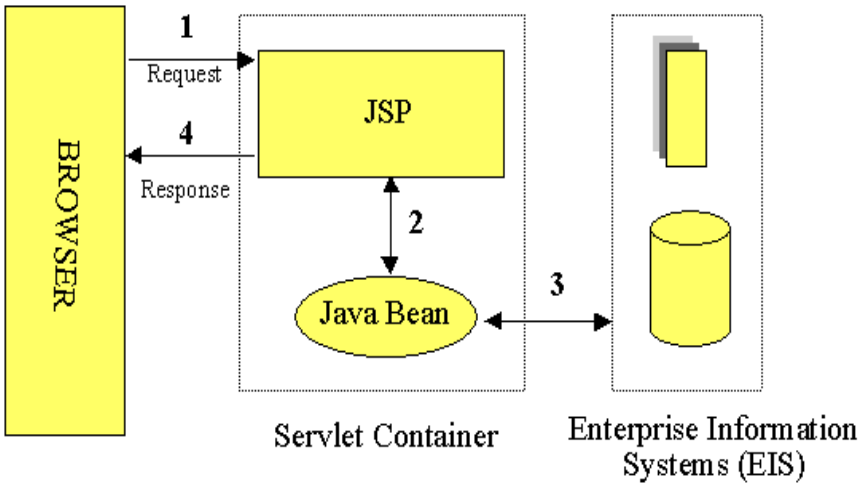
```
<formset>
    <form name="NSForm">
 <field property="studId" depends="required,integer">
 <arg0 key="Student Id " resource="false"/>
 <arg0 key="Student Id " resource="false"/>
 </field>
 <field property="studName" depends="required">
 <arg0 key="Student Name " resource="false"/>
 </field>
 <field property="address" depends="required">
 <arg0 key="Address " resource="false"/>
  </field>
 </form>
</formset>
```

* Jsp page is responsible to generate views in struts.

Ex: MVC 2



Ex :MVC 1



* **JSP Model 1** is called as **MVC 1** by java soft according to java soft in this architecture the java bean that contains the business logic has to be considered as model & jsp file should be consider as controller & view.
* The original **MVC** is called as **JSP Model 2** (or) **MVC 2** by java soft.
* As part of struts frame work apart from Action class sub classes of Action class like **ForwardAction , IncludeAction , DispatchAction etc** are provided.

<action parameter = "/xxx.jsp" path="/xxx"
type = "org.apache.struts.action.ForwardAction"/>

* Similar to servlet chaining we can use action chaining ie one action can forward the request to another action.

<action parameter ="/one.jsp" path="/xxx"
type = "org.apache.struts.actions.ForwardAction"/>

* If we place the above ActionMapping in struts-config.xml we can access **one.jsp** by using **xxx.do** as a URL in the Browser.
* When we use **/xxx.do** as a URL in Browser the web container invokes the ActionServlet & the ActionServlet invokes the **execute()** method of ForwardAction reads the value of parameter & forwards the request to the resource specify as a value of the parameter.
* ActionServlet internally makes use of the code that is part of request processor.
* If we want to use a different way for processing a request we provide our own subclass of request processor and we must provide the information about our own request processor as part of **struts-config.xml** as shown below

<controller processorClass = "org.students.ourRequestProcessor"/>

* If we develop a project we need to store the information about several actions as part of a single configuration file. It will be very difficult to organize the information. as part of **struts 1.1** a facility is provided to use multiple configuration files (one configuration file per module)
* To add the information about multiple configuration files as part of struts project we must provide the information as shown below in **web.xml**

<init-param>

```
    <param-name>config</param-name>
    <param-value>/WEB-INF/struts-config.xml</param-value>
   </init-param>
```

* If we use multiple configuration files

<param-name>config/mone</param-name>

* **mone** is called as the name of the module to access the action that are part of this module in the URL we must use **/mone** after context path.

Ex: http://localhost:8080/saone/mone

* We can place any no.of actions as part of various modules
* There may be problem if we use the JSP file directly in the URL for generating the form. While using it as part of a module to solve it we can provide a simple action as shown below.

* **Serialization :** Is the process of writing the state (information) of the object.
* **Di serialization :** The process of creating the object by reading the state of the object from the stream.

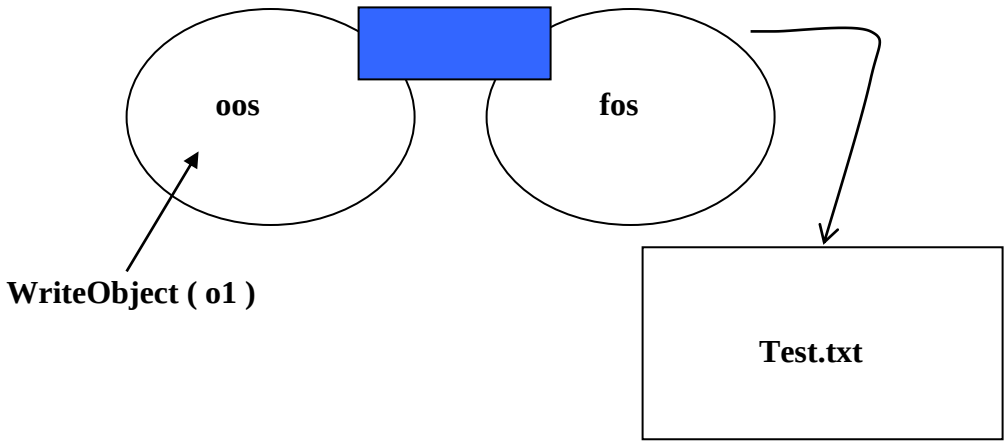
Ex:

1. **Stud.java**

```
public class stud
{
 String age;
 String name;
 public void setAge(String a)
 {
  age=a;
 }
  public void setName(String a)
  {
   name=a;
  }
 public String getAge()
 {
  return age;
 }
 public String getName()
 {
  return name;
 }
  public void print()
  {
   System.out.println("Age is....."+age);
   System.out.println("Name is......."+name);
  }}
```

2.**ser.java**

```
import java.io.*;
public class ser
{
 public static void main(String args[]) throws Exception
 {
  FileOutputStream fos=new FileOutputStream("sudhi.txt");
  ObjectOutputStream oos=new ObjectOutputStream(fos);
  Integer o1=new Integer(123);
  oos.writeObject(o1);
 oos.close();
 }
}
```



* In the above example ObjectOutputStream object is created by passing FileOutputStream object as a parameter (this establishes a link between **oos & fos**

When ObjectOutputStream.write object is called the pointed by **o1** will be written to **fos** & **fos** writes this information to the file **sudhi.txt**

Ex: ser.java

```
import java.io.*;
public class ser
{
 public static void main(String args[]) throws Exception
 {FileOutputStream fos=new FileOutputStream("sudhi.txt");
  ObjectOutputStream oos=new ObjectOutputStream(fos);
  stud o1=new stud();
  o1.setAge("25");
  o1.setName("Sudha");
  o1.print();
  oos.writeObject(o1);
 oos.close();
```

```
 }
}
```

Out put is :

```
Age is.....25
Name is.....Sudha
Exception in thread "main" java.io.NotSerializableException: stud
        at java.io.ObjectOutputStream.writeObject0(ObjectOutputStream.java:1054)

        at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:278)
        at ser.main(ser.java:11)
```

* The above code fails & throws the Exception **java.io.NotSerializableException** When the application calls write object method this indicates that the information about **stud** object can't be serialize (or) return.

* **integer , float , String are** Serializable **socket , stud** are not Serializable .

Ex:

1. **ClsOne.java**

```
public class ClsOne
{
}
```

2.**ClsTwo.java**

```
import java.io.Serializable;
public class ClsTwo implements Serializable
{
}
```

3. **App.java**

```
import java.io.*;
import java.net.*;
public class App
{
 public static void main(String args[]) throws Exception
 {
  //ClsOne o = new ClsOne();

  ClsTwo o = new ClsTwo();

  if(o instanceof java.io.Serializable)
  {
```

```
   System.out.println("Yes impl...Serializable");
  }
   else
   {
    System.out.println("Not impl...Serializable");
  }
 }
}
```

* We can write above code to check whether the object implements Serializable we use the operator **instanceof**
* An interface without methods called tagged interface (or) marked interface.
* When we develop our own classes to make the objects of our own classes Serializable we must implement **java.io.Serializable** interface as shown below

Ex: stud.java

```
import java.io.Serializable;
public class stud implements Serializable
{
 String age;
 String name;
 public void setAge(String a)
 {
  age=a;
 }
  public void setName(String a)
  {
   name=a;
  }
 public String getAge()
 {
  return age;
 }
 public String getName()
 {
  return name;
 }
  public void print()
  {
   System.out.println("Age is....."+age);
   System.out.println("Name is......."+name);
  }
}
```

2. **ser.java**

```
import java.io.*;
```

```
public class ser
{
 public static void main(String args[]) throws Exception
 {FileOutputStream fos=new FileOutputStream("sudhi.txt");
  ObjectOutputStream oos=new ObjectOutputStream(fos);
  stud o1=new stud();
  o1.setAge("25");
  o1.setName("Sudha");
  o1.print();
  oos.writeObject(o1);
 oos.close();
 }
}
```

Output is :
Age is.....25
Name is.....Sudha

And it will create a sudhi.txt file in that some code will be there

Ie ¬í □sr □studðcšðL□í‹□ □L □aget □Ljava/lang/String;L □nameq ~ □xpt □25t Sudha

* Serializable ,singleThreaded Model enterprise Bean, remote (or) interfaces without methods.
* In java these kind of interfaces are used to provide some kind of information about an object that's why these kind of interfaces are known as tagged (or) marker interfaces.

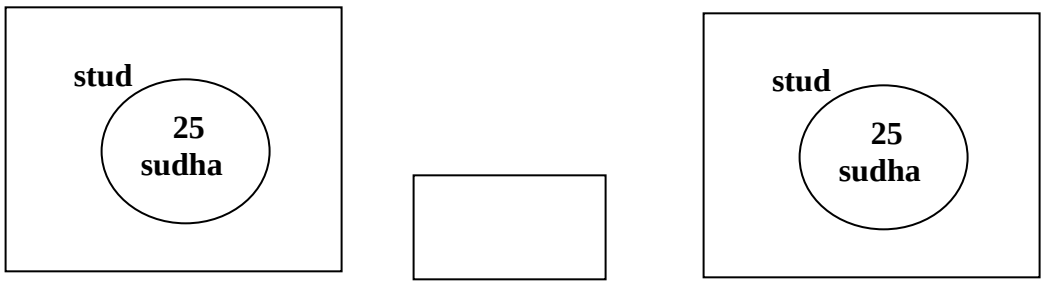
DeSerialization : The process of creating the object by reading the state of the object from the Stream.

Ex: Dser.java

```
import java.io.*;
public class Dser
{
 public static void main(String args[])throws Exception
 {
  FileInputStream fis=new FileInputStream("sudhi.txt");
  ObjectInputStream ois=new ObjectInputStream(fis);
  Object o=ois.readObject();
  stud s =(stud)o;
  s.print();
 }
}
```

* Some of the important objects Serializable are **java.langRuntime, java.net.Socket ,java.sqlConnection,java.sql.Resultset**.

RMI



* In the above examples we have used FileOutputStream to write the information about the object instead of **fos**. If the program uses SocketOutputStream the information about the object will be sent to another program.
* The second program can setup the link between SocketOutputStream & ObjectOutputStream. Using ObjectOutputStream the second program can DeSerializable the object (ie an object created by program 1 can be copy to program 2 using serialization & DeSerialization technique.

Note : RMI (Remote Method Invocation) internally uses serialization , Deserialization to pass the parameters & the return type.

* SocketOutputStream is used to send the data in another program.
* SocketInputStream is used to read the data in another program.
* A static field is a Field is not related to any object.

* When The object is serializable then it is not support a static variables.

Ex: stud.java

```
String age=null;
transient String name=null;
static String sd=null;
```

* When **stud** object is serialized only age will be serialized ,name will not be serialized as it is declared as transient .
* **sd** will not be serialized as it is declared as static.

Ex: stud.java

```
String age=null;
String name=null;
Socket sock=null;
```

* If we develop stud class with the following instance variables serialization fails as socket object is not serializable to solve this problem we must declare the sock variable as **transient**.
* Problem can be solved by providing the code as shown below.

```
String age;
String name;
transient Socket sock;
```

Ex: stud.java

```
String age=null;
transient String name=null;
static String sd=null;
```

* If the object based on the above class is serialized the information about **sd** will not be serialized,In a project if we have a **special** requirement like writing the information about static variable **sd** also then we must provide special methods **readObject** , **writeObject** as shown below.

Ex: stud.java

```
import java.io.*;
public class stud implements Serializable
{
 String age=null;
 String name=null;
 static String sock=null;
 private void writeObject(ObjectOutputStream out)throws IOException
 {
  System.out.println("write object executed.....");
  out.writeObject(name);
```

```
  out.writeObject(sock);
  out.writeObject(age);
 }
private void readObject(ObjectInputStream in)
                                        throws IOException,ClassNotFoundException
 {
  System.out.println("read object executed.......");
  age=(String)in.readObject();
  sock=(String)in.readObject();
  name=(String)in.readObject();
 }
   public void setAge(String a)
 {
  age=a;
 }
  public void setName(String a)
  {
   name=a;
  }
 public String getAge()
 {
  return age;
 }
 public String getName()
 {
  return name;
 }
  public void print()
  {
   System.out.println("sock is......"+sock);
   System.out.println("Age is......"+age);
   System.out.println("Name is......."+name);
  }
}
```

* If we run the **ser.java** then the output is

```
sock is......null
Age is......25
Name is.......Sudha
write object executed.......
```

* If we run the **dser.java** then the output is

```
read object executed.......
sock is......null
Age is......Sudha
Name is.......25
```

* special methods are **writeObjet , readObject**.
* When we serialize the **stud** object using (ObjectOututStream) **oos.writeObject()** it will call the writeObject provided in our **stud** class , similarly when we DeSerialize using **ois.readObject()** it will internally call the **readObject** method provides as part of **stud** class.

readObject & writeObject methods of stud class are declared as **private** but the code of ObjectOutputStream & ObjectInputStream can call these private methods.
* The code that invokes the writeObject & readObject are implemented as native methods (**c language**) & this code is considerd as part of **JVM**.

* java soft has defined a format called as object Serialization protocol for writing the information an object to a Stream . Any other company can define some other format of writing the information about the objects for this classes must **implement Externalizable interface** which is sub interface of **Serializable**.
* As part of Externalizable interface **readExternal & writeExternal** methods are provided.

These are two types of java objects

1. **local object**
2. **Remote object**

* The JVM identifies an object as remote object if it provides the implementation of the interface **java.rmi.Remote** interface.

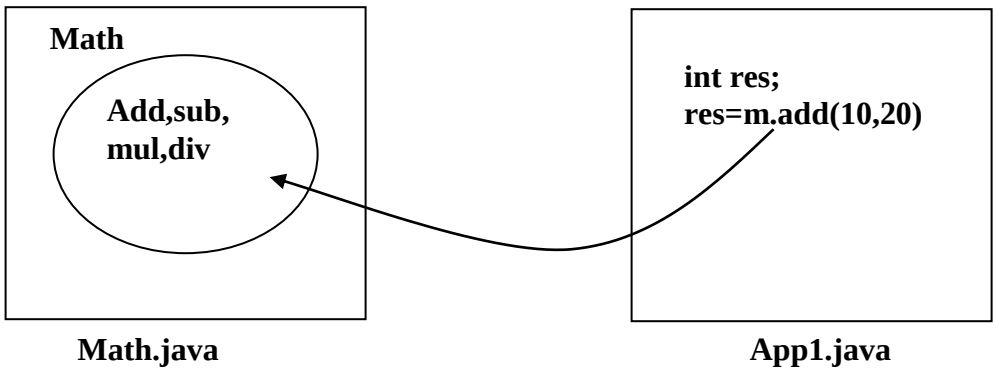
Ex: Math.java

```
public class Math
{
 public int mul(int a, int b)
 {
  System.out.println("Executing multiplication.....");
  return a*b;
 }
 public int div(int a, int b)
 {
  System.out.println("Executing Division.....");
  return a/b;
 }
 public int add(int a, int b)
 {
  System.out.println("Executing Addition.....");
  return a+b;
 }
 public int sub(int a, int b)
 {
  System.out.println("Executing Subtraction.....");
  return a-b;
 }
}
```

* The objects based on the above class are called as **local objects ,** a local object is an object that can be used from the code that is running in the JVM in which the objects is created.

Ex : App1.java

```
public class App1
{
 public static void main(String args[]) throws Exception
 {
  Math m=new Math();
  int res=m.add(10,20);
  System.out.println(res);
}
}
```



* By using RMI (Remote method Invocation) we can develop the java application running in one JVM that can invoke the methods on the objects that are created as part of another JVM (called as remote JVM , foreign JVM).
* The java programs using RMI technology can be executed on different physical machine.
* We can't use the above math class to create the objects that can be used by the code running in another JVM so we can't call the math object as remote objects.

PROCEDURE FOR CREATING A REMOTE CLASS :

Step 1: Create a remote interface.

```
public class One
{ }
```

Note : One is not a Remote Interface

```
public interface Two extends java.rmi.Remote
{ }
```

Note : Two is a Remote interface.

```
public interface Three extends java.rmi.Remote
{ }
```

Note: Three is a Remote interface (it extends Remote interface only)

Ex:

```
import java.rmi.*;
public interafce RIMath extends Remote                1
{
 public int add(int a,int b)throws RemoteException;                2

 public int sub(int a,int b)throws RemoteException;

 public int mul(int a,int b)throws RemoteException;                3

 public int div(int a,int b)throws RemoteException;
}
```

1. The interface must be a sub interface of **java.rmi.Remote** interface.
2. Every method in this interface must throw **java.rmi.RemoteException**
3. The parameters of the Remote methods & the written types of the remote method can be
   a. java primitive data types
   b. Serializable java classes.
   c. Remote interfaces.

Step 2 : Provide a remote class by implementing the remote interface

Ex: MathCls.java

```
import java.rmi.server.unicast;
import java.rmi.*;
public class MathCls extends java.rmi.server.UnicastRemoteObject implements RIMath{
 public MathCls() throws RemoteException{
  super();
 }
  public int add(int a, int b)throws RemoteException  {
   System.out.println("Executing Addition method.......");
   return a+b;
  }
  public int sub(int a, int b)throws RemoteException {
   System.out.println("Executing Addition method.......");
   return a-b;
  }
  public int mul(int a, int b)throws RemoteException  {
   System.out.println("Executing Addition method.......");
   return a*b;
  }
  public int div(int a, int b)throws RemoteException  {
   System.out.println("Executing Addition method.......");
   return a/b;
  }
}
```

* Here we not added the constructor so it is added **0** (zero) argument constructor but it is problem that it is added super class constructor ie **UnicatRemote** object so we have throw **RemoteException**

* After that compile

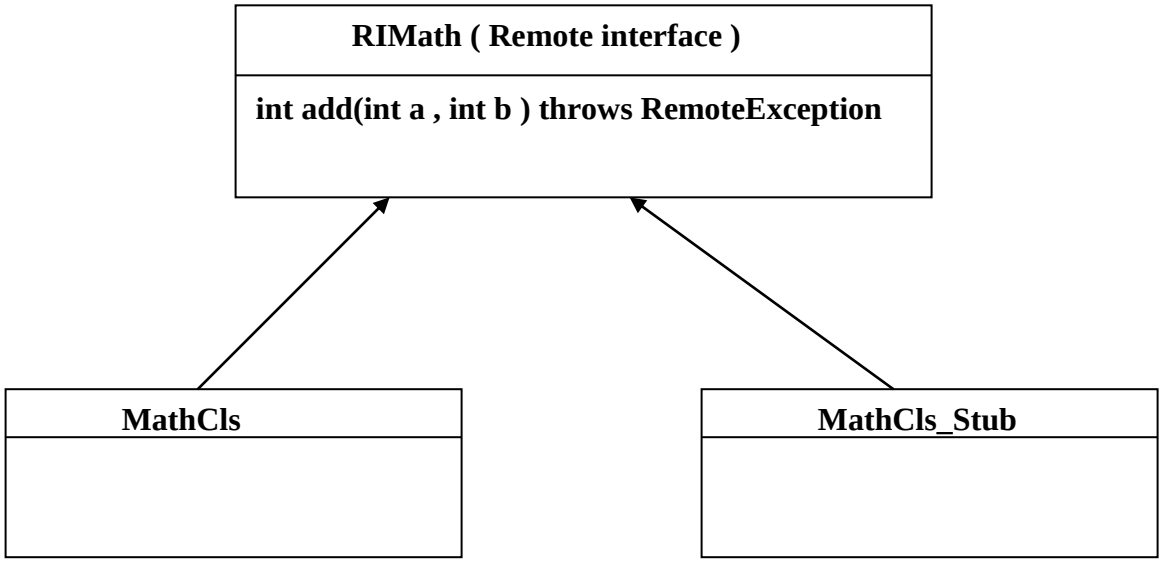
D:\>rmic MathCls

* It creates to use the remote object we must generate the code (**stub , skeleton**) using **rmic** for this we can use the following command.

D:\>rmic –keepgenerated MathCls

* If we don't give the **–keepgenerated** then it sees only .class files that's why we have to give **–keepgenerated** , it is given it creates .java files.

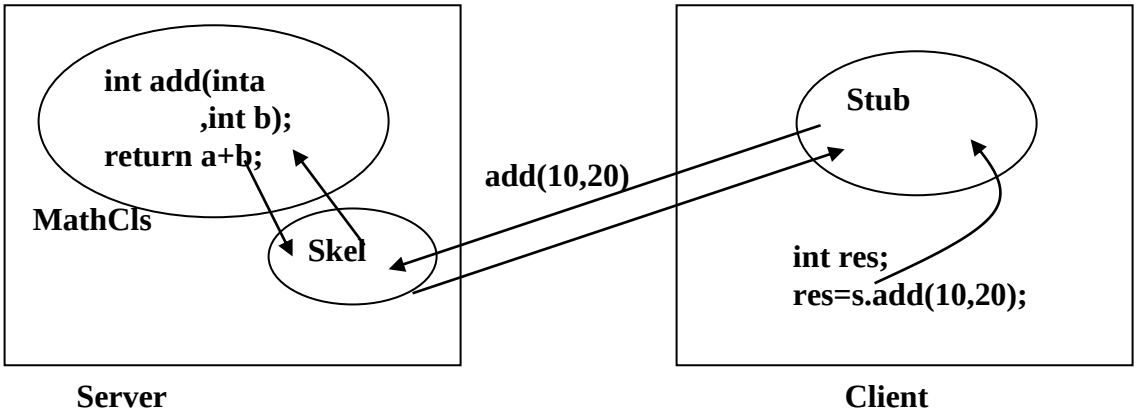
* The stub class that is generate by the **RMI compiler** provides the implementation of the remote interface provided by the developer.



RIMath & MathCls has to be provided by the The developer

Stub class will be generated by the **rmi** compiler.

* Here we mentioned **MathCls_Stub** but it is not working that's we have to write the **RIMath** interface.



* In **RMI** The application in which the remote object is available is called as **server &** the application which invokes the remote method is called as a **Client**.
* In RMI the **Stub** object will be used by the **Client** & the **Skelton** object will be used on the server, the **Stub** object is responsible for sending the request to the server by passing the information like the method the parameters of the method.

The **Skelton** is responsible for taking the information provided as part of request & is responsible for calling the method on the remote object.

After the method is executed the **Skelton** take the result & sends the result to the **Stub** & the **Stub** gives the result to the **Client**.

After that compile

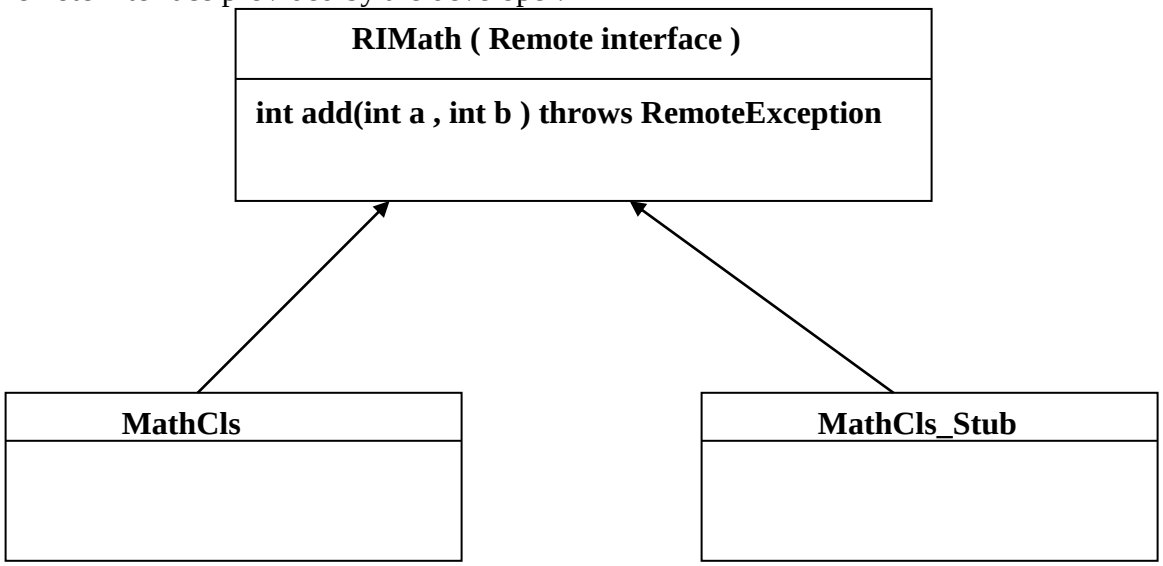
D:\>rmic MathCls

* It creates to use the remote object we must generate the code (**stub , skeleton**) using **rmic** for this we can use the following command.

D:\>rmic –keepgenerated MathCls

* If we don't give the **–keepgenerated** then it sees only .class files that's why we have to give **–keepgenerated** , it is given it creates .java files.

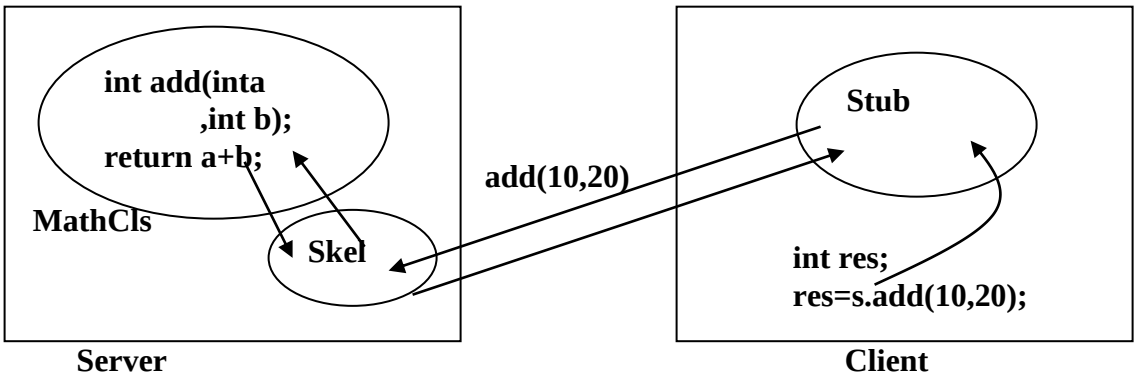
* The stub class that is generate by the **RMI compiler** provides the implementation of the remote interface provided by the developer.



RIMath & MathCls has to be provided by the The developer

Stub class will be generated by the **rmi** compiler.

* Here we mention **MathCls_Stub** but it is not working that's we have to write the **RIMath** interface.



* In **RMI** The application in which the remote object is available is called as **server &** the application which invokes the remote method is called as a **Client**.

* In RMI the **Stub** object will be used by the **Client** & the **Skelton** object will be used on the server, the **Stub** object is responsible for sending the request to the server by passing the information like the method the parameters of the method.

The **Skelton** is responsible for taking the information provided as part of request & is responsible for calling the method on the remote object.

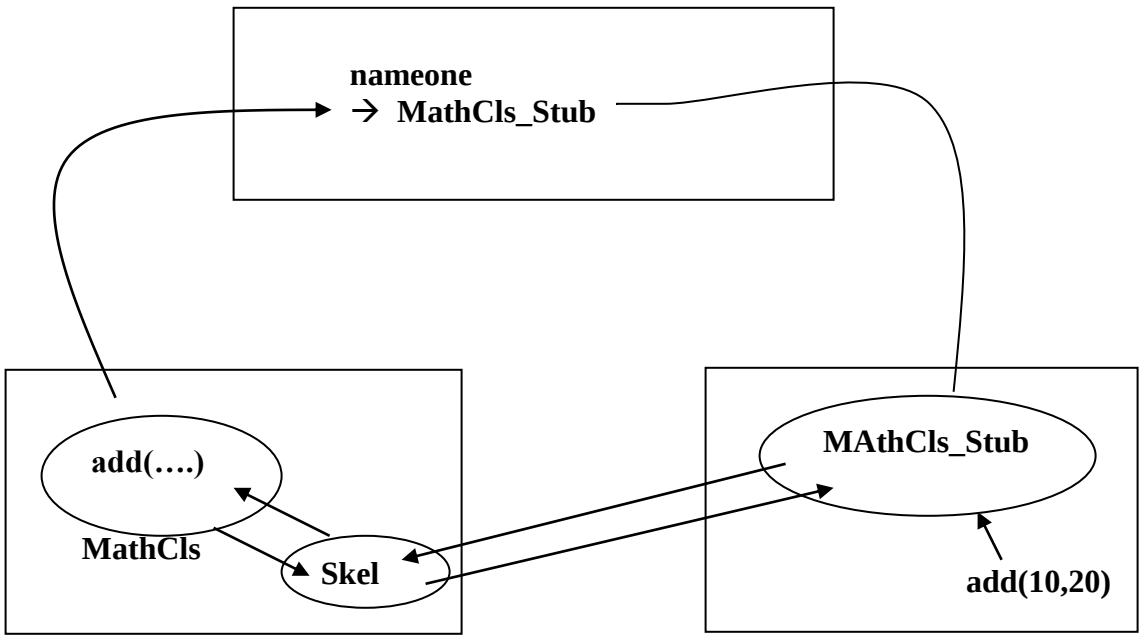
After the method is executed the **Skelton** take the result & sends the result to the **Stub** & the **Stub** gives the result to the **Client**.

PROCEDURE FOR DEVELOPING SERVER :

* RMI registry is a directory service it can be used to store the information about remote objects.

Server.java

```
import java.rmi.*;
import java.io.*;
public class server
{
 public static void main(String args[]) throws Exception
 {
  System.out.println("Create MathCls Object.....");
  RIMath ro=new MathCls();
  System.out.println("Storing MathCls Object Info in RMI registry..");
  Naming.bind("rmi://localhost/nameone",ro);
  System.out.println("Sucess");
 }
}
```



Server **Client**

* In the Server running we must set the class
* When we run the server

1. The remote object based on **MathCls** will be created after this when the server executes **Naming.bind()** method it will contact the rmiregistry & provides the information like the name of the object **Stub** class etc…… RMI registry is responsible for holding this information.

PROCEDURE FOR IMPLEMENTING THE CLIENT

Client.java

```
import java.rmi.*;
import java.io.*;
public class client
{
 public static void main(String args[]) throws Exception
 {
  Object o=Naming.lookup("rmi://localhost/nameone");
  System.out.println(o.getClass());
  RIMath remref=(RIMath)o;
  int res;
   res=remref.add(10,20);
 }
}
```

* When the Client application executes **Naming.lookup()** method it contacts the **rmiregistry** & ask the **rmiregistry** for the information about the object whose name s **nameone** .

The rmiregistry provides the information about the object using this information about the object using this information the **lookup()** creates a stub object (This stub object acts like a remote reference for the object in the server).

* When the Client calls **remref.add(10,20)**

Step 1: The Stub packs the parameters.(Marshalls the parameters) & sends the parameters to the server.
Step 2: The skeleton receives the parameter ,unpacks the parameters (unmarshalls the arameters).
Step 3: The skeleton calls the method on the remote object & takes the result.
Step 4: The skeleton marshall the result & sends it to the client.
Step 5: The stub receives the result & unmarshall the result & gives the result to the caller.

Note : The code to take care of communication (establishing the connection , disconnecting, sending the request, getting the response) will be taken care by the code that is part of **Stub & Skeleton** that is a java programmer need not no about network programming to use RMI , RPC , CORBA , DCOM are distributed technology's similar to RMI technology the main difference between these technology is the protocol that is use by the client & the server to communicate , In case of CORBA Stubs & Skeletons that are generated by IDL compilers (IDL to C++ , IDL to small talk , IDL to java) uses internet inter ORB protocol .

* RMI compiler prier to java1.2 generates the Stub & Skeleton using JRMP (java remote method protocol).
* The differences in RMI & CORBA is protocol.
* As part of java 1.2 RMI technology is implemented by using JRMP & IIOP.

Ex : RIOne.java

```
import java.rmi.*;
import java.net.*;
public interface RIOne extends Remote
{
 public int mone(int a, int b) throws RemoteException;
 public String mtwo(String uname) throws RemoteException;
 public void mthr(Socket s) throws RemoteException;
}
```

2.**RIOneCls.java**

```
import java.rmi.*;
import java.net.*;
import java.rmi.server.*;
public class RIOneCls extends UnicastRemoteObject implements RIOne
{
 public RIOneCls() throws RemoteException
 {
  super();
 }
 public int mone(int a, int b) throws RemoteException
 {
  System.out.println("addition Excuted.....");
  return a+b;
 }
 public String mtwo(String uname) throws RemoteException
 {
  System.out.println("String Excuted.....");
  return "Hi Sudhi"+uname;
 }
 public void mthr(Socket s) throws RemoteException
 {
   System.out.println("Socket Excuted.....");
 }
}
```

3.**Server.java**

```
import java.rmi.*;
import java.io.*;
import java.net.*;
public class Server
```

```
{
 public static void main(String args[]) throws Exception
 {
  System.out.println("Create MathCls Object.....");
  RIOne ro=new RIOneCls();
  System.out.println("Storing RIOCls Object Info in RMI registry..");
  Naming.bind("rmi://localhost/nameone",ro);
  System.out.println("Sucess");
 }
}
```

4.**Client.java**

```
import java.rmi.*;
import java.io.*;
import java.net.*;
public class Client
{
 public static void main(String args[]) throws Exception
 {
  Object o=Naming.lookup("rmi://localhost/nameone");
  System.out.println(o.getClass());
  RIOne remref=(RIOne)o;
  System.out.println(remref.mone(10,20));
  System.out.println(remref.mtwo("Sudhi"));
  Socket x=new Socket();
  remref.mthr(null);
 }
}
```

* When we call **mthr** by passing a Socket object an Exception will be thrown indicating Socket Can't be Serialized.
* EJB internally uses RMI so we can't use Socket connection , ResultSet which are not Serializable as parameters (or) as return types in the remote business interface.
* Most of the developers uses EJB technology to develop distributed computing projects in this case the developers need not provide a server application on their own the developers will be using the products like weblogic , webspeher , JBOSS etc as servers.
* Tomcat is a only web container it is not support a EJB, it is not a EJB container.

Ex :
1.RIBike.java

```
import java.rmi.*;
public interface RIBike extends Remote
{
 public void start()throws RemoteException;
 public void stop() throws RemoteException;
}
```

2.RIBikeCls.java

```
import java.rmi.*;
import java.rmi.server.*;
public class RIBikeCls extends UnicastRemoteObject implements RIBike
{
 public RIBikeCls() throws RemoteException
 {
  super();
 }
 public void start() throws RemoteException
 {
  System.out.println("Starting Bike method.....");
 }
 public void stop() throws RemoteException
 {
  System.out.println("Stop the Bike method.....");
 }
}
```

3.**RIBikeFactory.java**

```
import java.rmi.*;
public interface RIBikeFactory extends Remote
{
 public String creatString() throws RemoteException;
 public RIBike createBike() throws RemoteException;
}
```

4.**RIBikeFactoryCls.java**

```
import java.rmi.*;
import java.rmi.server.*;
public class RIBikeFactoryCls extends UnicastRemoteObject

implements RIBike
{
 public RIBikeFactoryCls() throws RemoteException
 {
  super();
 }
 public String createString() throws RemoteException
 {
  System.out.println("Creating String Object.....");
  String var=new String("test String");
  System.out.println("Returning String Object.....");
  return var;
 }
 public RIBike createBike() throws RemoteException
```

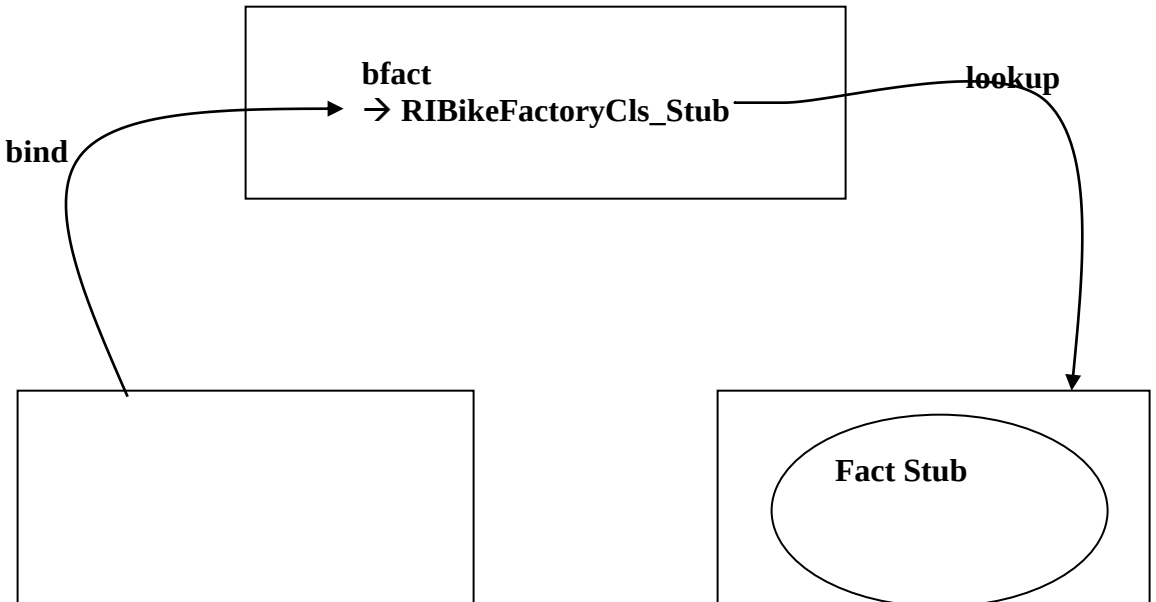
```
 {
  System.out.println("Cerating RIBike Object.....");
  RIBike var = new RIBikeCls();
  System.out.println("returning RIBike Object");
  return var;
 }
}
```

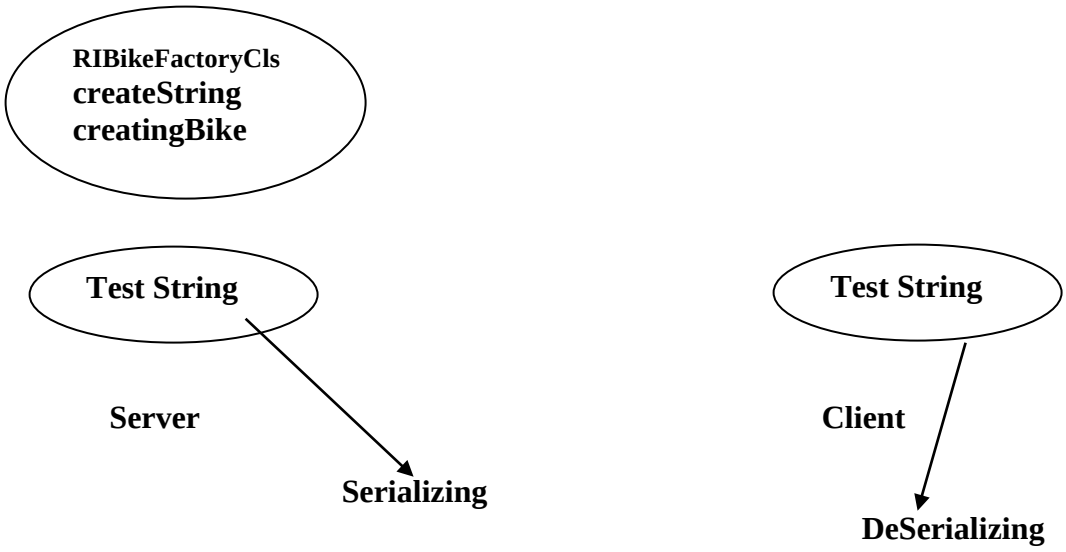
5.**Server.java**

```
import java.rmi.*;
import java.io.*;
public class Server
{
 public static void main(String args[]) throws Exception
 {
  RIBikeFactory ro=new RIBikeFactoryCls();
  Naming.bind("rmi://localhost/bfact",ro);
  System.out.println("Sucess");
 }
}
```

6. **Client.java**

```
import java.rmi.*;
import java.io.*;
public class Client
{
 public static void main(String args[]) throws Exception
 {
  Object o=Naming.lookup("rmi://localhost/bfact");
  RIBikeFactory remref=(RIBikefactory)o;
  Object o1=remref.createBike();
  System.out.println(o1.getClass());
 }
}
```





* When we run server it will create factory object & stores the information about the factory object in the **rmiregistry** with the name **bfact**.
* When the client calls createString() method on the server the string object will be created & it will be returned when it is return the information about the String object will be serialized and sent back in the client it will deserialize the object (in the client the string object will created).
* When createBike() is called in the server RIBikeCls object will be created (Remote Object) when it is returned the information about the stub will be send to the client & in the client the Stub object of RIBikeCls will be created if the client calls the on this stub object the method1 will be executed on the remote object available in the server.

* The Remote Garbage collector is responsible for removing the remote object from the JVM if there are no local references remote references (Stub objects).
* In distributed projects Factory Pattern is used to take care of creating the objects in this pattern we use a Factory object which takes the responsibility of creating the objects that are required.
* In case of EJB technology the Factory is called as home it is the responsible of the container to create the Factory object & registering the Factory object a developer need not write the code like our **Server.java**
* What is the advantage of EJB's?
Ans: We need not code to take care of several issues.
* If we develop a distributed project using RMI technology directly we have to write lot of code this can be avoided by developing the application using EJB technology.

If we use EJB technology the container takes care of generating the code to take care of various issues, the developer need to concentrate only on providing **Business Logic**.

* The methods are available in object class
    1. toString()
    2. hashCode()
    3. equals()

* System.out.println() is internally invokes the method **toString()**

* As part of java.lang.Object class java soft has provided the code for the methods **toString , equals , hashCode** that works fine with local objects these methods are implemented as part of **java.rmi.server.RemoteObject** that works fine with RemoteOject.
* UnicastRemote object contains the code that makes the object available to the other JVM's (Exporting the Objects).

Note: Instead of running rmiregistry as a separate process by using the command rmiregistry. We can run it directly as part of the server application by using the method as shown below:-

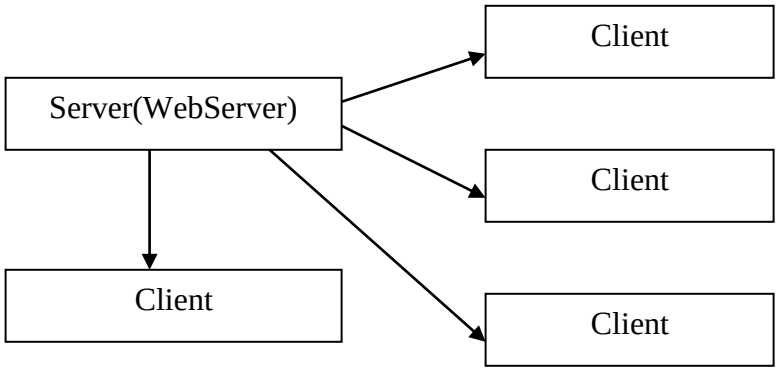
```
import java.rmi.*;
import java.rmi.server.*;
public class Server{
  public static void main(String[] args) throws Exception  {
      java.rmi.registry.LocateRegistry.createRegistry(1099);
      RIBikeFactory ro= new RIBikeFactoryCls();
      Naming.bind("rmi://localhost/xxx",ro);
   }}
```

An Exception will be thrown by methods to indicate that there is some sort of problem.

Note:-

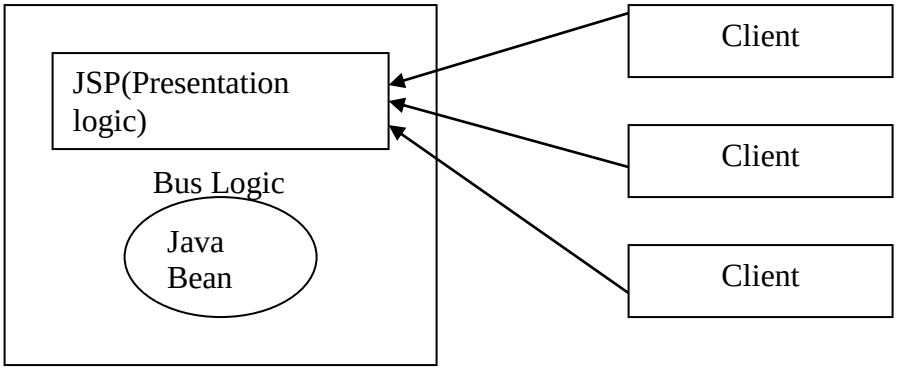
When a client invokes a method the rmi system may fail. To invoke the method due to various reasons.

(1) To indicate the failure the rmi system throws RemoteException. This is why we must declare that every method in the remote interface throws RemoteException.

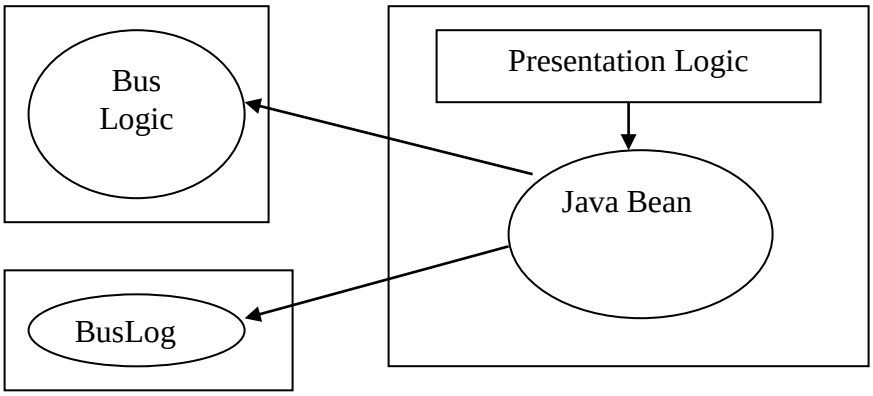


Note:-

To run the clients on multiple machines we have to copy several files (remote interfaces, stub-classes and the other classes on all the machines instead of this we can place all the classes and interfaces on a webserver and download the classes and interfaces from the webserver to the client machines by writing a small application using rmi classloader.



Web Container



Note:-

If we built an application by implementing business logic in javabeans presentation logic as part of jsp, when a request is sent from browser the whole code has to be executed in a single java virtual machine. There will be no problem if there are less number of clients sending the request concurrently. The Performance will be effected if more number of clients sends the request concurrently.

To improve the scalability of the application we can separate business logic run it as part of a different machine as shown in the diagram above.

An application that can withstand(serve) more amount of load (more amount of clients) is called as scalable application.

{ Running the same business logic on different machines concurrently to reduce the load is called as clustering.

Note:-Enterprise Javabeans can be used to implement the business logic . This is why enterprise javabeans are also called as business components. If we develop a distributed application directly using rmi technology as part of our code, we must provide code for

(1) Creating Object.
(2) Removing (Unreferencing) the objects.
(3) Activating objects.
(4) Passivating objects.
(5) Take care of security.
(6) Take care of transactions.
……..
(7) Business logic.

But if we use EJB technology we have to provide the code for business logic the remaining code will be taken care by the code of the container & the code that is generated by tools provided as part of container

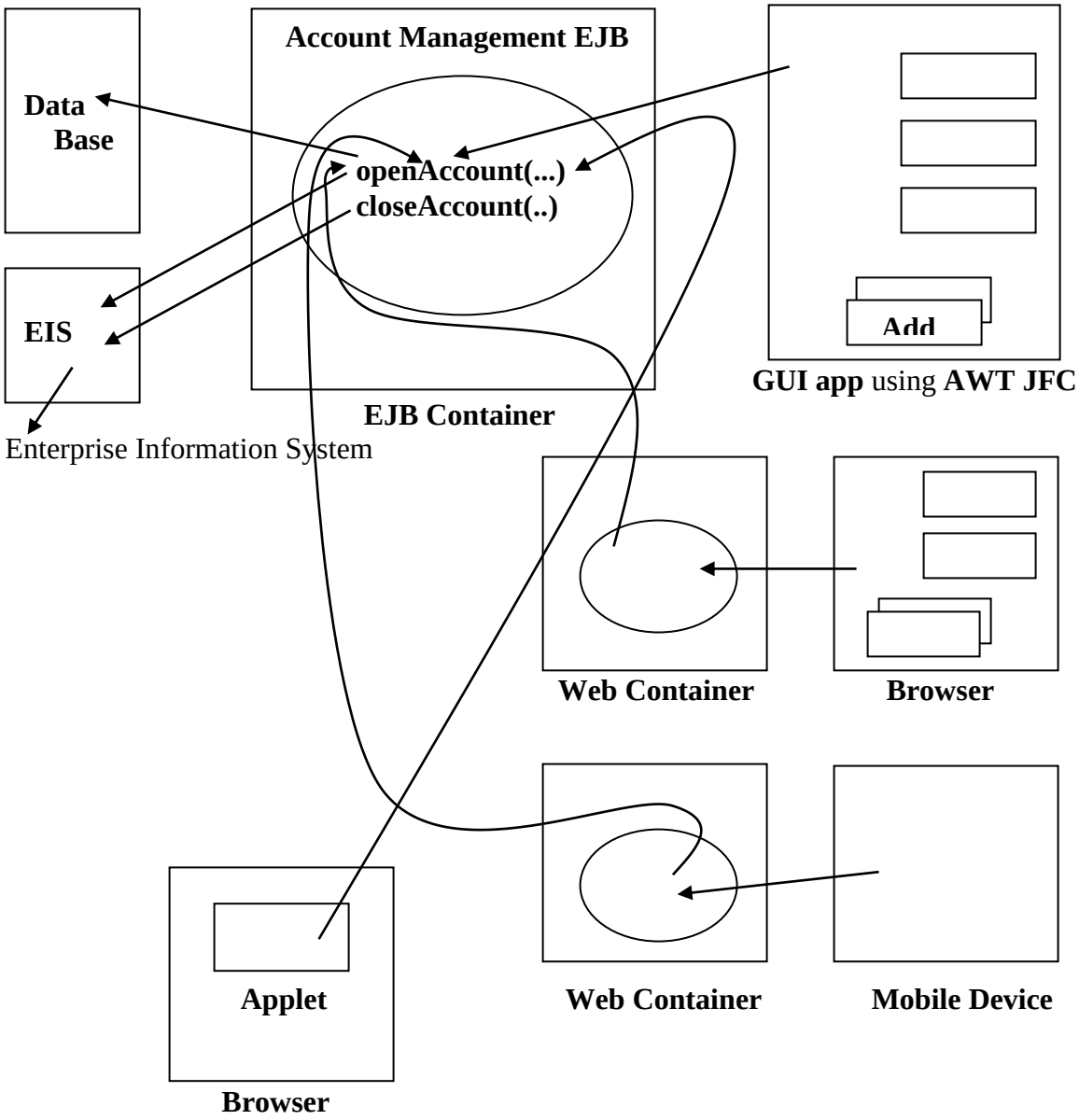
EJB

* **What is the difference between webserver & application server?**

* An application server is product that provides various facilities like **security management , transaction management , component management** etc & simplifies the development of the business logic but to day most of the product vendors are referring to their web servers as application servers by adding some additional facilities to their products.

* The EJB container like weblogic , webspeher , Jboss provides these facilities so we can call these products as application servers Microsoft (MTS) server also provides a set of facilities that simplifies the applications so this product can also called as application server.

A Typical EJB Architecture



* While implementing the EJB the developer need not bauther too much about the user interface the developer need to concentrate about the business logic & provide the code according to the Business Requirement.
* Once The Business logic is implemented in EJB the Business logic can be accessed from various types of clients as shown in the diagram.
* **Procedure for installing jbuilder & Configure the application**

1. install the JBUILDER software
2. Configure the Server

In menu select **Tools** menu

→Select **Configure Servers** option

↳ It will open **Configure Server** pop menu in that we select our Server name
Ex: my server is **weblogic Server8.x**

↳ Select **Enable** option it will enable the server

↳ After In that only We select **Custom** menu option

↳ Give **BEA Home Directory Name**
Ex : **c:/bea**

↳ Give **JDK Installation Directory Name**
Ex : **C:/j2sdk1.4.1_01**

↳ Give **Domain directory Name**
Ex : **C:/bea/user_projects/sudarshan** it is my domain
plz give what ever your domain names

↳ Give **Password** in what ever we give in weblogic server it is
Compulsory to give same password.

↳ Finally De Select the all **check** options then click **ok** button

Our server is Configured……

3 We do a Simple Project…

Step 1: in jbuilder window first click

File menu

↳Select **new** option

↳ It opens **object gallery** in that select **project option→project** click **ok** button

↳ In Project Wizard select **Project Name & select Path** where we to create
then click **Finish** button

After that select **File menu** ⟶ **new** option

↳ Select **EJB** option ⟶ **EJB Module** then click **ok** button

↳ Select Server which server we have to use Ex **Weblogic server8.x** click **ok** button

↳ Select EJB Module Origin **Create empty EJB Module** then click **next** button

↳ Select EJB Module **Name** then click **Finish** button

It will open new window In that select **Create EJB** menu in that **Session Bean** option

↳ Select Session Bean Properties give **Bean Name** Ex: xyz

↳ It will display **xyz** bean in that we click **right click** it will display new pop menu in that we select **Add** option in that we select **method option**

↳ finally we give method name

* If we develop an Enterprise java bean manually according to the EJB specification we must provide (ie developer)
    1. Remote Business Interface
    2. Remote Home Interface
    3. Bean class

* The Remote business interface must be a sub interface of **javax.EJB.EJBObject** (this interface is sub interface of **java.rmi.remote** interface.)

As part of Remote interface we must provide a set of business methods for this we must follow the rmi rules in developing the methods.

```
public interface TestEB extends EJBObject
{
  public String welcome(String uname)throws RemoteException;
  public int aff(int pone,int ptwo);
}
```

* The Remote home interface must be a sub interface of **EJBHome** (This is a sub interface of RemoteInterface) as part of the Remote Interface we must provide **create** method & this method should throw RemoteException ,CreateException .
* The return type of create method must be RemoteBusiness Interface.

```
public interface TestEBHome extends EJBHome {
  Public TestEB create() throws CreateException,RemoteException;
 }
```

* It is the responsibility of the container to generate the classes that implements RemoteBusinessInterfece & RemoteHomeInterface in case of some containers (Weblogic) we can physically see this classes by running a tool (EJBC).
* As part of the Bean class we must provide the business logic according to our requirements ,there are basically 3 types beans
    1. Message Bean
    2. Entity Bean
    3. Session Bean
* The Bean class must implements session Bean interface if we are developing session Bean.

```
public class TestEBBean implements SessionBean  {
  SessionContext      sessionContext;
public void ejbRemove(){}
public void ejbActive(){}
public void ejbPassivate(){}
```

```
    public void setSessionContext(SessionContext  sessionContext){
      this.sessionContext=sessionContext ;
    }
    public String welcome(String uname){
      return "Welcome……."+uname;
    }
     public int add(int pone , int ptwo){
       return pone+ptwo;
     }
   }
```

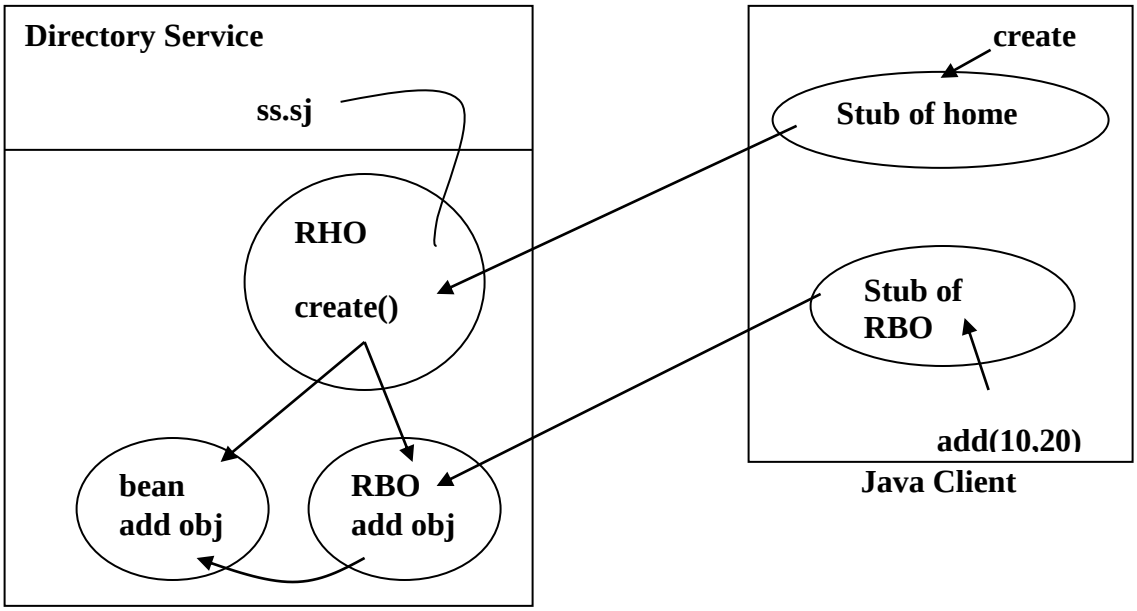
DEVELOPER IS RESPONSIBLE FOR PROVIDING :

1. Remote Business Interface (**RBI**) .
2. Remote Home Interface (**RHI**).
3. Bean Class

EJB container is responsible for generating :

1. Remote Business Class (Class implementing RBI)
2. Remote Home Class (Class implementing RHI)

Note : In case of weblogic we can see these classes physically , in case of JBOSS these classes will be generated when we deploy the EJB.



* When we deploy an enterprise java bean the EJB container creates **Remote Home Object** (**RHO**) & registers the Information about the **Remote Home Object** (**RHO**) in the directory Service.

PROCEDURE FOR DEVELOPING THE CLIENT :

Step 1: Gather the following information of the code

1. JNDI name - - - - - - - ► **ss.sj**
2. Remote Home Interface - - - - - - - ► **first.test**

Step 2: Provide the code following the steps given below.

1. get the reference of the **Initial Context Object**

```
Hashtable h = new Hashtable();
h.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WLInitialContextFactory");
h.put(Context.PROVIDER_URL, "t3://localhost:7001");
h.put(Context.SECURITY_PRINCIPAL, "sudarshan");//user
h.put(Context.SECURITY_CREDENTIALS, "soujanya");
Context ic = new InitialContext(h);
System.out.println(" initial context = "+ ic);
```

2. using **ic.lookup** get the reference of Remote Home Object

```
Object o = ic.lookup("ss.sj");
TestEBHome homeref = (TestEBHome)o;
```

Step 3: get the reference of the bean object by invoking the create method.

```
TestEB beanref = homeref.create();
```

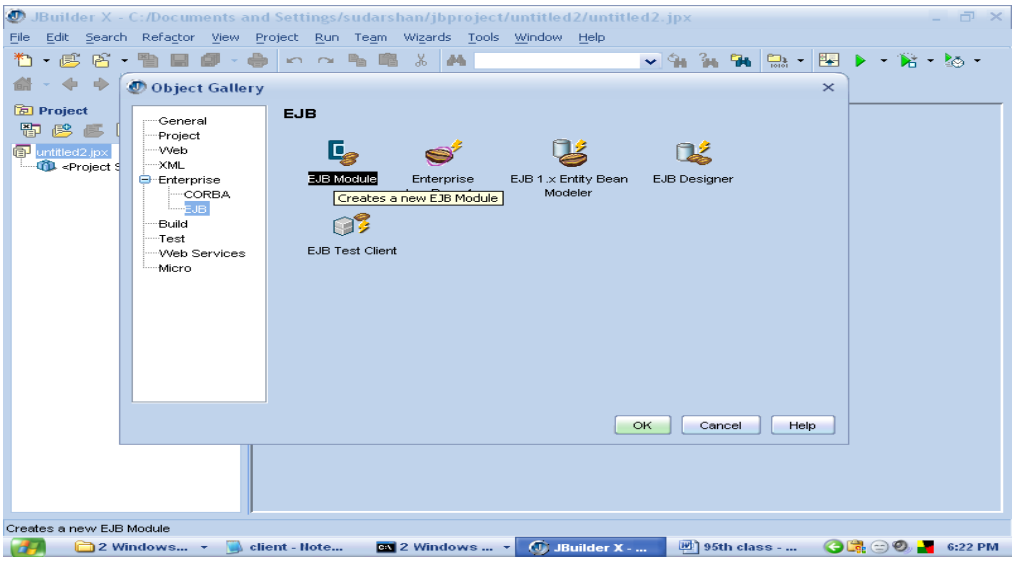
Step 4: invoke the business methods according to the request using the reference of bean object.

```
String res = beanref.welcome("Sudhi");
```

Step 5: Call **beanref.remove()** method to tell the container that the client has finished using bean.

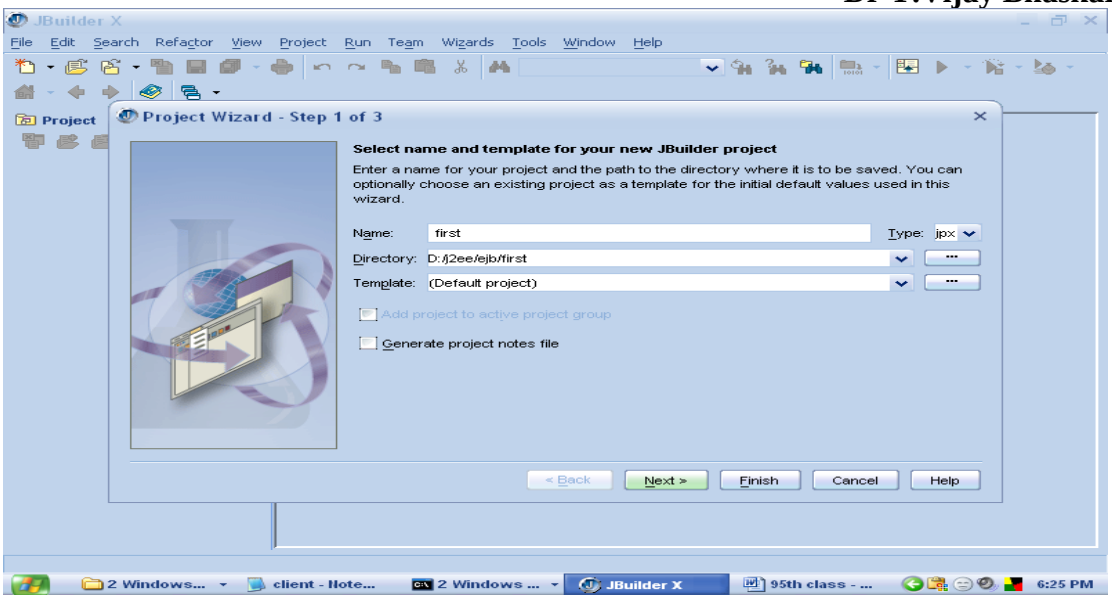
* **procedure for developing in Jbuilder :**

Step 1: open jbuilder select in menu bar **new** it will display new pop up menu

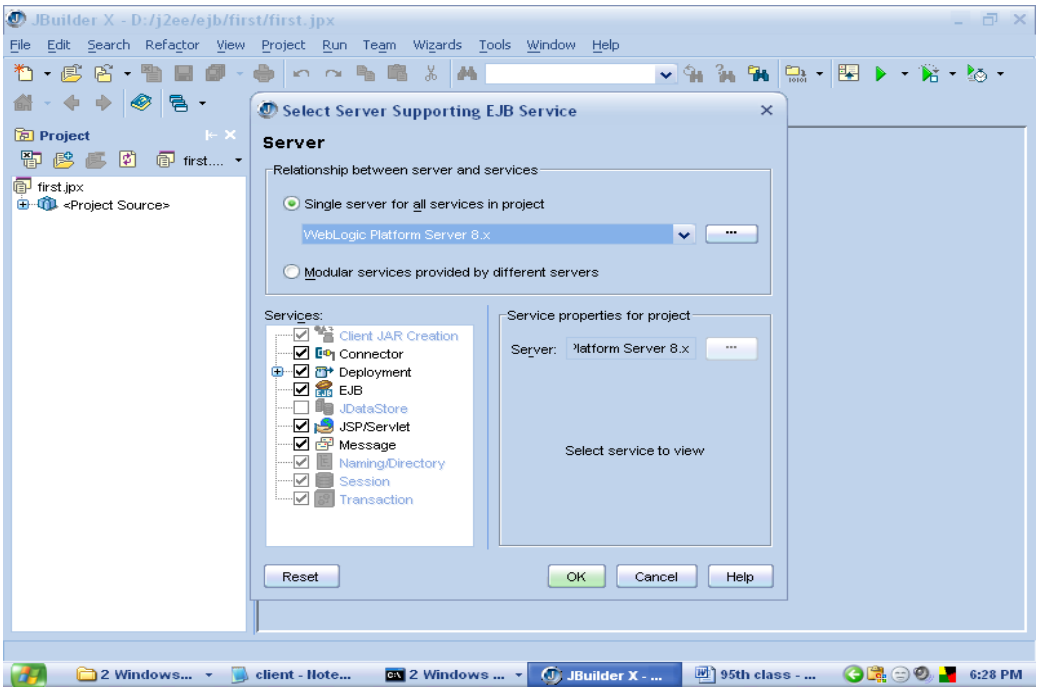


Select **EJB Module** after that click **ok** button

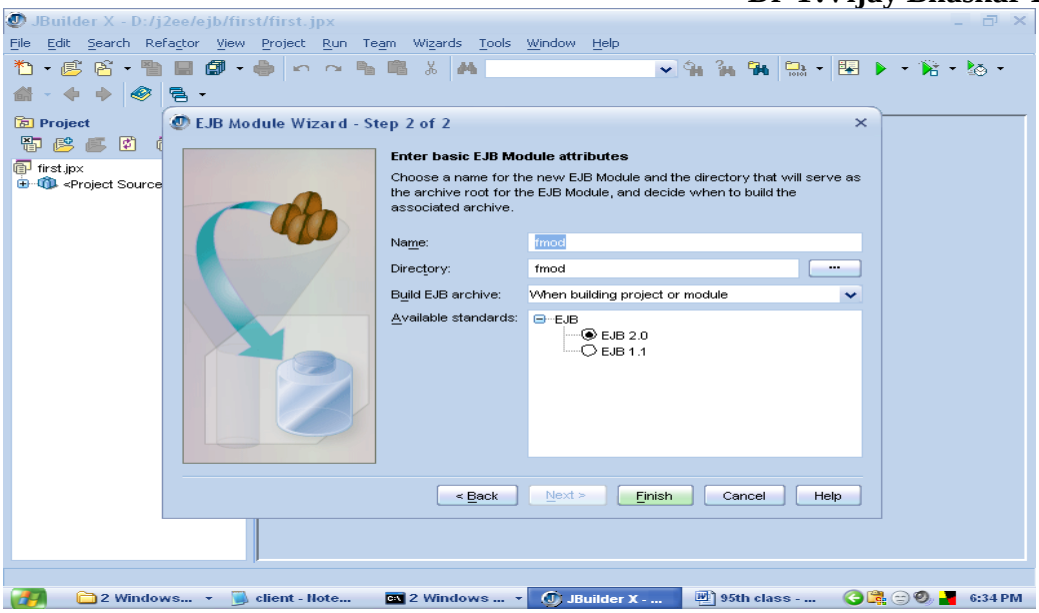
After that it displays



Give the project name Ex: my project name is **first**
Give Directory where we can place Ex: I am placed in **d:\j2ee\ejb>**
Finally click on **finish** button after that It displays another pop up menu

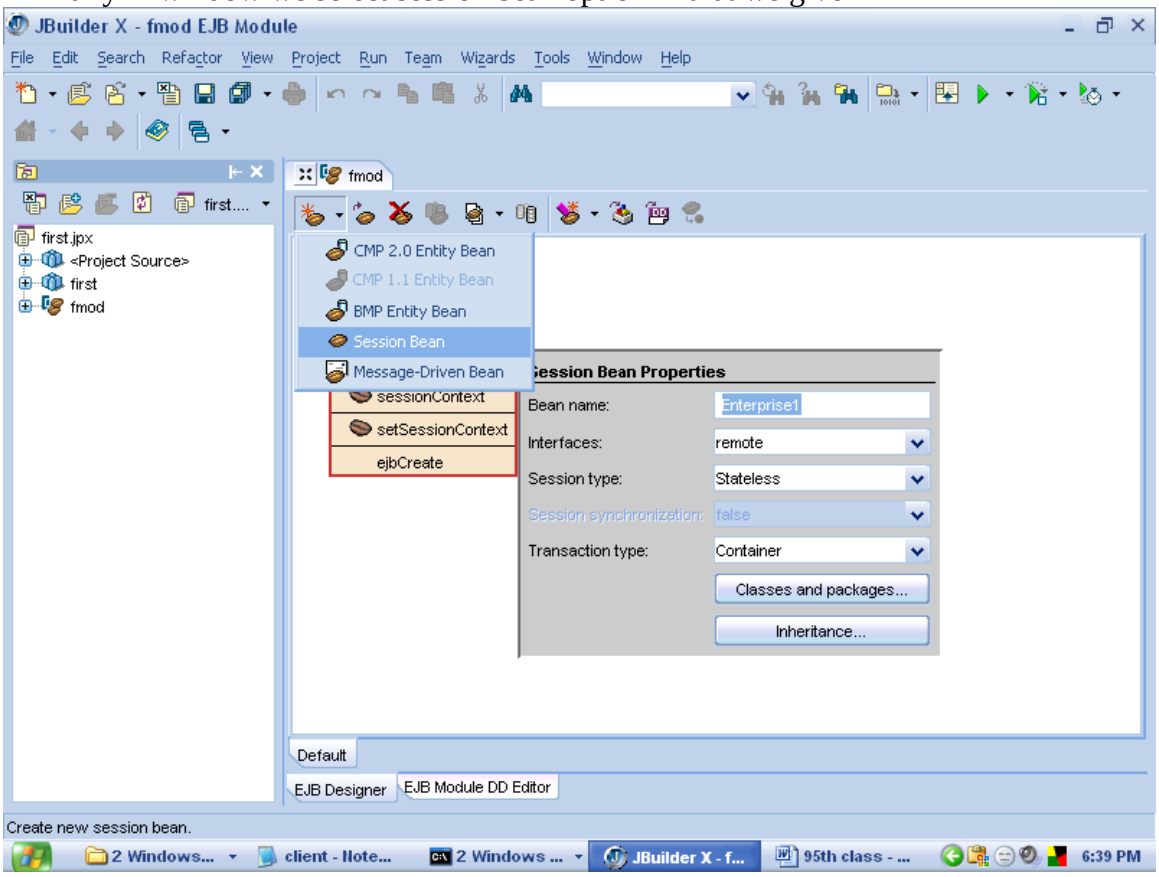


In that select our current web container Ex: my container is **weblogic 8.1** after that click **ok** button after that it displays another pop up menu it displays EJB module click on **next** button it displays another pop up menu



Give module name Ex: **fmod** click on **finish** button.

* Finally in window we select session bean option in that we give



In that we specify **Bean Name** Ex: test in that we select methods it is project require ment. Ex: I am enter two methods ie add() , welcome()

Step 2: after that what ever code add **testBean.java**

```
package first;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import javax.ejb.CreateException;

public class testBean
   implements SessionBean {
  SessionContext sessionContext;
  public void ejbCreate() throws CreateException {
  System.out.println("ejb Creative()");
  }

  public void ejbRemove() {
  }

  public void ejbActivate() {
  }

  public void ejbPassivate() {
  }

  public void setSessionContext(SessionContext sessionContext) {
    this.sessionContext = sessionContext;
    System.out.println("ejb SessionContext()");
  }

  public int add(int pone, int ptwo) {
    System.out.println("add() Executed.....");
    return pone+ptwo;
  }

  public String welcome(String uname) {
    System.out.println("welcome() executed.....");
    return "Welcome....."+uname;
  }
}
```

Step 3: after that make the project & deploye the project.

Ex : Client.java

```
import first.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.rmi.*;
import java.util.*;
```

```
public class client{
      public  static void main(String args[]){
       try{
          Hashtable h = new Hashtable();
          h.put(Context.INITIAL_CONTEXT_FACTORY,
             "weblogic.jndi.WLInitialContextFactory");
          h.put(Context.PROVIDER_URL, "t3://localhost:7001");
          h.put(Context.SECURITY_PRINCIPAL, "sudarshan");
          h.put(Context.SECURITY_CREDENTIALS, "soujanya");
          Context ic = new InitialContext(h);
             testHome homeref=(testHome)ic.lookup("ss.sj");
             test beanref=homeref.create();
             System.out.println(beanref.add(10,20));
             System.out.println(beanref.welcome("Sudarshan"));
             beanref.remove();
          }
          catch(Exception e)
            {}
          }
}
```

Out put is : client side is :

```
30
Welcome.....Sudarshan
```

Serverside is:

```
ejb SessionContext()
ejb Creative()
add() Executed.....
ejb SessionContext()
ejb Creative()
welcome() executed.....
```

* First We have set the Path after that we have to do the other operations

C:\bea\user_projects\sudarshan\>setenv

After that we have to set the path

d:j2ee\ejb>set CLASSPATH=%CLASSPATH%;.

* After that we have to go our current directory using same dos prompt.

* There are 3 types of beans
Ie

1. **Session Bean**
    a. **Statefull Session Bean**
    b. **Stateless Session Bean**

2. **Entity Bean**
   a. **Container managed persistent Entity Bean**
   b. **Bean Managed persistent Entity Bean**

3. **Message Driven Bean** (**Added as part of EJB 2.0)**.

* In case of EJB a Client that is included in calling the Business methods on the EJB is considered to be in conversation with the Bean.
* A stateless Session Bean is not responsible for remembering the conversational state of the client (ie the Bean will not remember what is done by the client earlier).
* Statefull Session Bean is responsible for remembering the conversational state of the client.

Cache : Is the area where bean objects are stored.

* In case of statefull session bean when the client calls the create method the container creates a bean object & then it will call setSessionContext & ejbCreate() method.

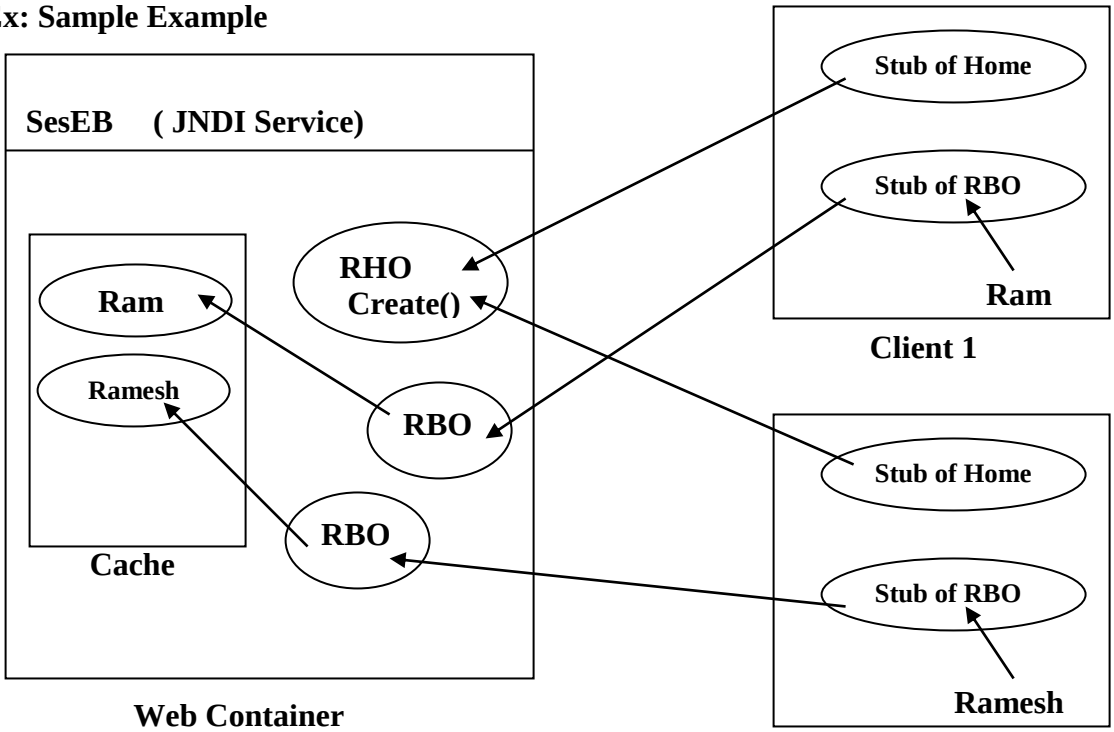
If 100 clients calls homeref.create() the container will create 100 bean objects (ie there will be a bean object for every client) when the business methods invoked by the client the container executes the business methods that was created earlier on behalf of the client.

* In case of statefull session beans when the client calls remove method, the ejb container executed ejbRemove() & it removes the bean object.

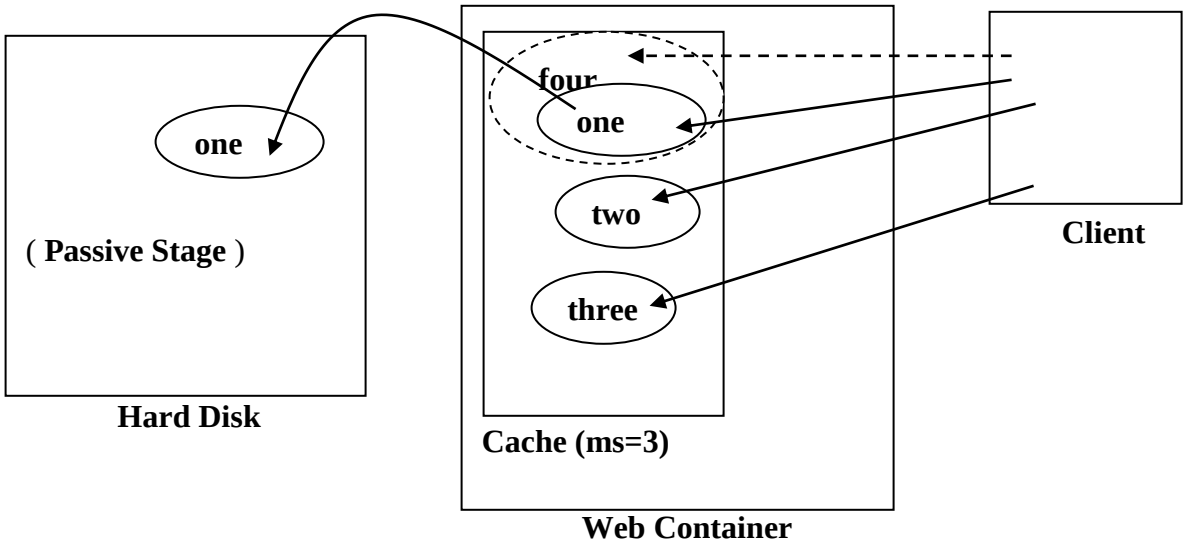
* Most of the containers allows us to time out value.

* Most of the containers removes the bean object if the object is not used for time out amount of time (before removing the object) the container calls ejbRemove() methods.

Ex: Sample Example



* For statefull session bean we can set the maximum beans in cache in the deployment descriptor.



* In case of statefull session beans when ever there is no space the container picks up one of the object & it will call **ejbPassivate()** method on that object after this the server serializes the object to the disk & removes the object from the **cache** this is called as passivating an object.
* If a Business method is called on an object which is passive then the container DeSerializes the object from the disk & it will call **ejbActivate()** method this process is called as Activating the bean.

* We can provide the code to acquire the resources in the **ejbCreate()** & provide the code to de allocate the resources in **ejbRemove()** method.
* If we need to use a socket connection as part of the code in the business methods we can provide the code to create a socket object in the **ejbCreate() ,** use this socket object in a business method, close the socket connection in **ejbRemove()**.
* As socket can't be serialized we must declare the instance variable referring to the socket object as **transient**.
* If we declare socket as **transient** while passivating the information about the socket will not be written to the disk, so while activating there will be socket object , the Business methods using socket object will fail after activation.
* To solve the above problem we can provide the code for closing the socket connection in **ejbPassivate()** method & opening the socket connection in **ejbActivate()** method that is we have to provide the code take care of transient objects as part of **ejbPassivate() & ejbActivate()**.

STATE LESS SESSION :

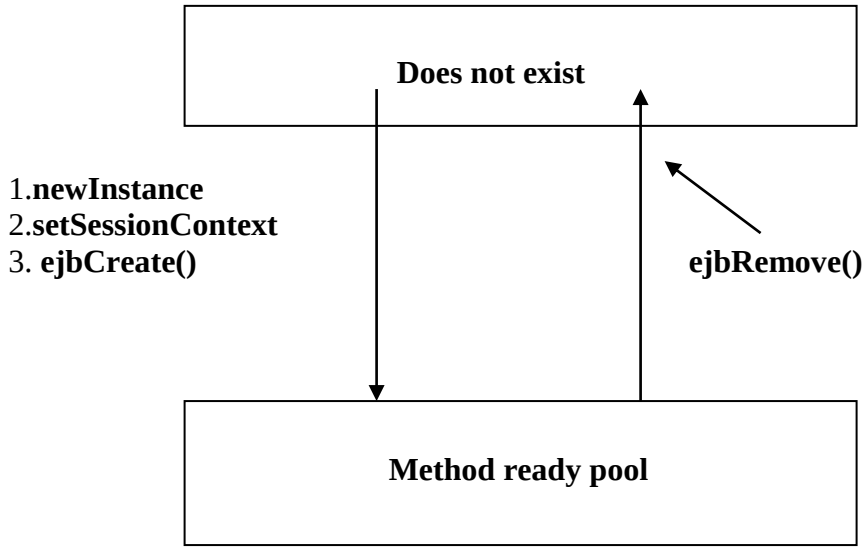
* In case of stateless session beans if there is no space the container will remove the object (that is it will not passivate the object).

The container will not call the ejbActive() , ejbPassivate() in case of state less session bean.

* According to the ejb specification the ejb container is responsible for managing the **pool** it is the responsibility for the container to add the bean object to the pool & remove the bean object from the pool at any point of time ie the client can't control the creation of bean objects & removal of bean objects.

* According to the ejb specification the container must call setSessionConext & ejbCreate() after creating the bean object. Before removing the bean object the conainer must call ejbRemove().

* In case of weblogic server as part of the Deployment descriptor we can specify initial beans in free pool , if we can specify initial beans in free pool , if we specify this to (7) seven bean objects will be created when we deploy the ejb , In case of weblogic we can specify max beans in free pool if this value as specified in 10 the weblogic server ensures that there will be a maximum of 10 bean objects in the pool.



* For what type of Beans the container manages the pools ?
Ans: Except statefull session beans the container manages the pools.

* Statefull session beans places more burden on the server (ie performance wise we can get better performance using stateless session bean).

In case statefull session beans the container has to maintain more no.of bean object if the objects are managed in the cache more amount of memory is required if some of the objects are placecd on the disk the container has to spend more amount of time in activation & passivation of the objects.

StateFull Session Beans	StateLess Session Beans
1. Remembering the state of client. 2. Client can associate with the bean object. 3. When ever client calls create() the bean object will be created & removed.	1. Not responsible for remembering state of client. 2. Client can't associate with the bean object. 3. It is an responsible of a container do that when do create & remove beans.

* If we have a form as shown below used for user registration we can implement the business logic using statefull session bean (or) by using state less session beans if we provide the business methods like setName(), setPwd(), setMName(), setFName(), setAddr(), storeData() we must declare bean as statefull session bean.

```
public class SFSB implements SessionBean
{
  private String uname,pwd,mname,fname,addr;
  public void setName(String n)
  {
    Uname=n;
  }
 public void setPwd(String n)
 {
   pwd=n;
}
 public void setMname(String n)
{
   Mname=n;
}
 public void setFname(String n)
 {
   fname=n;
}
 public void setAddr(String n)
```

```
 {
  addr=n;
}
public void storeData()
 {
  Code store data available in the instance variable in DB.
 }
}
```

* We can implement the business logic as part of the SLSB by providing the code as shown below.

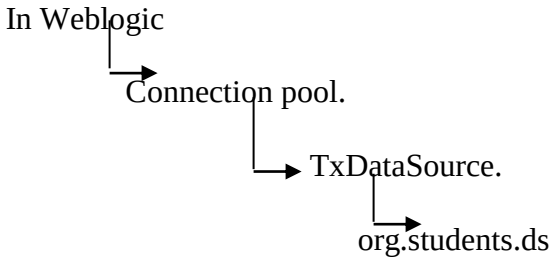
```
public class SLSB implements SessionBean
{
 public void storeData(String uname,String pwd,String mname,String fname,String addr)
  {
    Code to store data passed as params in DB.
 }
}
```

A Java Developer involved in the development of EJB project can use one of the following options to access the data :-

(1) By Providing JDBC code directly in Session Beans or Message Driven Beans.
(2) By using Entity beans.
(3) By using Java Data Objects (JDO).
(4) By using object-relation mapping technologies like hibernate,toplink etc.

The Entity Beans are heavy weight and less in performance.

sql> create table baccount(custId number primary key,cname varchar(15), caddr varchar(15), Balance number(10,2), statuc number(1));




```
import java.sql.*;
import javax.sql.*;
import javax.naming.*;
public class AccountManagerBean implements SessionBean
{
    SessionContext sessioonContext;
    DataSource ds=null;
    public void ejbCreate() throws Create Exception
    {
```

```
                try
                 {
                    System.out.println("Trying to get ds");
                    Context ic=new IntialContext();
                    ds=(DataSource)ic.lookup("org.students.ds");
                    System.out.println("got ds");
                  }catch(Exception e)
                  {   }
            }
            public void ejbRemove()
            {
                 ds=null;
             }
          }
```

We can get the connection from the connection pool is the purpose of datasource.

```
public void ejbActivate()
{
 }
 public void ejbPassivate()
 {
  }
  public setSessionContext (SessionContext sessionContext)
  {
    this.setSessionContext =sessionContext;
  }
  public void openAccount(String cid,String cname,String caddr, Striong abal);
  {
     try
     {
        Connection con=ds.getConnection();
        Statement stmt=con.createStatement();
         String sqlstmt="insert into baccount values' " cid+"',' "+cname+" ' ,' " +caddr+" ',' "
+abal+" ' ";
         System.out.println("Executing sql statement");
         stmt.executeUpdate(sqlstmt);
         con.close();
      }catch(Exception e)
      {  }
  }
  public void closeAccount(String cid)
  {
    try
    {
       Connection con= ds.getConnection();
       Statement stmt=con.CreateStatement();
       String sqlstmt="Update baccount set status='0' where custId= ' " +cid+"' " ;
```

```
      System.out.println("Executing sqlstmt…");
      stmt.executeUpdate(sqlstmt);
      con.close();
    }catch(Exception e)
    {  }
}
```

Cli1.java

```
import bapproj.*;
public class Cli1
{
    Hashtable h=new Hashtable();
    h.put(Context.INITIAL_CONTEXT_FACTORY,
          "weblogic.jndi.WLInitialContextFactory");

   h.put(Context.PROVIDER_URL, "t3://localhost:7001");
   h.put(Context.SECURITY_PRINCIPAL, "chakri");//user
   h.put(Context.SECURITY_CREDENTIALS, "shashikala"); //Password.
   Context ic=new InitialContext(h);
   Object o=ic.lookup("AccountManager");
   AccountManagerHome hr=(AccountManagerHome)o;
   AccountManager br=hr.create();
   br.openAccount("1","xxx","aaa","123","1");
   System.out.println("openAccount…");
   System.in.read();
   System.in.read();
   br.closeAccount("1");
   br.remove();
  }catch(Exception e)
  {   }
}
```

In the Tomcat

→ Webapps.

oaform.html

```
<html>
<head>
<title>Open New Account</title>
</head>
<body>
 <form action="oasrv">
    Customer Id
     <input type="text" name="custId"/><br>
    CustomerName
```

```
        <input type="text" name="custName"/><br>
      CustomerAddress
        <input type="text" name="custAddr"/><br>
      Opening Balance
        <input type="text" name="OpenBal"/><br>
        <input type="submit" value="openAcccount"/><br>
  </form>
  </body>
  </html>
```

OaSrv.java

```
    import java.util.*;
    import javax.servlet.*;
    import java.io.*;
    import javax.naming.*;
    import javax.servlet.http.*;

   import bapproj.*;
   public class OaSrv extends HttpServlet
   {
     public void service(HttpServletRequest request,HttpServletResponse response) throws
ServletException,IOException
     {
        PrintWriter out=response.getWriter();
        try
        {
          String vcid,vcname,vcaddr,vobal;
          vcid     =  request.getParameter("custId");
          vcname =  request.getParameter("custName");
          vcaddr  =  request.getParameter("custAddr");
          vobal    =   request.getParameter("custId");
          Hashtable h=new Hashtable();
          h.put(Context.INITIAL_CONTEXT_FACTORY,
                "weblogic.jndi.WLInitialContextFactory");

          h.put(Context.PROVIDER_URL, "t3://localhost:7001");
          h.put(Context.SECURITY_PRINCIPAL, "chakri");//user
          h.put(Context.SECURITY_CREDENTIALS,"shashikala"); //Password.
          Context ic=new InitialContext(h);
          Object o=ic.lookup("AccountManager");
          AccountManagerHome hr=(AccountManagerHome)o;
          AccountManager br=hr.create();
          br.openAccount(vcid,vcname,vcaddr,vobal);
          out.println("Account Created for U..");
        }catch(Exception e)
         {
              out.println(e);
```

```
    }
  }
```

Web.xml

```
<servlet-name>OaSrv</servlet-name>
 <url-pattern>OaSrv</url-pattern>
 copy bapproj into the WEB-Inf classes dir.
 Copy weblogic .jar into tomcat/common/lib
http://localhost:8080/bapp/oaform.html
```

Note:-

In order to run the servlet on tomcat which invokes the business logic on EJB's running inside weblogic. We must copy weblogic.jar under tomcat/common/lib and remote business interface, remote home interface servletclass under WEB-INF/classes directory.

* Session context can be used by the session bean to interact with its environment.

Ex: We can use **sessionContext.getCallerPrincipal()** to know the user who has invoked the method.

* As part of the ejb's we can use the methods like **sessionContext.getCallerPrincipal()** , **sessionContext.callerInRole()**.

* To take care of security in EJB's we have two options

1. We can provide our own code as part of the business methods to take care of security.
2. We can place the perminissions regarding who can invoke a particular method in the deployment descriptor.

It is easy to change the deployment descriptor when we need to make a change in the permission this is why it is better to use the second option to take care of security.

How To Manage The Transactions :

* con.commit() ,con.rollback() shouldn't in EJB.

* In java application we can control the data base transaction by providing the code shown below.

```
con.setAutoCommit(false);
try{
perform db ops using con.
con.commit();
}catch(Exception e)
{
con.rollback();
}
```

* As part of EJB we should not use connection.commit() , con.rollback()

As part of EJB's we must use transaction API (**JTA**) to control the transaction.

* **javax.transaction.UserTransaction**

Is an interface that is part of **JTA.**

* As part of **ejb's** we can use one of the following two options to control the transaction.

1. We can provide code to manage transaction using JTA (**this is called as bean managed transaction**).

2. We can setup transaction attributes in the deployment descriptor so that container can take care of transactions(**this is called as container managed transaction**).

userTransaction ut=sessionContext.getUserTransaction();

* In order to control the transaction we must first get the reference of user transaction object for this we can use **sessionContext.getUserTransaction()** .

```
public void someMethod()
{
  UserTransaction ut=null;
 try {
 ut=sessionContext.getUserTransaction();
ut.begin();
//code to perform db operations
ut.commit();
} catch(Exception e)
{}
}
```

* Once we get the reference of user transaction we can store a transaction using the begin method & end the transaction using commit (or) rollback()
* Session.getUserTransaction fails if we specify that the container has to manage the transaction.

```
*  public void openAccount(String cid,Srtring cname,String caddr,String obal)
   {
     try
     {
       //code to perform db operations…….
    } catch (Exception e)
      {
          sessionContext.setRollbackOnly();
          }
  }
```

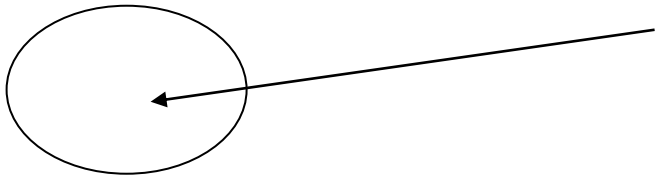
* setRollbackOnly() is called to tell the container that there is some problem in carrying out the transaction In this case the container internally executes ut.rollback() to roll back the transaction.

We can also throw Runtime Exception to indicate that there is some problem in carrying out the transaction.

* When we specify the container managed transaction so in that the default the container will take care of transactions.
* UserTransaction.rollback() (ut.rollback()) is used to roll back the transaction, this can't be used in case of container managed transaction.
* sessionContext.setRollbackOnly() can be used to tell the container that something gone has wrong , When this method is executed the transaction will not be rollback.

It is the responsible of the container to call ut.rollback() for rolling back the transaction.

* A standalone application ,an applet, a servlet which invokes a method on an EJB is called as a client of ejb.

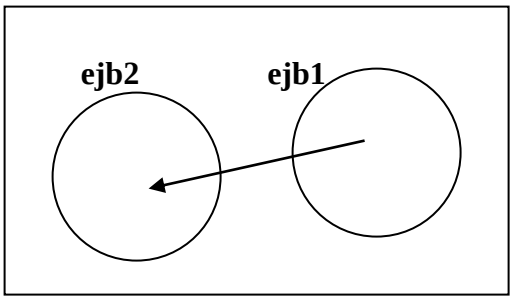


java App

Ejb Container

* The above diagram App (standalone) is acting as a client & this is called as remote client.

An **ejb1** can call a method on **ejb2** running in the same container , as shown in the below diagram in this case **ejb1** is called as the client of **ejb2** as both the ejb's are in the same container **ejb1** is called as local client of **ejb2**.



ejb Container

* Similar to remote business interface we have local business interface.
* Similar to remote home interface we have local home interface.

Remote Business Interface :

```
public interface ejbtwo extends EJBObject
{
        public void mx() throwas RemoteException
        public void my() throws RemoteException
}
```

Local Business Interface

```
public interface ejbtwoLocal extends EJBLocalObject
{
        public void mx();
        public void my();
}
```

Remote Home Interface

```
public interface ejbtwoHome extends EJBHome
{
```

```
  public ejbtwo create() throws CreateException, RemoteException;
}
```

Local Home Interface

```
public interface ejbtwoLocalHome extends EJBLocalHome
{
        Public ejbtwoLocal create() throws CreateException;
}
```

* It is the responsibility of the container to generate the classes implementing
    1. Remote Home Interface
    2. Remote Business Interface
    3. Local Home Interface
    4. Local Business Interface.

*** Stubs & Skeletons are required only if the client is a remote client.

*** If we use **Remote Business Interface & Remote Home Interface** from a local client internally stub & skeleton will be used unnecessary.

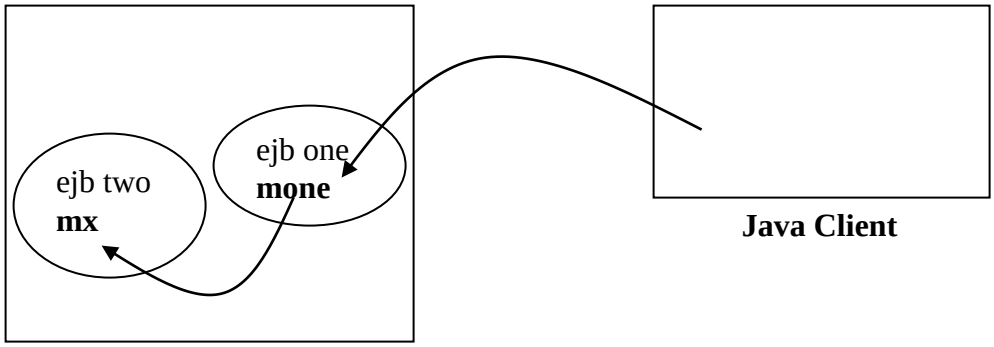
*** We can eliminate stubs & skeletons by using local interfaces from the local clients by eliminating stubs & skeletons we can improve the performance.

* When we provide remote & local interfaces we must provide
    1. JNDI name
    2. Local JNDI name in the Deployment Descriptor.

* For Developing a local client we can use
    Local Jndi Name ⟶ ejbtwoLocal
    Local Home Interface ⟶ mbproj.ejbtwoLocalHome
    Local Business Interface ⟶ mbproj.ejbtwoLocal

* In case of container like weblogic stubs & skeletons are not used even if the Remote Interface are used by local clients.
* We use short circuiting to eliminate the Stubs & skeletons this is done by weblogic people.

Transactional Attributes



Mandatory :

1 If a client is not participating in transaction then the container will throw an Exception.

2. If a client is participating a transaction then the container will run the method in the transaction provided by the client (is the container will not start a new transaction)

* **MANDATORY :**

1. **Client without transaction :**
   Illegal, Container throws an Exception.
2. **Client with transaction :**
   The container runs the method in the transaction of the client.

* **NEVER :**

1. **Client without transaction :**
   Method will be executed **without a Transaction**.
2. **Client with transaction :**
   Illegal , Container throws an Exception.

Note : Shouldn't be used if the method performs DB operations.

* **REQUIRED :**

1. **Client without transaction :**
   Container starts a new transaction and runs the method in the new transaction.
2. **Client with transaction :**
   The container runs the method in the transaction of the client.

* **REQUIRES NEW :**

1. **Client without transaction :**
   Container starts a new transaction and runs the method in the new transaction.
2. **Client with transaction :**
   Container starts a new transaction & runs the method in the new transaction.

Note : In this case when a client calls the transaction **t1** then the container create a new transaction **t2** & executed, If we specify **rollbackOnly** also then it also commit the old transaction why because in that the transaction may execute in new transaction **t2** ie it is not execute in old transaction **t1**.

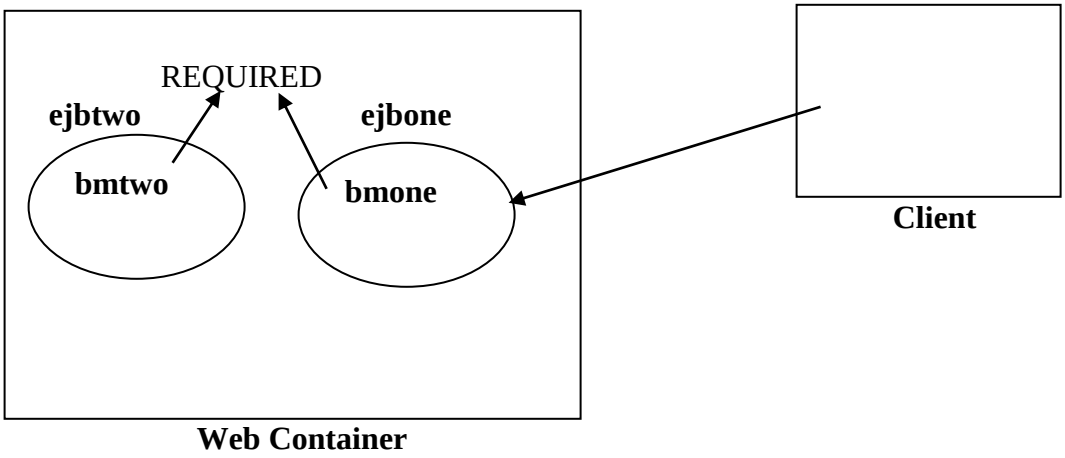
* **SUPPORTS :**

1. **Client without transaction :**
   Container runs the method without a transaction.
2. **Client with transaction :**
   The Container runs the method in the transaction of the client.

* **NOT SUPPORTED :**

1. **Client without transaction :**
   Container runs the method without a transaction.
2. **Client with transaction :**

Container runs the method without a transaction.



Ex:

```
1.  try{
        insert into tab1
        ejbtwo.bmtwo();
       }
        catch(Exception e)
        {
          sessionContext.setRollbackOnly();
        }

2.  try{
         insert into tab2
        }
        catch(Exception e)
        {
          sessionContext.setRollbaclOnly();
        }
```

* What are ENV-ENTRIES ?
* How to access ENV-ENTRIES ?
* What is ENC (Environment Naming Context) ?

* We can develop flexible applications by avoiding hard coding in case of EJB's we can use Environment Entries for this Environment Entries are similar to **init parameter / context parameters** used in servlet based application.
* We can access environment entries using a special context called Environment Naming Context (**ENC**).

In weblogic server we trying to create a transaction so we need to start weblogic builder it is located in

C:\bea\weblogic700\server\bin ⟶ startWLBuilder

*After that it is open a new window in that we select any one project like **baproj.jar** file open it and deploy the that project

* After that we choose **Resources** option add **environment** option & give the **entry name, entry type , entry value**.

Ex: I give a sample names like

Environment Name	Environment type	Environment value
Ds	java.lang.String	abcd
age	java.lang.Integer	25
Price	java.lang.Float	25461.23

How to configure JBOSS Server ?

F:\>jar xvf jboss-3.2.3.zip
F:\ jboss-3.2.3\bin>

Set the classpath

F:\ jboss-3.2.3\bin>set JAVA_HOME=c:\j2sdk1.4.1_01

In Windows Os type **run.bat**
In Linux Os type **run.sh**

In JBOSS we deploy the web application so we can copy the jar file is
F:\jboss-3.2.3\server\default\deploy

In JBOSS we open the index file so we can type below URL
http://localhost:8080/jmx-console

In JBuilder we configure the JBOSS server it is the same process is weblogic server previous classes we have done plz see previous class notes.

* **java:comp/env** is the name of a special context called as environment naming context, environment entries of an ejb will be available under this context.

```
public void mone()
{
  try
  {
    Context ic = new InitialContext();
    Context enc = (Context) ic.lookup("java:comp/env");
    String a = (String) enc.lookup("ds");
   Integer i = (Integer) enc.lookup("age");
  Float f = (Float) enc.lookup("price");
  System.out.println(a);
  System.out.println(i);
  System.out.println(f);
 }catch(Exception e)
  {}
```

}

* In order access jboss frame we must use **jbossall-client.jar**.
* It is copy in our work directory means we have compile client file .
  It is available in **jboss-3.2.3 → client**.

D:\work> set CLASSPATH=jbossall-client.jar;.

* In order to acess the EJB running in JBOSS we must provide the code as shown below.

```
HashTable h= new HashTable()
// set the properties that are required
```

Note: we can get the list of property names. And property values from JBOSS docs.

```
Context ic=new InitialContext(h);
Object o = ic.lookup("info/inetsolve/bean/one");
SBoneHome homeref=(SBoneHome)o;
h.put("java.naming.factory.initial","org.jnp.interfaces.NamingContextFactory");
h.put("java.naming.provider.url","jnp://localhost:1099");
----------------------------------------------------------------------
----------------------------------------------------------------------
----------------------------------------------------------------------
```

* **Service Locator Design Pattern :**

We can consider an ejb as a service in order to acess the ejb we must get the reference the home object.

Getting the reference of the home object using **ic.lookup** is called as locating a service.

We can consider connection pool as a service.

In order to use the connection pool we must use **ic.llokup()** to get the reference of data Source this can be considered as locating a service.

A Service Locator Design pattern suggest that we can provide a separate class which takes care of locating the service.

```
public class WLServiceLocator
{
  public service object locateService(String jndi_name) throws Exception
   {
      Hashtable h= new Hashtable();
h.put(Context.INITIAL_CONTEXT_FACTORY,"weblogic.jndi.WLinitialContex
                                                                tFactory");
h.put(Context.SECURITY_PRINCIPAL,"admin");
Context ic=new InitialContext(h);
Object o=ic.lookup(jndi_name);
return 0;
}
}
```

* How do you transfer data from the client to the ejb or ejb to the client?

Ans: We can use data transfer Object Design pattern for transforming the data between the client & EJB.

HIBERNATE

* **O – R – M (Object Relation Mapping) tools**

* In the development of a project we use the relational database management system like oracle , MySql , Sql Server , Sybase etc. to store the data using a set of tables for Ex : to store the information about the employees working for a company we can use employee table as shown below.

Empno (pk)	ename	salary	address	od
1	eone	12345	aone	xxxx
2	etwo	23456	atwo	yyyy
3	ethree	34567	athree	zzzz
4	efour	45678	afour	aaaa

* When we develop a project using object oriented programming languages like java, c++ ,c Sharp etc. we tri to represent every thing as an object for ex : In our program we can use an employee objects to hold the information about an employee i.e to represent the information about 10 employees in an application we can create 10 employee objects.

```
public class Employee
{
 private Long empno;
 private String empName;
 private Float salary;
 private String address;
 private String odd;
 // Setters & getters to support
 // empno, empName ,salary, address & odd
}
```

* The tools like **hibernate , toplink** etc are called as object relational mapping tools these tools represent the data available in the data base as objects (i.e these tools are maps the objects to the data in the RDBMS)
* **Hibernate** can be use as part of managed environment as part of EJB's & unmanaged environment ex: hibernate usefrom servlets , standalone java applications.
* The developers of hibernate suggests that we can use hibernate to perform 90 percent of database related task of our project using hibernate without sacrificing the performance of application.
* In a project we may need to perform bulk updates (modifying huge no.of records in the data base for these kind of operation it is recommended to use JDBC API directly instead of using Hibernate).
* Hibernate can use the connections that are managed by the connection pool of a product like weblogic (or) use a connection from its own connection pool.
* We must provide the details of the jdbc driver class , jdbc url user name ,password to hibernate software so that it can create a connection pool on its own.
* The default file name for storing the configuration details is **hibernate.cfg.xml**

* To configure the MySql Data Base

1 Install MySql Software
2. It is located in **c:\MySql**
3. we enter in **c:\MySql\bin>**
4. **c:\MySql\bin>mysql**
5. it is displaying in **MySql >**
6. **MySql> use test**

* The classes & interfaces provided by hibernate are packed as part of **hibernatex.jar** there classes & interfaces internally uses several other classes & interfaces which are available as part of various jar files.
* Hibernate performs insertion, update, deletions operations using where class.
* Hibernate uses the primary key to identify the row while performing the data base operation in the where class similar to the following statements.

Ex:

```
update emp set ename='new name' where emp=10;
Select eno,ename,sal from emp where eno=10;
```

* If we need to access the data from the above table we must provide a java class to the hibernate as shown below.

Ex: **Employee.java**

```
package our.pack;
public class Employee{
private Long empNumber;
private String empName;
private Float salary;
public void setEmpNumber(Long i){
        empNumber=i;
}
public void setEmpName(String n){
        empName=n;
}
public void setSalary(Float a){
        salary=a;
}


public Long getEmpNumber(){
        return empNumber;
}
public String getEmpName(){
        return empName;
}
public Float getSalary(){
        return salary;
}
}
```

* Hibernate documents is recommended to using wrapper classes instead of integer means primitive data types.
* In the above class we have not provided constructors when we compile this compiler will add a zero argument constructor when we add constructors to our java classes we must provide a zero argument constructor as hibernate internally uses a zero argument constructor.
* We must specify the information about the name of the class that has to be used while deciding with a table in a file called as hibernate mapping file (HBM).

* To copy the Existing Student1.hbm.xml file & rename the Employee.hbm.xml.

Ex : **Employee.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="our.pack.Employee" table="emp">
      <id name="empNumber" type="Long">
```

```
        <column name="eno" />
        <generator class="assigned" />
      </id>
      <property name="empName" type="string">
        <column name="ename" length="10" />
      </property>
      <property name="salary" type="float">
        <column name="sal" />
      </property>
   </class>
</hibernate-mapping>
```

* Hibernate uses the information available in the mapping files to access the data from the tables.
* As part of **hibernate.cfg.xml** we must provide the information like the jdbc driver class jdbc url , user name, password the names of the HBM files as shown below.

Ex: **hibernate.cfg.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
<hibernate-configuration>
   <session-factory>
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
      <property name="hibernate.cglib.use_reflection_optimizer">true</property>
      <property
name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
      <property name="hibernate.connection.url">jdbc:mysql://localhost/test</property>
      <property
name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <mapping resource="Employee.hbm.xml" />
   </session-factory>
</hibernate-configuration>
```

* The most import interface that are used as part of hibernate are
  1. session
  2. SessionFactory
  3. Query.

* Session org.hibernate.session object is required to perform the data base operations.
* To create the session object we must use SessionFactory object we must use Configuration object.

* Hibernate internally uses **log4j package** uses **log4j properties** file.
To see all the messages generated by hibernate then we have to write in **log4j properties** file is **log4j.rootLogger=DEBUG, A1**
To see only serious error messages then we have to type **log4j.rootLogger=FATAL, A1**

* When configure method is called on configure object is created it searches for hibernate.properties (or) hibernate.cfg.xml If these files are not available it will throw an Exception.
* Configure method reads the information available in cfg file & the information available in hbf file specified in cfg file as resources.
Note : it will take lot of time to read the information from these files so it is not advisable to call configure() method repeatedly .
* Using the configuration object we must create session factory object.
* Creation of SessionFactory is also an expensive operation this is why we should repeatedly call build SessionFactory().
* As SessionFactory creation takes more amount of space it is considered as heavy weight object.

Ex: **App.java**

```
import our.pack.Employee;
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
 Transaction tx=session.beginTransaction();
// Code to save (insert) a object to DB.
 our.pack.Employee e1,e2;
 e1=new our.pack.Employee();
 e2=new our.pack.Employee();
 e1.setEmpNumber(new Long(1));
 e1.setEmpName("EmpOne");
 e1.setSalary(new Float(11111));
 e2.setEmpNumber(new Long(2));
 e2.setEmpName("EmpTwo");
 e2.setSalary(new Float(22222));
 session.save(e1);
 session.save(e2);
 tx.commit();
 session.close();
```

```
}
}
```

* Session object is a light weight object & it is not **thread Safe** (it is not recommended to use a session object from multiple threads)
* SessionFactory can be used from multiple threads ie it is thread safe.
* In order to perform the transaction we must use a transaction object.
* Hibernate.cfg.xml in that file we have to give

< property name="show_sql"> true </property> in this it is display the sql statements in the run time.

* If we give **false** it is not displaying.
* **save** method can be used to ask hibernate to save (insert) the data available in java object in the data base table.
* Hibernate groups a set of statements together & sends the statements at once to the data base server as a batch this improves the performance of the application.

* **tx.commit()** is execute hibernate checks the object that are modifyied (this is called dirty checking) & hibernate issues the appropriate SQL statements (insert , update , delete) to make the changes to the data base (ie hibernate synchronizes the data base with the data available in the java objects).
* We can use the method **session.evict** to remove the objects reference from the hibernates session cache.
* The evicted object is called as detached an object .
* The changes made to the detached object will not be synchronized with the data base **session.evit(e1);**
* We can add (reattach) the object that was evited earlier using **session.update()**

Ex: **App.java**

```
import our.pack.Employee;
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
// Code to save (insert) a object to DB.
 our.pack.Employee e1,e2;
 e1=new our.pack.Employee();
 e2=new our.pack.Employee();
 e1.setEmpNumber(new Long(1));
 e1.setEmpName("sudhi");
 e1.setSalary(new Float(256347));
 e2.setEmpNumber(new Long(2));
 e2.setEmpName("soudhi");
```

```
 e2.setSalary(new Float(123546));
  session.update(e1);
  session.update(e2);
  tx.commit();
 session.close();
}
}
```

* When the update() is called the objects will be add to the cache & Hibernate. When the session is flushed hibernate executes update statement & make a change to the existing row with eno=3 if there is no row hibernate throws an Exception.
* **session.save** or **session.update** methods perform either save (insert) or update.
* save or update method either saves the data (insert) or updates the existing data (using update sql statement).

* In **Employee.hbm.xml** file we use **<generator class="increment" />**
* When we use increment as a class for the generator
    1. We should not set the value of the primary key in our code while saving a new object.
    2. When the object is saved hibernate automatically generates the primary key.

Ex: **App.java**

```
import our.pack.Employee;
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
// Code to save (insert) a object to DB.
 our.pack.Employee e1,e2;
 e1=new our.pack.Employee();
 e2=new our.pack.Employee();
 e1.setEmpName("sudarshan");
 e1.setSalary(new Float(456872));
 e2.setEmpName("soudhi");
 e2.setSalary(new Float(66666));
 session.save(e1);
 session.save(e2);
 tx.commit();
 session.close();
```

```
}
}
```

* If there are no rows in data base & above code as executed hibernate generates the primary key's as one & two & stores the data in data base.
* Hibernate supports other generator class also.
* In the earlier example we have used **<generator class="assign" />** to indicate that hibernate should not automatically generate the id's (ie by using assign we are telling hibernate that our code will assign the value for primary key).

* **seession.delete()** can be used to ask hibernate to delete the data from the data base.
* Sequence generator class internally uses sequence object (or) java object supported by oracle If we use sequence with mysql hibernate fails to generate id's.

Ex: <generator class="sequence"/>

Ex : **App.java**

```
import our.pack.Employee;
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
 our.pack.Employee e1,e2;
 e1=new our.pack.Employee();
 e2=new our.pack.Employee();
 session.load(e1,new Long(1));
 session.delete(e1);
 tx.commit();
 session.close();
}
}
```

* If we use generator class **native** hibernate uses better strategy that suits the data base server.

Ex: <generator class="native"/>

* create table emp(eno integer primary key auto_increment, ename varchar(10) ,sal float);

* Here automatically MySql will generate the eno values.
* If we use auto increment MySql server will add the value for primary key automatically.
* In case of MySql we must create a table with auto increment future to use native generator class.

use in **Employee.hbm.xml :**

<generator class="native" />

Ex : we use to delete **App.java**

```
import our.pack.Employee;
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
our.pack.Employee e1,e2;
 e1=new our.pack.Employee();
 e2=new our.pack.Employee();
 e1.setEmpNumber(new Long(1));
 e2.setEmpNumber(new Long(2));
 session.delete(e1);
 session.delete(e2);
 tx.commit();
 session.close();
}
}
```

* Hibernate internally uses delete() that's why we are using primary key.
* Hibernate supports a language called (**HQL**) **Hibernate Query Language** which is similar to SQL.

Ex: App.java

```
 import java.util.*;
import our.pack.Employee;
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App1{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure("hibernate.cfg.xml");
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
our.pack.Employee e1,e2;
 Query qry=session.createQuery("from our.pack.Employee");
 List elist=qry.list();
 System.out.println("one of objects"+elist.size());
 for(int i=0; i<elist.size();i++)
```

```
 {
  e1=(our.pack.Employee)elist.get(i);
  System.out.println(e1.getEmpNumber());
  System.out.println(e1.getEmpName());
  System.out.println(e1.getSalary());
  System.out.println("----------------");
  }
  tx.commit();
  session.close();
 }
}
```

Output is :

```
 one of objects2
1
EmpOne
11111.0
----------------
2
EmpTwo
22222.0
----------------
```

* java soft has provided an API called reflection API which can be used to get the information about the details of in class as an alternative to this API sum products are using **cg (code generation) lib**.
* (**POJO** ⟶ plained old java object) It it reference to normal java class.
* When **query.list** is executed hibernate converts hql to sql & sends it to the data base server. Hibernate gets the data & creates java objects by placing the data in the java objects.
* The objects that contains the data will be held in the hibernates session cache.
* We can use **session.clear** to remove the references of the objects from hibernates session cache.

Name of the class

```
Query qry=session.createQuery("from Employee as emp where emp.empNumber<3");
```

alias

Name of the property

* The above query returns a list of employee kobjects whose empNumber is less then 3

Ie the out pit is :

```
 one of objects2
1
EmpOne
11111.0
----------------
2
EmpTwo
22222.0
```

---

* In the above query we have used **3** directly instead of directly specying the value we can use positional parameters (or) named parameters.

Query qry=session.createQuery("from Employee as emp where emp.empNumber<? and emp.empNumber>?");
qry.setParameter(0,"5");
qry.setParameter(1,"2");
* remaining is the same as above program.
Output is:
one of objects2
3
EmpThree
33333.0

4
EmpFour
44444.0
---
* The **?** in the above HQL are called as positional parameters the first question mark is parameter **zero** & second question mark is **one** is called as parameter one in order to execute the **list()** , We must set the values of the parameters for this we must use **setParameter()**.
* We can write the same query using named parameters as shown below.

Query qry=session.createQuery("from Employee as emp where
emp.empNumber<:maxval and emp.empNumber>:minval");
qry.setParameter("minval","1");
qry.setParameter("maxval","5");

Output is :

one of objects3
2
EmpTwo
22222.0

3
EmpThree
33333.0

4
EmpFour
44444.0
---
* In the above **:minval** & **:maxval** are provided to specify two parameters **minval & maxval** .
* In order to execute the **list()** method we must said the value of the named parameter minval & maxval as shown below.
qry.setParameter("minval","1");

qry.setParameter("maxval","5");

* The advantage is the readablilty of the code so that's why the developers may prefer to use.

* We can use criteria object to specify the restrictions and sort order.

Note:-

When we use the queries to retrieve the columns selectively using hql as shown below.

```
Query qry=
        session.createquery("select emp.empName from   our.pack.Employee as emp");
```

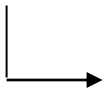
In the App4 program only

```
After for{
            Object o=elist.get(i);
             System.out.println(o.getClass());
           }
```

Note:-

When the query is executed hibernate places the employee name in string objects.

[[[L java/lang/String



Declaration of 3D array.

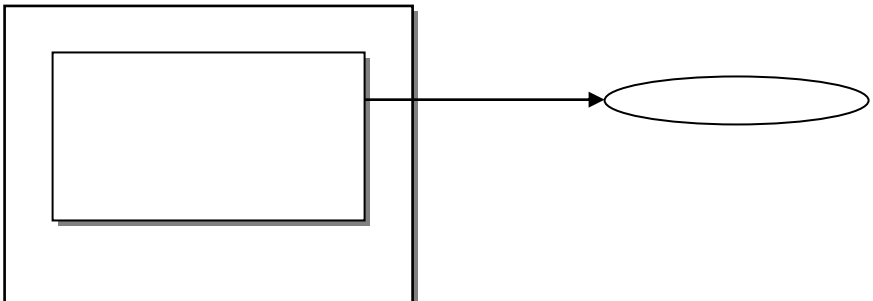
```
public class Test
{
 p.s.v.m(String[] args) throws Exception
 {
  Object o[][][]=new Object[10][12][15];
  S.o.pln(o);
 }
}
```

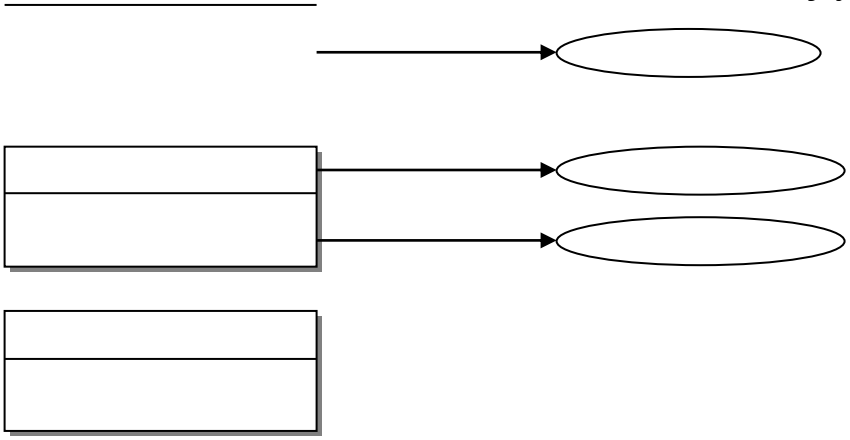
Program App4

```
        Query qry=
      session.createQuery("select emp.empName,emp.salary from our.pack.Employee as emp");
```

Note:-

When we above query is executed hibernate will create array's size 2 and place empName in array[0] and empNumber.





```
List elist=qry.list();
for(i=0;i<elist.size();i++)
{
   Object o[]=(Object[])elist.get(i);
  S.o.pln(o[0]);
  S.o.pln(o[1]);
  S.o.pln(o[2]);
}
```

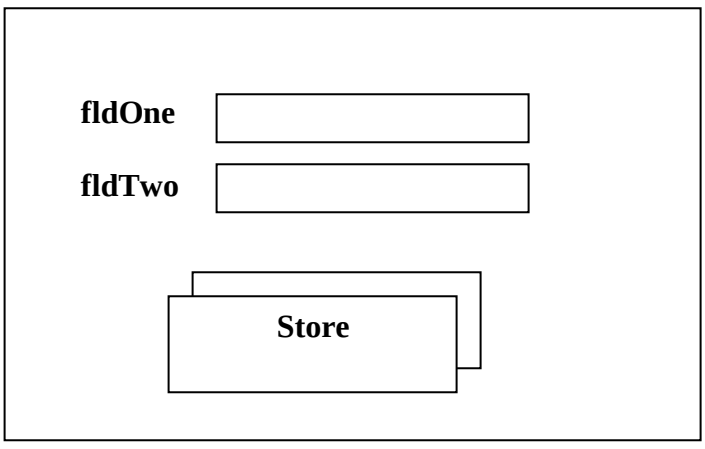
Note:-

In the old version of hibernate we have session.find() method that can be used to get the data using hql. In the new version as an alternative to this we have session.createQuery is provided.
F:\hibernate\docapi\index.html.

* **stmt.executeUpdate("truncate table emp");**
Returns **0 (**zero) as count as the statement is DDL statement.

* **stmt.executeUpdate("delete from emp");**
Returns **n** as count where **n** is no.of rows deleted it is an DML statement.

* To using Servlet accepting Hibernate procedure.
1 .copy **Servlet-api.jar** in our Hibernate /lib folder.



MySrv.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.hibernate.cfg.*;
import org.hibernate.*;
import our.pack.Employee;
public class MySrv extends HttpServlet{
 public void service(HttpServletRequest request,HttpServletResponse response)throws ServletException,IOException
 {
   String vfldOne,vfldTwo;
 PrintWriter out=response.getWriter();
try
 {
  vfldOne=request.getString("fldOne");
  vfldTwo=request.getString("fldTwo");
  // code to use Hibernate.
  Configuration conf;
  conf=new Configuration();
  conf=conf.configure();
  SessionFactory sf=conf.buildSessionFactory();
  Session session=sf.openSession();
  Transaction tx=session.beginTransaction();
  Employee e1=new Employee();
  e1.setEmpName(vfldOne);
  e1.setEmpSalary(vfldTwo);
  session.save(e1);
  tx.commit();
  session.close();
 }catch(Exception e)
  {}
 }
}
```

* In order to run the above servlet successfully in any container.

Step1 : Copy hibernate related jar files & the jdbc driver of the data base under WEB-INF\lib

Step2 : Copy servlet class means our class & employee class under WEB-INF/classes Directory (means MySrv.class, our pack).

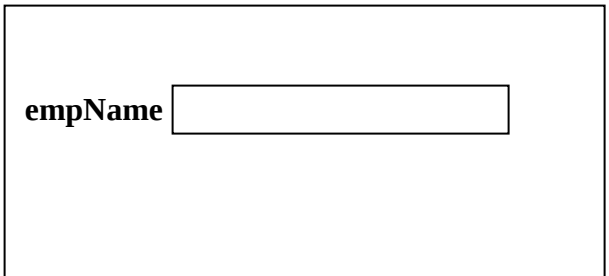
Step3: Copy Hibernate.cfg.xml, log4j.properties, Employee.hbm.xml under WEB-INF\classes directory

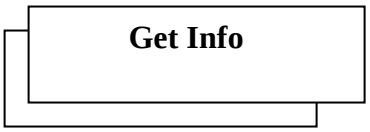
Step4 : configure the servlet by adding the information in web.xml.

Ex: **App3.java**

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import org.hibernate.cfg.*;
import org.hibernate.*;
import our.pack.Employee;
public class App3 extends HttpServlet{
 public void service(HttpServletRequest request,HttpServletResponse response)throws
ServletException,IOException
 {
   String vfldOne,vfldTwo;
 PrintWriter out=response.getWriter();
try
 {
  vfldOne=request.getString("fldOne");
  vfldTwo=request.getString("fldTwo");
  // code to use Hibernate.
  Configuration conf;
  conf=new Configuration();
  conf=conf.configure();
  SessionFactory sf=conf.buildSessionFactory();
  Session session=sf.openSession();
  Query qry=sessio.createQuery("from our.pack.Employee");
  java.util.List elist=qry.list();
  for(int i=0; i<=list.size();i++)
  {
   e1=(our.pack.Employee)elist.get(i);
   out.print(e1.getEmpNumber());
   out.print(e1.getEmpName());
   out.print(e1.getSalary());
  }
  session.close();
 }catch(Exception e)
  {}
 }
}
```

* The above servlet takes more amount of time as we have provided the code to create the sessionFactory in service method in this case every time.





* When we send the request sessionFactory will be created & this process takes lot of time.
* To improve the performance we can use **HibernateUtil class**. This class is designed according to **thread local design pattern**.

* The code for this class is provided as part of hibernates reference manual.

Session session=our.pack. hibernateUtil.currentSession();
tx.commit();
our.pack.HibernateUtil.CloseSession();

* The advantages of using the above code are
1. The sessionFactory object will be created only once ie we will maintain singleton object.
2. Even if a thread calls current session method for **n** no.of times it will return the reference of only one sessionObject.

* What is thread local Design Pattern?
* Thread local class can be used to manage thread local variables. (ie the thread local class maintain one copy of any object for every thread).
* In case Hibernate Util class thread local object is used to maintain one copy of session object for every thread.
* Hibernate util class (Even though it is not part Hibernate API) is being widely used by most of the as part of hibernate developers as part of standalone, servlet, ejb projects.
* In order to access multiple data bases from hibernate based application we need to provide multiple **hibernate.cfg.xml** files.

* We can use Hibernate to access table with primary keys with multiple coloumns.

Ex : **Student.java**

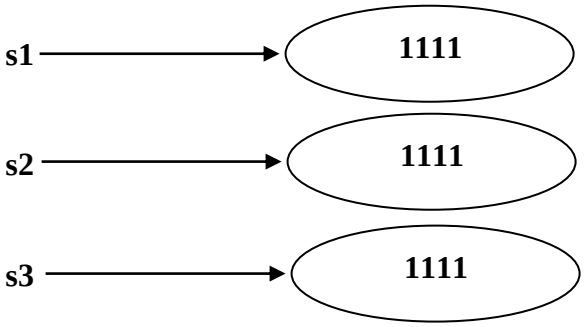
```
public class Student
{
  public int sid;
  public String name;
  public String fname;
}
```

* In the above class we have not provided equals & hashCode method.

```
public class App
{
```

```
 public static void main(String args[])
 {
   Student s1,s2,s3;
   s1=new Student();
   s2=new Student();
  s3=new Student();
  s1.sid=1111;
  s2.sid=2222;
  s3.sid=1111;
 System.out.println(s1.equals(s2));
 System.out.println(s1.equals(s3));
 }
}
```

Out put is : **false**
false



* The above program creates **three** objects to provide the information about ie the object pointed by **s1** is providing a student with a id **1111** & the object pinted by **s3** also provides the information about the same student whose id is also **1111**. but **s1.equals(s3)** returns false. This is because of executing equals method available in object class in order to deal with this kind of faces we must provide our own equals & hash code methods.

Student.java

```
public Boolean equals(object  otherobj)
{
 if (!(otherobj instance of student))
 {
   return false;
 }
 if(otherobj = = this)
 {
   return true;
 }
  Student that=(Student) otherobj;
   if(this.sid= =that.sid)
   {
     return true;
    }
```

```
    return false;
  }
}
```

* If two objects are consider to be equal the hashCode method must return the same hash table.
* hashTable.put method internally uses the hashCode & equals method when we call the when we call put method it internally calls hashCode on the key.

If there is an object available with the same hashCode then it will call equals method If the equals method returns true the object will not be added.

* If we create the exact copy of an object it is called as scrolling.

* **JSP BufferOverflow**:-

Where there is no space to write (or) to store the jsp throws an exception called jspBufferOverflow.

Heap:- heap is an area where the objects are placed.

* where there is no space on the stack to place the local variables then it throws exception called StackOverflow exception.
* The SQL statements that are understood by database server are called native SQLstatements.

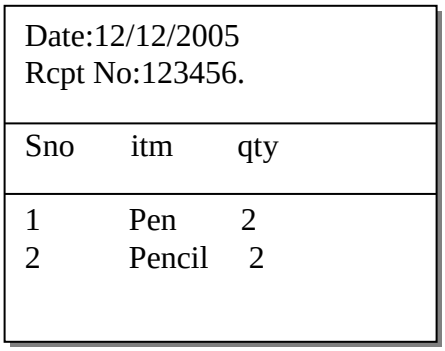
Advantages of EJB:

The Container provides the security service.

Flushing in Hibernate:

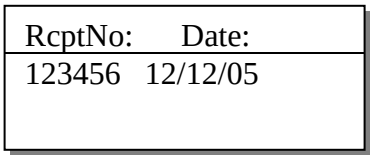
Whatever the changes done to the objects that are applied to the database.

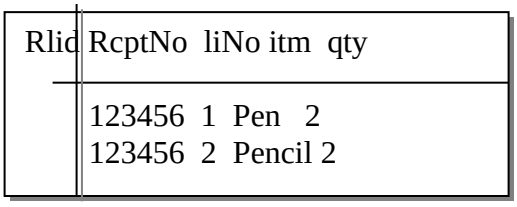
* When we open a file the data will not be written to hard disk, it will be stored in memory and after some time it will be stored in hard disk is called flush of a file.
* A primary key is a combination of one or two columns which is used to identify a row.



Pointer receipt given to Customer in store.

Rcpt





Rcptline.

Info.stored by application in
two different tables as part of DB.

Note:

In the above database design we have rcptline table in which rcptnumber, line number columns combined together are used to identify the rows uniquely (i.e. in this example the primary key is the combination of rcptno and line no. We can access this kind of tables (i.e.) can be accessed from hibernate.

F:\hibernate\part6

* It two objects are equal the hash code that is used by two objects must be return same hash code.

javac .\my\pack\Rcptline.java

```
package my.pack;
public class Rcptline implemnts seriallizable{
  private Integer  rcptId;
  private Integer lineId;
  private String  product;
  private Float qty;
  //setters and getters for 4 props.
  // we must implement hash code and equals methods.
 Public int hashcode()  {
   return rcptId.hashcode()+lineId.hashcode();
 }
 public Boolean equals(Object o){
 if(!(o instance of RcptLine))
       return false;
       if(this==o)
       return true;
  RcptLine otherobj=(RcptLine)o;
  If((this.rcptId.equls(otherobj.rcptId)&&((thisd.lineId.equals(otherobj.lineId)))
     return true;
else
      return false;
   }}   }
```

RcptLine.hbm.xml

```
<class-name=”my.pack.RcptLine” table=”rcptline”>
<composite-id>
      <key-property name=”rcptId” />
      <key-property name=”lineId” />
</composite-id>
<property name=”product” type=”string” />
<property name=”qty” type=”float” />
</class>
```

* Associated with the session means alreadyis available in cache of the session.

Note:-

Hibernate team suggest using a primary key with a single column instead of using multiple columns.

Note:-

Instead of providing a class rcptline as shown above we can provide a class like

```
Package my.pack;
Public class RcptLinePK{
 private Integer rcptId;
 private Integer lineId;
  //setters and getters for 2 props.
  //we must implement hash code and equals methods.
}
package my.pack;
public class RcptLinePK
{
  private String rlpk;
  private String product;
  private Float qty;
//setters and getters for 3 props.
// we must implement hashcode and equals methods.
}
```

Note:-

How to deal with a cable that uses multiple columns as a primary key?

Ans(1):- We can add an additional column even though it is not required according to our application or requirements. This columns can be used as primary key.

Ans(2):- We can provide a class supportingall the properties like rcptId, lineId, product, qty and provide the hashcode and equals methods.

Ans(3):- We can provide a primary key class to deal with the columns rcptId and lineId. In this class we must provide hashcode and equals methods. This class can be used as one of the property of the class that actually mapped with the table.

What is a component in hibernate?

Ans:- In the hibernate we can use a class as a property of another class. The class that is used as a property is called as component.

Tid	tname	sal	street	city	state

Teacher

Sid	sname	course

Student

```
public class stucent{
Integer sid;
String sname;
String street;
String city;
String state;
//setters and getters to deal with 6 props.
}

public class Teacher
{
  Integer tid;
  String tname;
  String street;
  String city;
  String state;
//setters and getters to deal with 6 props.
}
```

Note:
In a hibernate based application we can deal with the above two tables using the above two java classes.

If the java programmer want to see street, city, state combined together as an object called as address then we can develop three classes as shown below.

```
package my.pack;
public class Address{
private String street;
private String city;
private String state;
//setters and getters for 3 props.
}
```

Note:- Address class can be used as part of Student and Teacher.

```
package my.pack;
public class Teacher
{
  private Integer tid;
  private String tname;
```

```
  private Float salary;
  private my.pack.Address address;
  public void setAddress)(my.pack.Address a){
          address= a;
  }
  public my.pack.Address getAddress(){
    return address;
  }
  //setters and getters for 3 props.
}
```

* A component is a class which is used as part of another class that is part of table.

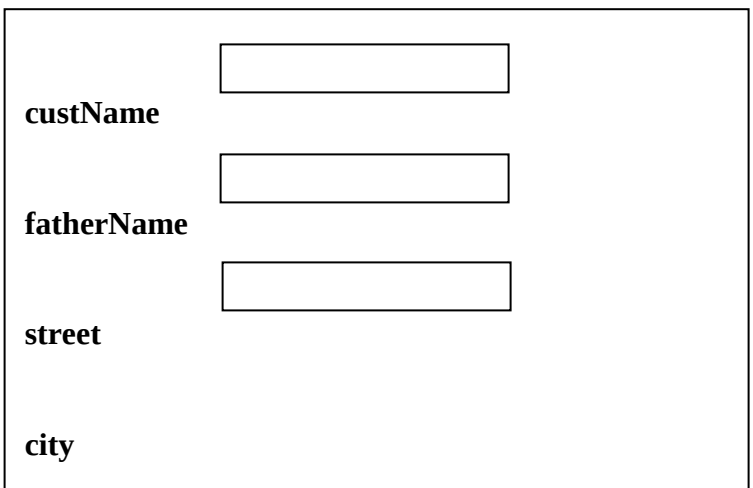
Note:

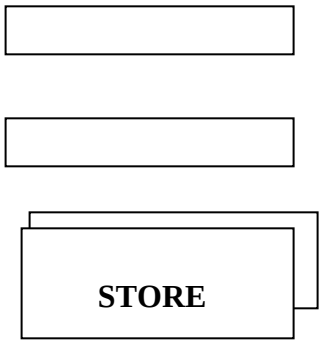
In this example Address class is used as a proerty of student class and teacher class.

* **SOME DATA BASE COMMON QUESTIONS :**

1. What is an entity ?
2. What are the various types of relation ship that can exist between the entities ?
3. What is a transaction ?
4. What is inner join ?
5. What is outer join ?
6. What is co-related sub query ?
7. What is primary key , foreign key ?
8. What are various isolation levels ?
9. What is local transaction ? & what is global transaction ?
10. What low level locking / page level locking / table level locking ?
11. What is single phase commit & two phase commit ?
12. Can u tell us a query to pick up …….. from table ?
13. Can u tell how to pickup the 5^{th} highest sala ry in emp table ?
14. What is Optimestic locking & pasimesting locking ?

Ex : **Hibernate dealing one to one relation**





* If the user provide the details using above form & clicks on the store button the application must store the data in **one table** (or) **two tables** depending upon the design of the data base.
* Using one table the data base designer can design a single table shown below to store the data.

cid (pk)	cname	fname	street	city	state
1	sudhi	k.B.Reddy	E.C.Ngr	HYD	AP

Customer Table

* To access this table data using hibernate data we can develop a class supporting six properties.
* The data base designers can design two tables as shown below to store the same data.

cid (pk)	cname	fname
1	sudhi	K.B.Reddy

Customer

cid (pk)	street	city	state
1	E.C.Ngr	HYD	AP

Address

* The data in address table is linked with (associated with) the data in customer table using the column **cid** of address for every record in customer table is associated with only one record of the address table this is why we can say there exist **one-to-one** relation ship between customer & address.
* Hibernate can be used to maintain the relation ships between various entities.
* In order to get (or) maintain the data from above two table we must use two classes.

1.**Address.java**

```
package my.pack;
public class Address{
private Integer custNo;
private String street;
private String city;
private String state;
public void setCustNo(Integer i){
        custNo=i;
}
public void setStreet(String n){
        street=n;
}
public void setCity(String n){
        city=n;
}
public void setState(String n){
        state=n;
}
public Integer getCustNo(){
        return custNo;
}
public String getStreet(){
        return street;
}
public String getCity(){
        return city;
}
public String getState(){
        return state;
}
}
```

2. **Customer.java**

```
package my.pack;
public class Customer{
private Integer custNo;
private String custName;
private String fatherName;
private my.pack.Address address;
public void setCustNo(Integer i){
        custNo=i;
}
public void setCustName(String n){
        custName=n;
}
public void setFatherName(String n){
        fatherName=n;
}
```

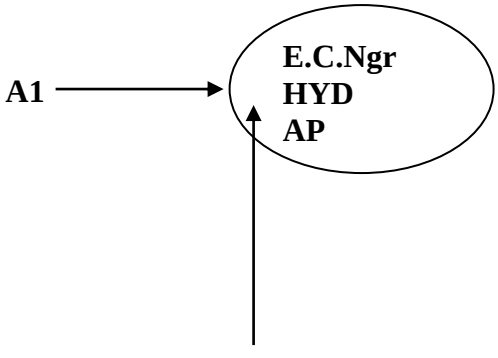
```
public void setAddress(my.pack.Address a){
        address=a;
}
public Integer getCustNo(){
        return custNo;
}
public String getCustName(){
        return custName;
}
public String getFatherName(){
        return fatherName;
}
public my.pack.Address getAddress(){
        return address;
}
}
```

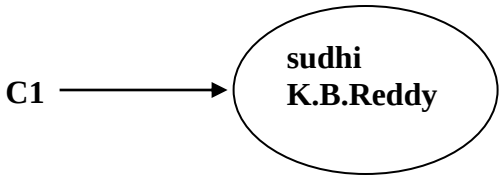
* In the customer class we have provide **four** properties , **three** of them **custNo , custName & fatherName** represents the column of customer table.
* The **fourth** property is used to set the association between **customet & address table**.

Address.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="my.pack.Address" table="address">
<id name="custNo" type="integer">
   <column name="cid" />
   <generator class="increment" />
</id>
<property name="street" type="string">
   <column name="street" length="10" />
</property>
<property name="city" type="string">
    <column name="city" length="10" />
</property>
<property name="state" type="string">
    <column name="state" length="10" />
</property>
</class></hibernate-mapping>
```

Foreign key : It is a key used to link (or) associate with the another table that contain a primary key.





3. **App1.java**

```
import org.hibernate.cfg.*;
import org.hibernate.*;
public class App1{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
my.pack.Address a1=new my.pack.Address();
a1.setCity("HYD");
a1.setStreet("E.C.Ngr");
a1.setState("AP");
my.pack.Customer c1=new my.pack.Customer();
c1.setCustName("sudhi");
c1.setFatherName("K.B.Reddy");
c1.setAddress(a1);
session.save(c1);
tx.commit();
session.close();
}
}
```

4.**App2.java**

```
import org.hibernate.cfg.*;
import org.hibernate.*;
import java.util.*;
public class App2{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Query qry=session.createQuery("from Customer");
System.out.println("----------------------");
```

```
List clist=qry.list();
for(int i=0;i<clist.size();i++){
        my.pack.Customer c=(my.pack.Customer)clist.get(i);
        System.out.println(c.getCustName());
        System.out.println(c.getFatherName());
        System.out.println(c.getAddress().getStreet());
        System.out.println(c.getAddress().getCity());
}
System.out.println("----------------------");
session.close();
}
}
```

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.cglib.use_reflection_optimizer">true</property>
      <property
name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
      <property name="hibernate.connection.url">jdbc:mysql://localhost/test</property>
      <property
name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <mapping resource="customer.hbm.xml" />
      <mapping resource="address.hbm.xml" />
   </session-factory>
</hibernate-configuration>
```

* If we run the above programs then it create two tables ie address table & customer table & insert the data.

Out put is :

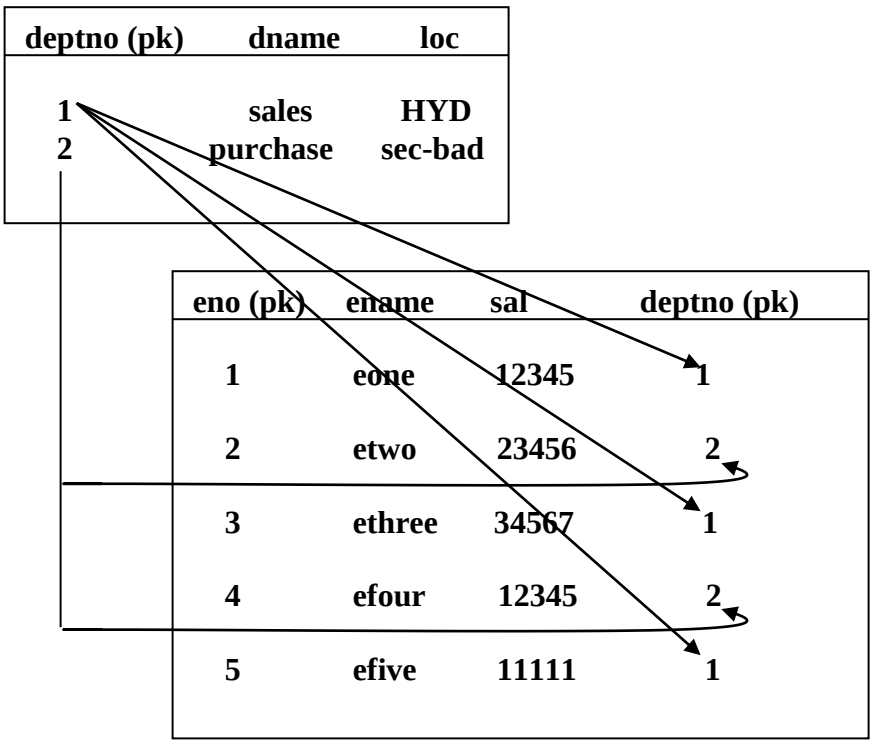
```
----------------------
Hibernate: select customer0_.cid as cid0_, customer0_.cname as cname0_, customer
0_.fname as fname0_ from customer customer0_
Hibernate: select address0_.cid as cid1_0_, address0_.street as street1_0_, addr
ess0_.city as city1_0_, address0_.state as state1_0_ from address address0_ wher
e address0_.cid=?
sudhi
K.B.Reddy
E.C.Ngr
HYD
----------------------
```

* If we specify **cascade=save-update** as part of **one-to-one** eliminate in the customer.hbm.xml file.
<one-to-one name="address" class="my.pack.Address" cascade="all"/>

* Hibernate will insert customer + address data when we use **session.save()** asking hibernate to save customer data.
* Similarly when we ask hibernate to update customer it will update customer & address data.
* If we specify **cascade = delete** when we ask hibernate will delete customer data & the associated address data.
* If we specify **cascade = all** save update & delete operations on customer will be performed on customer as well as address.
* If **cascade = null** then the operations save update delete on customer will not be cascaded to address.

Ex: **Hibernate dealing one to many & many to one relation ship**

Dept Table



deptno (pk)	dname	loc
1	sales	HYD
2	purchase	sec-bad

eno (pk)	ename	sal	deptno (pk)
1	eone	12345	1
2	etwo	23456	2
3	ethree	34567	1
4	efour	12345	2
5	efive	11111	1

Emp Table

* From the data available in the above tables we can say the relation ship between dept & emp is one to many (or) we can also say the relation ship between emp & dept is many to one.

1. **Emp.java**

```
package my.pack;
import java.io.Serializable;
```

```
public class Emp implements Serializable {
   private Integer eno;
   private String ename;
   private Float sal;
   private my.pack.Dept dept;
   public Emp() {
         System.out.println("Emp obj created");
   }
   public Integer getEno() {
      return this.eno;
   }
   public void setEno(Integer eno) {
      this.eno = eno;
   }
   public String getEname() {
      return this.ename;
   }
   public void setEname(String ename) {
      this.ename = ename;
   }
   public Float getSal() {
      return this.sal;
   }
   public void setSal(Float sal) {
      this.sal = sal;
   }
   public my.pack.Dept getDept() {
      return this.dept;
   }
   public void setDept(my.pack.Dept dept) {
      this.dept = dept;
   }
}
```

2. **Dept.java**

```
package my.pack;
import java.io.Serializable;
import java.util.*;
public class Dept implements Serializable {
   private Integer dno;
   private String dname;
   private String loc;
   private Set empSet;
   public Dept() {
         System.out.println("Dept obj created");
   }
   public Integer getDno() {
      return this.dno;
   }
```

```
    public void setDno(Integer dno) {
        this.dno = dno;
    }
    public String getDname() {
        return this.dname;
    }
    public void setDname(String dname) {
        this.dname = dname;
    }
    public String getLoc() {
        return this.loc;
    }
    public void setLoc(String loc) {
        this.loc = loc;
    }
    public Set getEmpSet(){
        return this.empSet;
    }
    public void setEmpSet(Set e){
        this.empSet=e;
    }
}
```

3. **Dept.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="my.pack.Dept" table="dept">
    <id name="dno"  type="integer" column="dno">
        <generator class="increment" />
    </id>
    <property name="dname" type="string" column="dname"  length="10"/>
    <property name="loc" type="string" column="loc"  length="10"/>
    <set name="empSet" cascade="all" inverse="true" lazy="false">
        <key column="deptno"/>
        <one-to-many class="my.pack.Emp"/>
    </set>
</class>
</hibernate-mapping>
```

4. **Emp.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="my.pack.Emp" table="emp">
   <id name="eno" type="integer"
      column="eno">
      <generator class="increment" />
   </id>
   <property name="ename" type="string"  column="ename" length="10"/>
   <property name="sal" type="float"  column="sal" length="12"/>
   <many-to-one name="dept"  class="my.pack.Dept" column="deptno" cascade="all"/>
</class>
</hibernate-mapping>
```

5. **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.cglib.use_reflection_optimizer">true</property>
      <property
name="hibernate.connection.driver_class">org.gjt.mm.mysql.Driver</property>
      <property name="hibernate.connection.url">jdbc:mysql://localhost/test</property>
      <property
name="hibernate.dialect">org.hibernate.dialect.MySQLInnoDBDialect</property>
      <property name="show_sql">true</property>
      <property name="hbm2ddl.auto">update</property>
      <mapping resource="Emp.hbm.xml" />
      <mapping resource="Dept.hbm.xml" />
   </session-factory>
</hibernate-configuration>
```

5. **App1.java**

```
import org.hibernate.cfg.*;
import org.hibernate.*;
import java.util.*;
public class App1{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
```

```
Session session=sf.openSession();
Transaction tx=session.beginTransaction();
Set es=new HashSet();
my.pack.Dept d = new my.pack.Dept();
d.setDname("dept one");
d.setLoc("Hyd");
d.setEmpSet(es);
my.pack.Emp e;
e=new my.pack.Emp();
e.setEname("Eone");
e.setSal(new Float(1111));
e.setDept(d);
es.add(e);
e=new my.pack.Emp();
e.setEname("Etwo");
e.setSal(new Float(22222));
e.setDept(d);
es.add(e);
session.save(d);
tx.commit();
session.close();
}
}
```

6. **App2.java**

```
import org.hibernate.cfg.*;
import org.hibernate.*;
import java.util.*;
public class App2{
public static void main(String args[])throws Exception{
Configuration conf;
conf=new Configuration();
conf=conf.configure();
SessionFactory sf=conf.buildSessionFactory();
Session session=sf.openSession();
Query qry=session.createQuery("from Dept");
System.out.println("----------------------");
List clist=qry.list();
for(int i=0;i<clist.size();i++){
        my.pack.Dept d=(my.pack.Dept)clist.get(i);
        System.out.println(d.getDname());
        System.out.println(d.getLoc());
        System.in.read();
        System.in.read();
        Set es=d.getEmpSet();
}
System.out.println("----------------------");
session.close(); }}
```

* **Immediately Loading (or) Aggressive Loading (or) lazy = "false " :**

If we have Dept table & Emp table & associate with both tables , When ever Hibernate calls it is automatically calls both the objects , means Dept data & Emp data

* **lazy binding (or) lazy = "true " :**

Here hibernate calls only dept data means it is loaded only Dept data.

ie

```
<set name="empSet" cascade="all" inverse="true" lazy="true">
<key column="deptno"/>
<one-to-many class="my.pack.Emp"/>
</set>
```

* Hibernate will load the data from dept table when we execute the following query.

```
Query qry=session.createQuery("from Dept");
```

(ie Hibernate will not load the employee data associated with department data immidiatly).

* If the application doesn't accept employee data then Hibernate will not load the employee data at all in this case.

In this case the employee data will be loaded.

* If we specify **lazy = false** our application executes the above query hibernate will load the data from department & the associated employee table irrespective of whether the application uses employee data.
* If we maintain the association from Dept to Em as well as from Emp to Dept we must specify **inverse="true"**

* If we need to execute a piece of code while weblogic is starting up we must provide a stratup class.

* **Configure Hibernate with weblogic:**

Step 1: Copy Hibernate related jar files to **c:\bea\lib**

Step 2: Create **ourclassses** directory **c:\bea\ourclasses**

Step 3: Provide a **startup** class (refer to hib3courses\part3\MyStartup.java).

* In startup class provide the code to create sessionFactory object & the code to register the sessionFactory object in the Directory service.

Step 4: In order compile stratup class run **wlsetcp.bat** (available in part3).

```
D:\bea\user_parojects\sudarshan> setenv
D:\j2ee\hibernate\part3>  echo %CLASSPATH%
D:\j2ee\hibernate\part3>setenv
D:\j2ee\hibernate\part3>wlsetcp.bat
```

Step 5: In order to use startup class with weblogic we must register the startup class with weblogic.

Procedure :

1) copy the lib directory available in this directory to
the home directory of bea. (ie. c:\bea)

2) create a directory with the name ourclasses under c:\bea
andcopy st artup class,other classes,cfg,hbm under
c:\bea\ourclasses directory

3) open dos window and run wlsetcp.bat

4) run startweblogic command to start weblogic server.

5) using weblogic console create connection pool.

6) using weblogic console create a datasource.

7) configure weblogic startup classes from console.

8) restart weblogic server.

Step 6: As a startup class create a sessionFactory which uses te data source we must configure a **data source** in connection pooling.

Step 7: Configure the connection pool & create a data source with a JNDI name **mysqlds**

Step 8: restart weblogic.

* **session.flush() :** It is only responsible for inserting , updating ,deleting .
* **commit() :** When ever we use commit method it internally calls flush method.

* To deal with how to Deal Entity Beans in data base

* **Entity Bean :**

1. CMP EB : In CMP's containers are responsible for generating jdbc code to access data from DB.

Note : In **ejb 1.x** versions very few facilities are provided with cmp's (ex:is not provided to manage relation ships).

2. BMP EB : In bmp's developer has to provide the jdbc code as part of bean class to access data from Data Base.

1. CMP :

Procedure :

Step 1 : first to create connection pool & Data Source in Weblogic

Ex: in our project **jndi** name is : **orapool**

Step 2 : To copy classes12.jar file in **jbuilderx / lib** directory

Step 3 : To create a table in Data Base.

Ex: in our project is **create table student(sid varchar(10) primary key , sname varchar(20) , address varchar(20));**

Step 4 : open jbuilder & create a new project

Ex: in our project name is **st**.

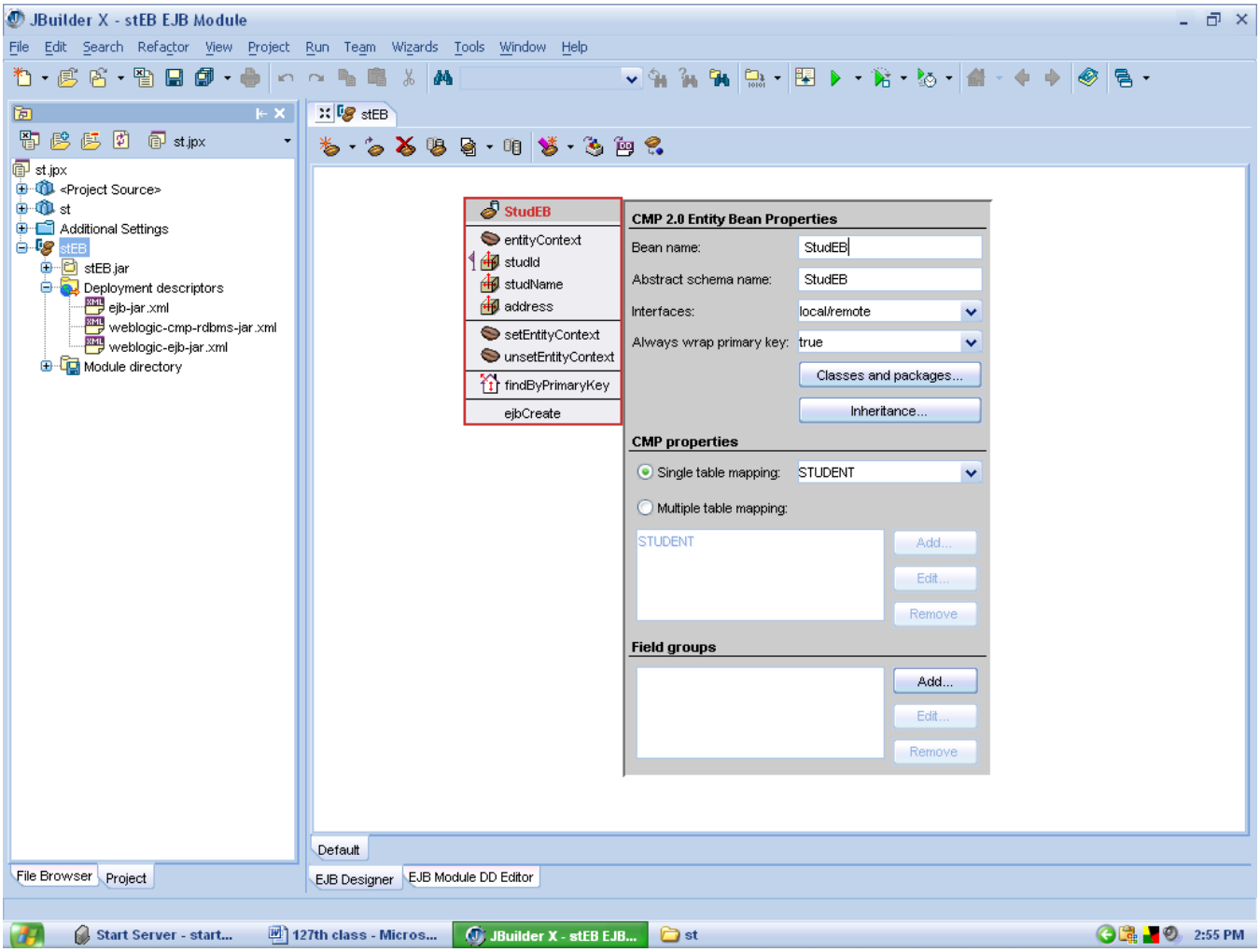
Step 5: After That we select in jbuilder **import Schema From Data Base**

In that We have Select **Driver** , **URL ,UserName , Password , DataBase name, JNDI name** are be given Ex: in our project jndi name is **orapool**.

Step 6: After that we have select in jbuilder **create EJB** menu in that we select **CMP 2.0 Entity Bean** option, it is displaying pop up menu in that we have to given information **CMP 2.0 Entity Bean Properties , CMP Properties** in that select our data base table.

After that we add Fields like studId, studName,address.In last we make the project, deployee the project .

(In these time weblogic server is also in running mode other wise it is given Exception)



* Jbuilder creates jar files & we create client program to access the data in data base.

Client.java

```
import javax.naming.*;
import java.rmi.*;
import java.util.*;
import st.*;
public class Client
{
 public static void main(String args[])throws Exception
 {
   Hashtable h = new Hashtable();
   h.put(Context.INITIAL_CONTEXT_FACTORY,
    "weblogic.jndi.WLInitialContextFactory");
   h.put(Context.PROVIDER_URL, "t3://localhost:7001");
   h.put(Context.SECURITY_PRINCIPAL, "sudarshan");
   h.put(Context.SECURITY_CREDENTIALS, "soujanya");
   Context ic = new InitialContext(h);
   System.out.println(" initial context = "+ ic);
   Object o=ic.lookup("StudEBRemote");
   st.StudEBRemoteHome hr=(st.StudEBRemoteHome)o;
   hr.create("5","sudha","sou");
   System.out.println("Table is inserted.....");
  }
}
```

Source : - http://www.techinterviews.com/index.php?p=89

1. **What is a class?** A class is a blueprint, or prototype, that defines the variables and the methods common to all objects of a certain kind.
2. **What is a object?** An object is a software bundle of variables and related methods.An instance of a class depicting the state and behavior at that particular time in real world.
3. **What is a method?** Encapsulation of a functionality which can be called to perform specific tasks.
4. **What is encapsulation? Explain with an example.** Encapsulation is the term given to the process of hiding the implementation details of the object. Once an object is encapsulated, its implementation details are not immediately accessible any more. Instead they are packaged and are only indirectly accessible via the interface of the object
5. **What is inheritance? Explain with an example.** Inheritance in object oriented programming means that a class of objects can inherit properties and methods from another class of objects.
6. **What is polymorphism? Explain with an example.** In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class. More specifically, it is the ability to redefine methods for derived classes. For example, given a base class shape, polymorphism enables the programmer to define different area methods for any number of derived classes, such as circles, rectangles and triangles. No matter what shape an object is, applying the area method to it will return the correct results. Polymorphism is considered to be a requirement of any true object-oriented programming language
7. **Is multiple inheritance allowed in Java?** No, multiple inheritance is not allowed in Java.
8. **What is interpreter and compiler?** Java interpreter converts the high level language code into a intermediate form in Java called as bytecode, and then executes it, where as a compiler converts the high level language code to machine language making it very hardware specific
9. **What is JVM?** The Java interpreter along with the runtime environment required to run the Java application in called as Java virtual machine(JVM)
10. **What are the different types of modifiers?** There are access modifiers and there are other identifiers. Access modifiers are public, protected and private. Other are final and static.
11. **What are the access modifiers in Java?** There are 3 access modifiers. Public, protected and private, and the default one if no identifier is specified is called friendly, but programmer cannot specify the friendly identifier explicitly.
12. **What is a wrapper class?** They are classes that wrap a primitive data type so it can be used as a object
13. **What is a static variable and static method? What's the difference between two?** The modifier static can be used with a variable and method. When declared as static variable, there is only one variable no matter how instances are created, this variable is initialized when the class is loaded. Static method do not need a class to be instantiated to be called, also a non static method cannot be called from static method.

14. **What is garbage collection?** Garbage Collection is a thread that runs to reclaim the memory by destroying the objects that cannot be referenced anymore.
15. **What is abstract class?** Abstract class is a class that needs to be extended and its methods implemented, aclass has to be declared abstract if it has one or more abstract methods.
16. **What is meant by final class, methods and variables?** This modifier can be applied to class method and variable. When declared as final class the class cannot be extended. When declared as final variable, its value cannot be changed if is primitive value, if it is a reference to the object it will always refer to the same object, internal attributes of the object can be changed.
17. **What is interface?** Interface is a contact that can be implemented by a class, it has method that need implementation.
18. **What is method overloading?** Overloading is declaring multiple method with the same name, but with different argument list.
19. **What is method overriding?** Overriding has same method name, identical arguments used in subclass.
20. **What is singleton class?** Singleton class means that any given time only one instance of the class is present, in one JVM.
21. **What is the difference between an array and a vector?** Number of elements in an array are fixed at the construction time, whereas the number of elements in vector can grow dynamically.
22. **What is a constructor?** In Java, the class designer can guarantee initialization of every object by providing a special method called a constructor. If a class has a constructor, Java automatically calls that constructor when an object is created, before users can even get their hands on it. So initialization is guaranteed.
23. **What is casting?** Conversion of one type of data to another when appropriate. Casting makes explicitly converting of data.
24. **What is the difference between final, finally and finalize?** The modifier final is used on class variable and methods to specify certain behaviour explained above. And finally is used as one of the loop in the try catch blocks, It is used to hold code that needs to be executed whether or not the exception occurs in the try catch block. Java provides a method called finalize( ) that can be defined in the class. When the garbage collector is ready to release the storage ed for your object, it will first call finalize( ), and only on the next garbage-collection pass will it reclaim the objects memory. So finalize( ), gives you the ability to perform some important cleanup at the time of garbage collection.
25. **What is are packages?** A package is a collection of related classes and interfaces providing access protection and namespace management.
26. **What is a super class and how can you call a super class?** When a class is extended that is derived from another class there is a relationship is created, the parent class is referred to as the super class by the derived class that is the child. The derived class can make a call to the super class using the keyword super. If used in the constructor of the derived class it has to be the first statement.
27. **What is meant by a Thread?** Thread is defined as an instantiated parallel process of a given program.
28. **What is multi-threading?** Multi-threading as the name suggest is the scenario where more than one threads are running.
29. **What are two ways of creating a thread? Which is the best way and why?** Two ways of creating threads are, one can extend from the Java.lang.Thread and can implement the rum method or the run method of a different class can be called which

implements the interface Runnable, and the then implement the run() method. The latter one is mostly used as first due to Java rule of only one class inheritance, with implementing the Runnable interface that problem is sorted out.

30. **What is deadlock?** Deadlock is a situation when two threads are waiting on each other to release a resource. Each thread waiting for a resource which is held by the other waiting thread. In Java, this resource is usually the object lock obtained by the synchronized keyword.
31. **What are the three types of priority?** MAX_PRIORITY which is 10, MIN_PRIORITY which is 1, NORM_PRIORITY which is 5.
32. **What is the use of synchronizations?** Every object has a lock, when a synchronized keyword is used on a piece of code the, lock must be obtained by the thread first to execute that code, other threads will not be allowed to execute that piece of code till this lock is released.

Good questions asked during Java interview

1. **Is "abc" a primitive value?** - The String literal "abc" is not a primitive value. It is a String object.
2. **What restrictions are placed on the values of each case of a switch statement?** - During compilation, the values of each case of a switch statement must evaluate to a value that can be promoted to an int value.
3. **What modifiers may be used with an interface declaration?** - An interface may be declared as public or abstract.
4. **Is a class a subclass of itself?** - A class is a subclass of itself.
5. **What is the difference between a while statement and a do statement?** - A while statement checks at the beginning of a loop to see whether the next loop iteration should occur. A do statement checks at the end of a loop to see whether the next iteration of a loop should occur. The do statement will always execute the body of a loop at least once.
6. **What modifiers can be used with a local inner class?** - A local inner class may be final or abstract.
7. **What is the purpose of the File class?** - The File class is used to create objects that provide access to the files and directories of a local file system.
8. **Can an exception be rethrown?** - Yes, an exception can be rethrown.
9. **When does the compiler supply a default constructor for a class?** - The compiler supplies a default constructor for a class if no other constructors are provided.
10. **If a method is declared as protected, where may the method be accessed?** - A protected method may only be accessed by classes or interfaces of the same package or by subclasses of the class in which it is declared.
11. **Which non-Unicode letter characters may be used as the first character of an identifier?** - The non-Unicode letter characters $ and _ may appear as the first character of an identifier
12. **What restrictions are placed on method overloading?** - Two methods may not have the same name and argument list but different return types.
13. **What is casting?** - There are two types of casting, casting between primitive numeric types and casting between object references. Casting between numeric types is used to convert larger values, such as double values, to smaller values, such as byte values.

Casting between object references is used to refer to an object by a compatible class, interface, or array type reference.

14. **What is the return type of a program's main() method?** - A program's main() method has a void return type.
15. **What class of exceptions are generated by the Java run-time system?** - The Java runtime system generates RuntimeException and Error exceptions.
16. **What class allows you to read objects directly from a stream?** - The ObjectInputStream class supports the reading of objects from input streams.
17. **What is the difference between a field variable and a local variable?** - A field variable is a variable that is declared as a member of a class. A local variable is a variable that is declared local to a method.
18. **How are this() and super() used with constructors?** - this() is used to invoke a constructor of the same class. super() is used to invoke a superclass constructor.
19. **What is the relationship between a method's throws clause and the exceptions that can be thrown during the method's execution?** - A method's throws clause must declare any checked exceptions that are not caught within the body of the method.
20. **Why are the methods of the Math class static?** - So they can be invoked as if they are a mathematical code library.
21. **What are the legal operands of the instanceof operator?** - The left operand is an object reference or null value and the right operand is a class, interface, or array type.
22. **What an I/O filter?** - An I/O filter is an object that reads from one stream and writes to another, usually altering the data in some way as it is passed from one stream to another.
23. **If an object is garbage collected, can it become reachable again?** - Once an object is garbage collected, it ceases to exist. It can no longer become reachable again.
24. **What are E and PI?** - E is the base of the natural logarithm and PI is mathematical value pi.
25. **Are true and false keywords?** - The values true and false are not keywords.
26. **What is the difference between the File and RandomAccessFile classes?** - The File class encapsulates the files and directories of the local file system. The RandomAccessFile class provides the methods needed to directly access data contained in any part of a file.
27. **What happens when you add a double value to a String?** - The result is a String object.
28. **What is your platform's default character encoding?** - If you are running Java on English Windows platforms, it is probably Cp1252. If you are running Java on English Solaris platforms, it is most likely 8859_1.
29. **Which package is always imported by default?** - The java.lang package is always imported by default.
30. **What interface must an object implement before it can be written to a stream as an object?** - An object must implement the Serializable or Externalizable interface before it can be written to a stream as an object.
31. **How can my application get to know when a HttpSession is removed?** - Define a Class HttpSessionNotifier which implements HttpSessionBindingListener and implement the functionality what you need in valueUnbound() method. Create an instance of that class and put that instance in HttpSession.
32. **Whats the difference between notify() and notifyAll()?** - notify() is used to unblock one waiting thread; notifyAll() is used to unblock all of them. Using notify() is preferable (for efficiency) when only one blocked thread can benefit from the change

(for example, when freeing a buffer back into a pool). notifyAll() is necessary (for correctness) if multiple threads should resume (for example, when releasing a "writer" lock on a file might permit all "readers" to resume).

33. **Why can't I say just abs() or sin() instead of Math.abs() and Math.sin()?** - The import statement does not bring methods into your local name space. It lets you abbreviate class names, but not get rid of them altogether. That's just the way it works, you'll get used to it. It's really a lot safer this way.
    However, there is actually a little trick you can use in some cases that gets you what you want. If your top-level class doesn't need to inherit from anything else, make it inherit from java.lang.Math. That *does* bring all the methods into your local name space. But you can't use this trick in an applet, because you have to inherit from java.awt.Applet. And actually, you can't use it on java.lang.Math at all, because Math is a "final" class which means it can't be extended.
34. **Why are there no global variables in Java?** - Global variables are considered bad form for a variety of reasons: Adding state variables breaks referential transparency (you no longer can understand a statement or expression on its own: you need to understand it in the context of the settings of the global variables), State variables lessen the cohesion of a program: you need to know more to understand how something works. A major point of Object-Oriented programming is to break up global state into more easily understood collections of local state, When you add one variable, you limit the use of your program to one instance. What decided to ban global variables. you thought was global, someone else might think of as local: they may want to run two copies of your program at once. For these reasons, Java
35. **What does it mean that a class or member is final?** - A final class can no longer be subclassed. Mostly this is done for security reasons with basic classes like String and Integer. It also allows the compiler to make some optimizations, and makes thread safety a little easier to achieve. Methods may be declared final as well. This means they may not be overridden in a subclass. Fields can be declared final, too. However, this has a completely different meaning. A final field cannot be changed after it's initialized, and it must include an initializer statement where it's declared. For example, public final double c = 2.998; It's also possible to make a static field final to get the effect of C++'s const statement or some uses of C's #define, e.g. public static final double c = 2.998;
36. **What does it mean that a method or class is abstract?** - An abstract class cannot be instantiated. Only its subclasses can be instantiated. You indicate that a class is abstract with the abstract keyword like this:

```
37.         public abstract class Container extends Component {
```

    Abstract classes may contain abstract methods. A method declared abstract is not actually implemented in the current class. It exists only to be overridden in subclasses. It has no body. For example,

```
public abstract float price();
```

    Abstract methods may only be included in abstract classes. However, an abstract class is not required to have any abstract methods, though most of them do. Each subclass of an abstract class must override the abstract methods of its superclasses or itself be declared abstract.

38. **What is a transient variable? -** transient variable is a variable that may not be serialized.
39. **How are Observer and Observable used? -** Objects that subclass the Observable class maintain a list of observers. When an Observable object is updated it invokes the update() method of each of its observers to notify the observers that it has changed state. The Observer interface is implemented by objects that observe Observable objects.
40. **Can a lock be acquired on a class? -** Yes, a lock can be acquired on a class. This lock is acquired on the class's Class object.
41. **What state does a thread enter when it terminates its processing? -** When a thread terminates its processing, it enters the dead state.
42. **How does Java handle integer overflows and underflows? -** It uses those low order bytes of the result that can fit into the size of the type allowed by the operation.
43. **What is the difference between the >> and >>> operators? -** The >> operator carries the sign bit when shifting right. The >>> zero-fills bits that have been shifted out.
44. **Is sizeof a keyword? -** The sizeof operator is not a keyword.
45. **Does garbage collection guarantee that a program will not run out of memory? -** Garbage collection does not guarantee that a program will not run out of memory. It is possible for programs to use up memory resources faster than they are garbage collected. It is also possible for programs to create objects that are not subject to garbage collection
46. **Can an object's finalize() method be invoked while it is reachable? -** An object's finalize() method cannot be invoked by the garbage collector while the object is still reachable. However, an object's finalize() method may be invoked by other objects.
47. **What value does readLine() return when it has reached the end of a file? -** The readLine() method returns null when it has reached the end of a file.
48. **Can a for statement loop indefinitely? -** Yes, a for statement can loop indefinitely. For example, consider the following: for(;;) ;
49. **To what value is a variable of the String type automatically initialized? -** The default value of an String type is null.
50. **What is a task's priority and how is it used in scheduling? -** A task's priority is an integer value that identifies the relative order in which it should be executed with respect to other tasks. The scheduler attempts to schedule higher priority tasks before lower priority tasks.
51. **What is the range of the short type? -** The range of the short type is -(2^15) to 2^15 - 1.
52. **What is the purpose of garbage collection? -** The purpose of garbage collection is to identify and discard objects that are no longer needed by a program so that their resources may be reclaimed and reused.
53. **What do you understand by private, protected and public? -** These are accessibility modifiers. Private is the most restrictive, while public is the least restrictive. There is no real difference between protected and the default type (also known as package protected) within the context of the same package, however the protected keyword allows visibility to a derived class in a different package.
54. **What is Downcasting ? -** Downcasting is the casting from a general to a more specific type, i.e. casting down the hierarchy
55. **Can a method be overloaded based on different return type but same argument type ? -** No, because the methods can be called without using their return type in which case there is ambiguity for the compiler

56. **What happens to a static var that is defined within a method of a class ?** - Can't do it. You'll get a compilation error
57. **How many static init can you have ?** - As many as you want, but the static initializers and class variable initializers are executed in textual order and may not refer to class variables declared in the class whose declarations appear textually after the use, even though these class variables are in scope.
58. **What is the difference amongst JVM Spec, JVM Implementation, JVM Runtime ?** - The JVM spec is the blueprint for the JVM generated and owned by Sun. The JVM implementation is the actual implementation of the spec by a vendor and the JVM runtime is the actual running instance of a JVM implementation
59. **Describe what happens when an object is created in Java?** - Several things happen in a particular order to ensure the object is constructed properly: Memory is allocated from heap to hold all instance variables and implementation-specific data of the object and its superclasses. Implemenation-specific data includes pointers to class and method data. The instance variables of the objects are initialized to their default values. The constructor for the most derived class is invoked. The first thing a constructor does is call the consctructor for its superclasses. This process continues until the constrcutor for java.lang.Object is called, as java.lang.Object is the base class for all objects in java. Before the body of the constructor is executed, all instance variable initializers and initialization blocks are executed. Then the body of the constructor is executed. Thus, the constructor for the base class completes first and constructor for the most derived class completes last.
60. **What does the "final" keyword mean in front of a variable? A method? A class?** - FINAL for a variable: value is constant. FINAL for a method: cannot be overridden. FINAL for a class: cannot be derived
61. **What is the difference between instanceof and isInstance?** - instanceof is used to check to see if an object can be cast into a specified type without throwing a cast class exception. isInstance() Determines if the specified Object is assignment-compatible with the object represented by this Class. This method is the dynamic equivalent of the Java language instanceof operator. The method returns true if the specified Object argument is non-null and can be cast to the reference type represented by this Class object without raising a ClassCastException. It returns false otherwise.
62. **Why does it take so much time to access an Applet having Swing Components the first time?** - Because behind every swing component are many Java objects and resources. This takes time to create them in memory. JDK 1.3 from Sun has some improvements which may lead to faster execution of Swing applications.