Advjava

By Mr.Ratan

JDBC

Adv java Syllabus:

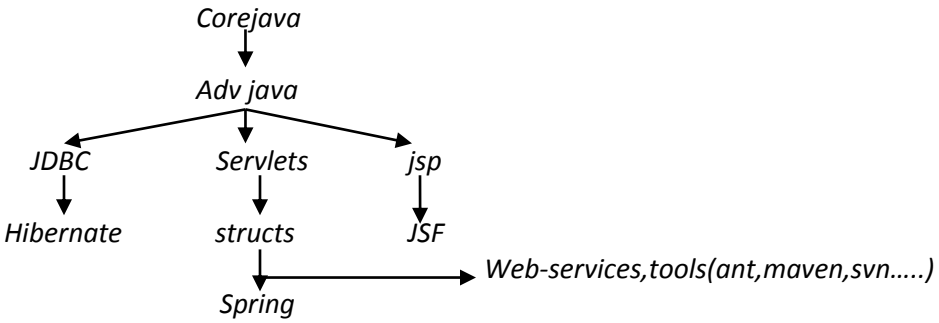
Jdbc	***:***	*java database connectivity*
Servlets		
jsp	***:***	*java server pages*
jstl	***:***	*java standard tag library*

Parts of the java: *Core java & adv java are not official terms.*

*As per the **sun micro system** standard the java language is divided into three types.*

1) *J2SE/JSE(java 2 standard edition)* *corejava,jdbc*
2) *J2EE/JEE(java 2 enterprise edition)* *servlets,Jsp,EJB,JMS,web-services XML,JNDI*
3) *J2ME/JME(java 2 micro edition)*

Learning process of java:



JDBC(java to database connectivity):

1) *The process of connecting the java application to the particular data base is called java to database connectivity.*
2) ***Jdbc is a technology that enables the java program to manipulate the data stored in the data base.***
3) ***Jdbc is a API it provides more interfaces & few classes to provide the connection between java to database.***
4) *Jdbc is designed only for the java applications it doesn't give support for any other technology.*
5) *Jdbc is used to write the persistence logics. The logics which are interacting with database are called persistence logics.*

Database :-*The main purpose of the database is to store the data.*

Ex:- Oracle,MySql……etc

Sun micro systems introduced JDBC as part of the jdk1.1 in 1997& all the jdbc classes & interfaces are present in two packages.

1. *Java.sql*
2. *Javax.sql*

JDB 4.0 (included in Java SE 6).
JDBC 4.1 (included in Java SE 7)
The latest version, JDBC 4.2, included in Java SE 8. Relea0sed in 2013

Technology vs. Framework:

Technologies : J2SE J2EE J2ME
Frameworks : struts, spring , hibernate,jsf…etc

- ✓ *Frameworks are developed based on technologies source code.*
  Jdbc ---> hibernate
  Servlets ---> structs
  Jsp ---> jsf
  EJB's ---> Spring

- ✓ *The framework will support all the features of technologies,*
  - *It supports extra features.*
  - *It overcomes the limitations of technologies.*

- ✓ *The technologies will give good performance when compare to technologies.*

- ✓ *When we develop the application by using technology, the code is duplicated & length increased but when we develop the application by using framework it reduce the code duplication & length of the code because framework having more predefined support.*

- ✓ *The frameworks are flexible to use.*

Different types of applications:-

- ✓ *Standalone applications*
  Developed by using j2se
  Contains main method called by JVM
  Used extension .java .class ***.jar***

- ✓ *Web-applications*
  Developed by using j2ee(servlets & jsp) technology : structs JSF framework
  No-main method contains life cycle methods(init() service() destroy()) called by server
  Used extension ***.war***

- ✓ *Enterprise applications*
  Developed by using EJB technology & spring frame work
  Used extension ***.ear***

- ✓ *Distributed applications*
  Developed by using RMI , web-services v, Micro services
  Used extension ***.aar*** *application archive file*

- ✓ *Mobile applications*
  Developed by using android ,IOS
  Different types of mobile apps
  - *Native apps: Installed & doesn't required internet Conn to run : candy crush*
  - *Hybrid apps : installed & required internet connection to run : facebook*
  - *Web apps : run through web browser*

  Used extension ***.apk****android application package*

ODBC vs JDBC :

Even today C and C++ application are connecting with oracle database using a set of function given by Oracle Corporation in a ***orcl.h*** *header file.*

But problem with the above communication is a front end application become as a database dependent application because every database vendor give its own set of function for communication.

To overcome the database dependent problem ODBC (Open database connectivity) community formed by Microsoft corporation.

ODBC community has provided ODBC API, to connect with any database in a database independent manner.

ODBC:

1. *ODBC is given by Microsoft Corporation it is a database dependent & platform dependent.*
2. *It is written in C-language.*

JDBC :

1. *It is platform & database independent.*
2. *It is written in java.*

JDBC Overview :

JDBC specifications:-

- ✓ *Jdbc specifications are given by Sun micro systems.*
- ✓ *It is a pdf document (JRS java requirement specifications)*

JDBC API:-

- ✓ *API given by Sun Micro Systems.*
- ✓ *It contains more interfaces & few classes.*
- ✓ *These interfaces & classes are present two packages (**java.sql , javax.sql**).*
- ✓ *These two packages are present in **rt.jar** file.*
- ✓ *The jar file location is:C:\Program Files\Java\jre1.8.0_65\lib\rt.jar (extract the jar file check the two packages classes & interfaces).*

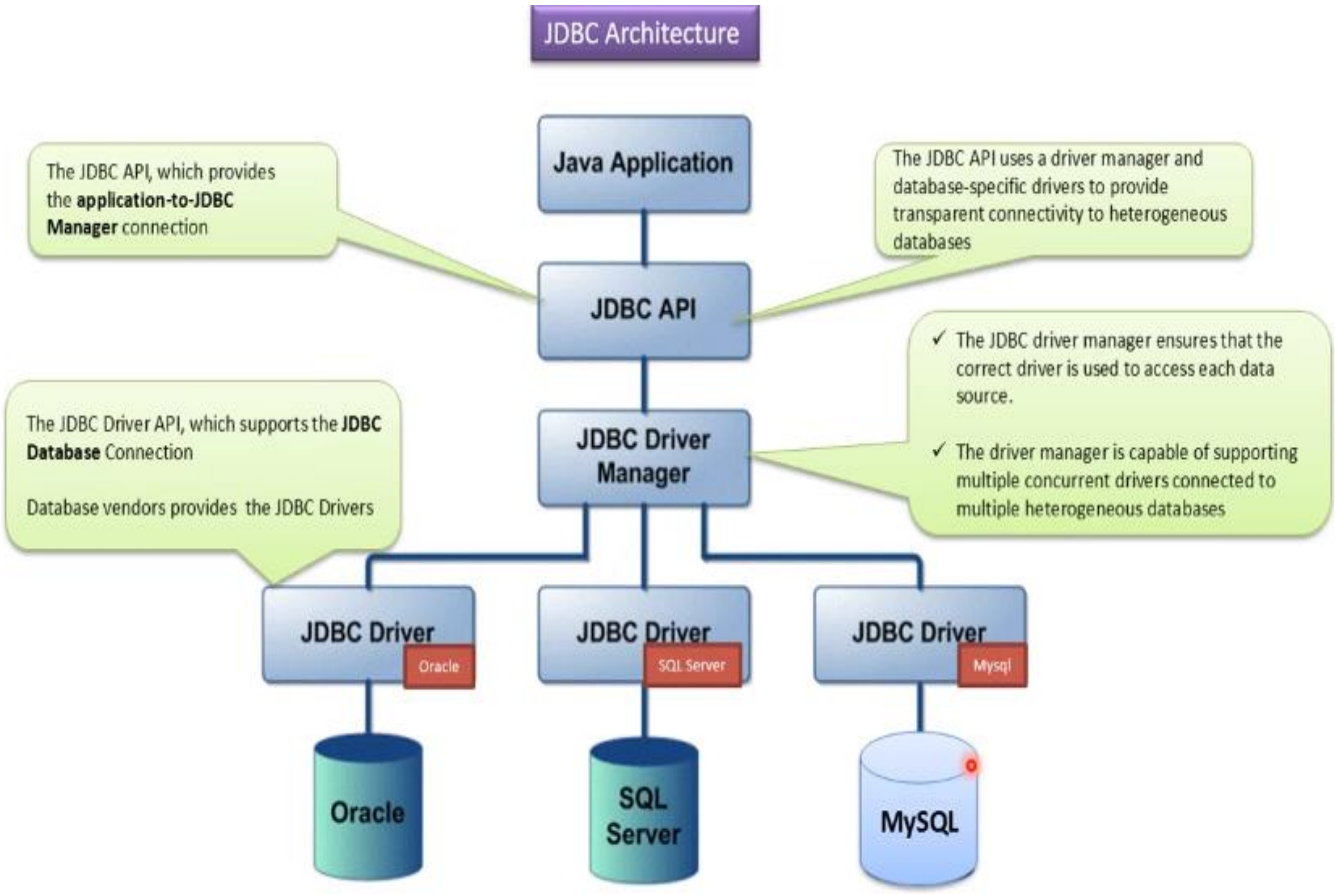
JDBC implementations:-

- ✓ *Implementations are given by database vendors in the form of jar file.*

  Oracle 10g ----> ojddbc14.jar
  Oracle 11g ----> ojdbc 6.jar , ojdbc5.jar
  Oracle 12c ----> ojdbc6.jar, ojdbc7.jar
  MySql ---> Mysql-connector.jar

- ✓ *Extract the jar file we will get all implementations classes .and to check these class file code use java de-compiler software.*

JDBC Architecture Diagram:



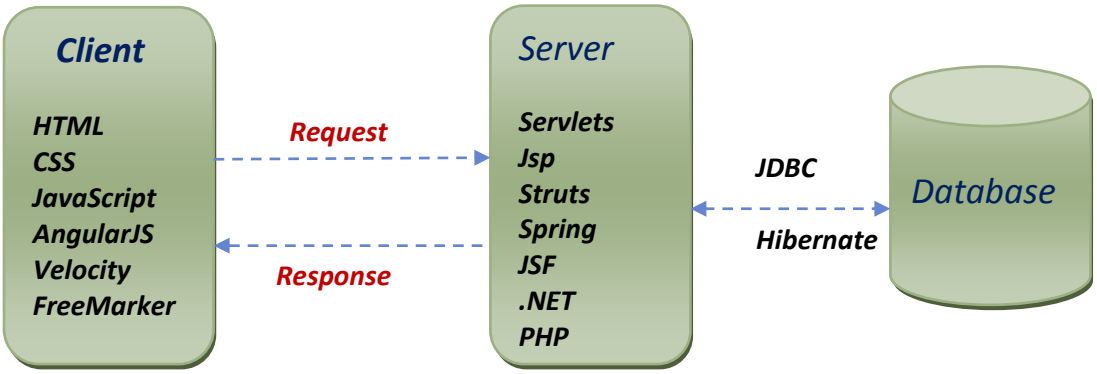
Different type of logics in web application:

1) *Presentation layer.*
2) *Requested data gathering logics.*
3) *Validation logics.*
4) *Business logics/service layer*
5) ***Persistence logics. ----- JDBC , HIBERNATE***
6) *Session management logics.*
7) *Middleware logics.*

Web-application architecture:

The web application contains three layers,

1. *Presentation layer : html, css , js*
2. *Business layer : servlets ,jsp*
3. *Persistence layer : JDBC*



Client: *Who sends the request & takes the response is called client.*
ex: InternetExploral, MozillaFrefox, opera, chrome ……. Etc

Server: *The server contains the applications & it performs fallowing actions.*

a. *It takes the request from the client.*
b. *It identifies the requested resource.*
c. *It processes the request (Request Processing).*
d. *It generates the response based on client request.*

ex: Tomcat,GlassFish,WebLogic,JBOSS,WebSphere …… etc

Database: *Database is used to store the details like client details, application details……etc.*
ex :- Oracle,MySql ….etc

Client side technologies:
The technologies which are used to write the programming at client side are called client side technologies. ex : jsp,html,css,…etc

Server side technologies:
The technologies which are used to provide the programming at server side are called server side technologies. ex: servlets , jsp , structs , spring,Php,.net….etc

J2se:- *Used to develop the standalone applications it contains main method executed by JVM.*
J2EE:- *Used to web applications it does not contains main method, it contains life cycle methods (init() service() destroy()) called by servers.*

There are two types' methods in java

1. *Inline method : declared by user & called by user*

2. *Callback methods: these methods are automatically called by server or JVM or other method..etc*

Main	***--->***	*called by JVM*
Init () service() destroy()	***--->***	*called by server*
Finalize	***--->***	*called by Garbage Collector*
Run () method	***----->***	*called by start()*

Key interfaces of JDBC:

*The below interfaces are present in **java.sql** package.*

1. *Connection*
2. *Statement*
3. *PreparedStatement*
4. *CallableStatement*
5. *ResultSet*
6. *ResultSetMetaData*
7. *DatabaseMetaData*
8. *SavePoint*

*The below interface are present in **javax.sql** package*

9. *RowSet*
10. *PooledConnection*
11. *DataSource*

Note: *All the JDBC main classes and interfaces present in two packages*

java.sql *package*	*:*	*Basic Concepts*
javax.sql	***:***	*Advanced features like connection pooling.*

Prerequisite topics of JDBC:

1. *Exception handling*
2. *Factory method*
3. *Interfaces*
4. *extends vs implements*
5. *flow control statements.*

Basic Information about oracle database:

The present version of oracle is ***Oracle 12c***

Oracle 12c
Oracle 11g
Oracle 10g
Oracel 9i

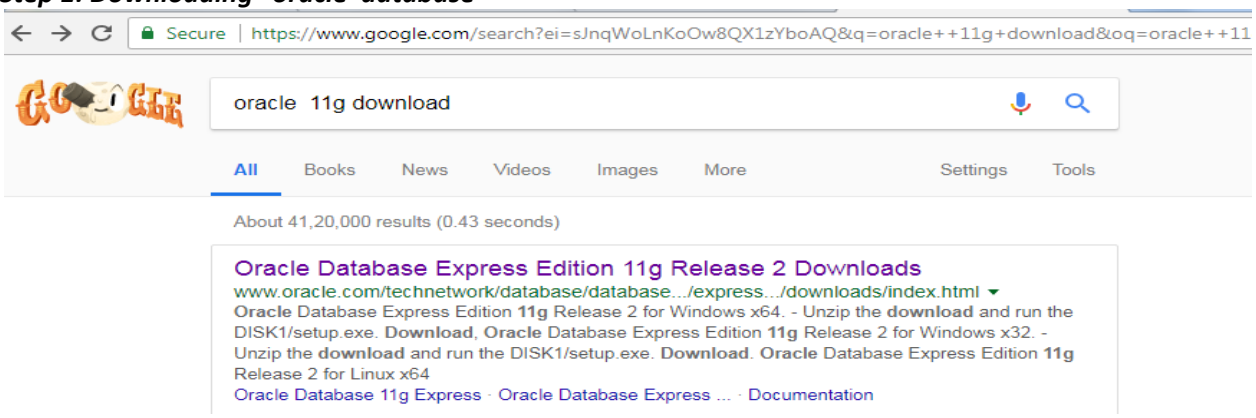
Express edition: *Logical Name :* ***xe***

*Default user name : **system***
*password : **Give at the time of installation // manager***

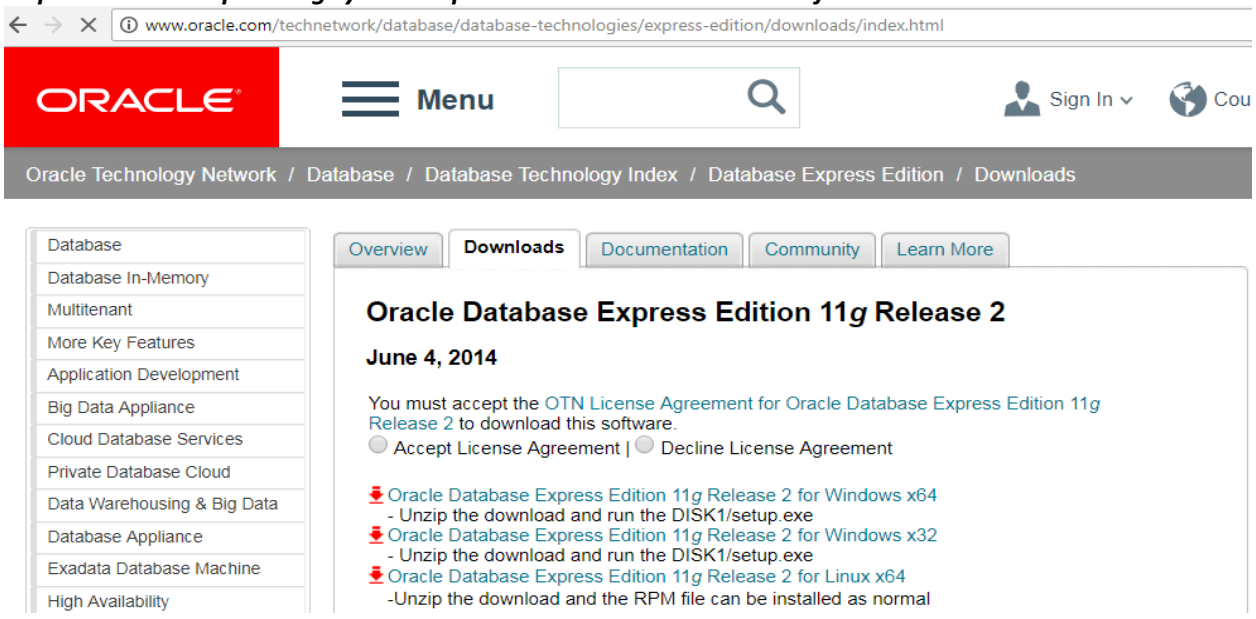
Enterprise edition: *Logical Name :* ***orcl***

*Default user name : **scott***
*Password : **at the time of installation // tiger***

Step 1: Downloading oracle database



Step 2: Based on operating system & process used download the software.



Oracle database installation guide:

Download "Oracle Database Express Edition 11g Release 2 for Windows x64 / Oracle Database Express Edition 11g Release 2 for Windows x32" from below link.
http://www.oracle.com/technetwork/products/express-edition/downloads/index.html#

Registration free: register first then give the username & password.
Username:
Password:

The Oracle Express Edition installation automatically comes with "system" & "sys" accounts. During installation, we have to enter common password for both accounts. The standard password is "manager". However you can give different password.

Check oracle services are started.
Run--> services.msc
OracleServiceXE --"Started" status --> Statup type is "Automatic"
OracleXETNSListener --"Started" status --> Statup type is "Automatic"

Connect to SQL prompt: *in search type runSQL command line we will get below window.*

```
SQL*Plus: Release 10.2.0.1.0 - Production on Wed Sep 7 19:15:22 2016

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> connect
Enter user-name: system
Enter password:
Connected.
SQL>
```

Or

```
SQL*Plus: Release 10.2.0.1.0 - Production on Wed Sep 7 19:16:01 2016

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> conn system/ratan
Connected.
SQL>
```

Frequently used Queries in JDBC:

1) *create table emp(eno number primary key, name varchar2(20),esal number);*
2) *desc emp;*
3) *insert into emp values(100,'rattaiah',25000);*
4) *select * from emp;*
5) *update emp set esal=esal+200 where esal>10000;*
6) *truncate table emp;*
7) *delete from emp where esal>10000;*
8) *drop table emp;*

*core java applications are executed by : **JVM***
*servlets are executed by : **servers***
*The database queries are executed by : **DBE (Database Engine)***

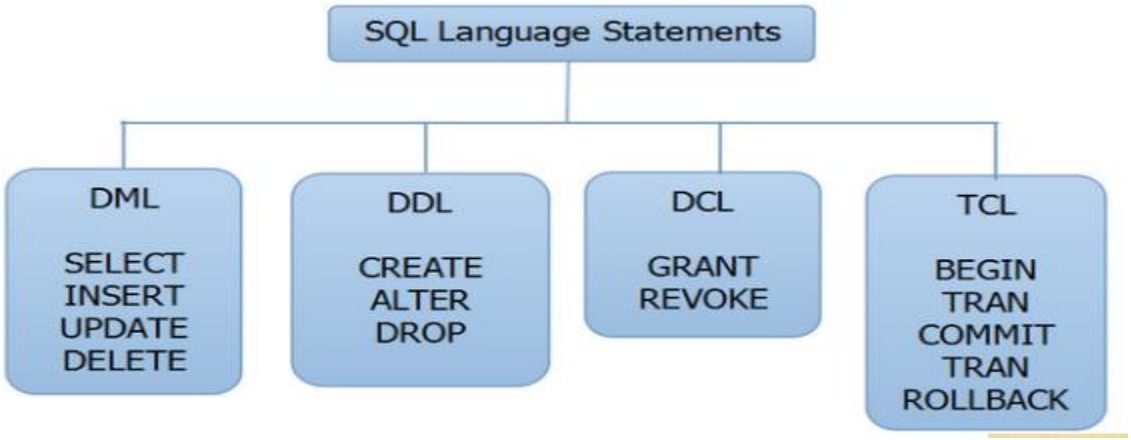
Every query execution the database engine perform fallowing steps.

1) ***Query Tokenization*** : *Divided the query into number of small tokens.*
2) ***Query Parsing*** : *Tokens are arranged in the form of tree structure is called query parsing.*
3) ***Query Optimization:*** *After parsing we can optimize the query to improve execution speed.*
4) ***Query Execution*** : *After optimization we can execute the query and see the output.*

```
SQL> drop table emp;

Table dropped.

SQL> commit;

Commit complete.

SQL> create table emp(eid number,ename varchar2(30),esal number);

Table created.

SQL> insert into emp values(111,'ratan',10000);

1 row created.

SQL> insert into emp values(222,'anu',20000);

1 row created.

SQL> select * from emp;

       EID ENAME                                ESAL
---------- ------------------------------ ----------
       111 ratan                               10000
       222 anu                                 20000

SQL> update emp set esal=esal+100 where esal>5000;

2 rows updated.
```

SQL statements are divided into four categories:

1. DML (Data Manipulation Language)
2. DDL (Data Definition Language)
3. DCL (Data Control Language)
4. TCL (Transaction Control Language)



Information about MySql database:

The default port number: 3306

*Default user name : **root***
*Password : **at the time of installation // root***

Download the software : http://dev.mysql.com/downloads/mysql/
Click on "Archives"
Click on "MySQL Community Server"
Select version: 5.5.32
Select Platform: Microsoft Windows

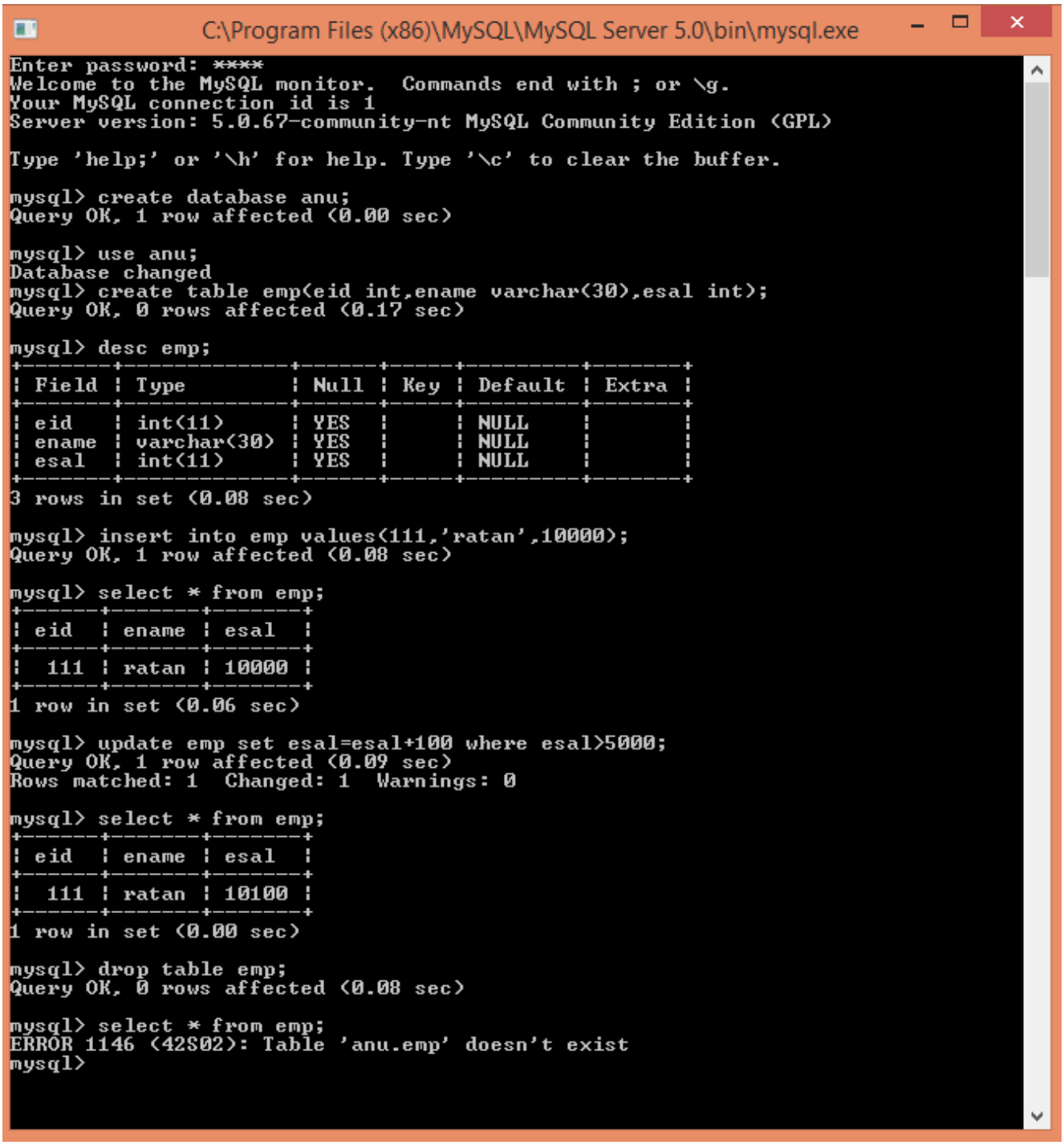
If your system type is 32-bit, then download "Windows (x86, 32-bit), MSI Installer"
If your system type is 64-bit, then download "Windows (x86, 64-bit), MSI Installer"

Register first then enter email & password free registration.
Email:
Password:

Install MYSQL in "Typical" mode.
The 'root' is default account. We have to enter password for root account. The suggested password is 'manager'.

Download "http://mirrors.ibiblio.org/pub/mirrors/maven2/mysql/mysql-connector-java/3.0.10/mysql-connector-java-3.0.10-stable-bin.jar"

Connection details:
driverClassName= com.mysql.jdbc.Driver
url= jdbc:mysql://localhost:3306/ratan
username= root
password= root



mysql> CREATE DATABASE southwind;
Query OK, 1 row affected (0.03 sec)

mysql> DROP DATABASE southwind;
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO products VALUES (1001, 'PEN', 'Pen Red', 5000, 1.23);

*mysql>SELECT * FROM emp;*

mysql> UPDATE emp SET esal = esal+100 where esal>10000;

mysql> drop table emp;

More info about Mysql quires :
https://www.ntu.edu.sg/home/ehchua/programming/sql/MySQL_Beginner.html

To work with mysql database use tool name called : ***SQLYog***

JDBC vs corresponding Java version & jar file version:-

Oracle Database version	JDK Version supported	JDBC specification compliance	JDBC Jar files specific to the release
12.1 or 12cR1	JDK8, JDK 7 & JDK 6	JDBC 4.1 in the JDK 8 & JDK 7 drivers JDBC 4.0 in the JDK 6 drivers	ojdbc7.jar for JDK 8 and JDK 7 ojdbc6.jar for JDK 6
11.2 or 11gR2	JDK 6 & JDK 5 JDK 7 & JDK 8 supported in 11.2.0.3 and 11.2.0.4	JDBC 4.0 in the JDK 6 drivers JDBC 3.0 in the JDK 5 drivers	ojdbc6.jar for JDK 8, JDK 7 and JDK 6. ojdbc5.jar for JDK 5
11.1 or 11gR1	JDK 6 & JDK 5	JDBC 4.0 in the JDK 6 drivers JDBC 3.0 in the JDK 5 drivers	ojdbc6.jar for JDK 6 ojdbc5.jar for JDK 5

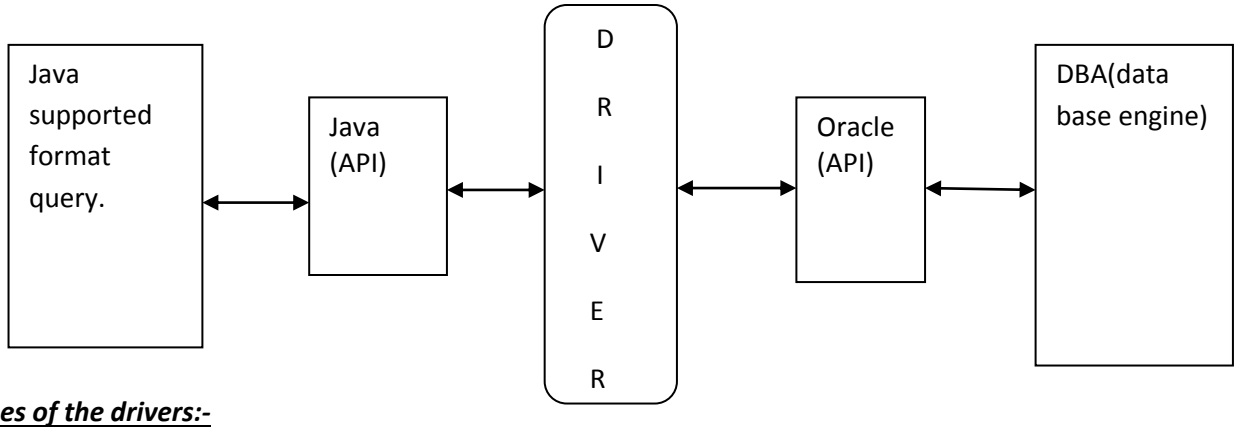
Overview of the JDBC:

1) ***Connection****: it is used to provide the connection between java application to database.*
2) ***Statement*** *: it is used to send the queries to database side nothing but execute the query.*
3) ***PreparedStatement****: it is used to execute the query but number of steps are reduced performance will be increased.*
4) ***CallableStatement*** *: it is used to execute procedures & functions of oracle.*
5) ***ResultSet*** *: it is used to store the data coming from database like table data.*
6) ***ResultSetMetaData*** *: it is used to get metadata of the particular table.*
7) ***DatabaseMetaData*** *: it is used to get metadata of the particular database which we connect.*
8) ***SavePoint*** *: it is used to manage the transactions.*
9) ***RowSet*** *: it is also used to store the data but it will provide more flexibility when compare to resultSet object.*
10) ***DataSource***
11) ***PooledConnection***

The above two interfaces are used to manage the Connection pooling concept.

Driver:-

1) *Jdbc driver is software it enables the java application to interact with database.*
2) *Driver is interface between the java application and the particular database.*
3) *Driver is acting as a translator between the java application and particular database. And it Is converting java formatted SQL queries into the Database supported SQL queries.*



Types of the drivers:-

1) *Type-1 driver* ***(JDBC-ODBC Bridge Driver)***
2) *Type-2 driver* ***(Native API driver)***
3) *Type-3 driver* ***(Network protocol Driver)***
4) *Type-4 driver* ***(Thin Driver)***

DBC API:-

Before JDBC, ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

Drivers in JDBC:-

- *To use the jdbc Driver in java application we must register that driver into Driver manager service.*
- *DriverManager service is built in service to manage set of jdbc drivers.*
  Use the following code to register jdbc driver into DriverManager service.

  Step1:-creating driver Object
  sun.jdbc.odbc.JdbcOdbcDriver obj = new sun.jdbc.odbc.JdbcOdbcDriver();
  step2:registering driver into driver manager service.
  DriverManager.registerDriver(obj);

  The approach of registering driver not required.

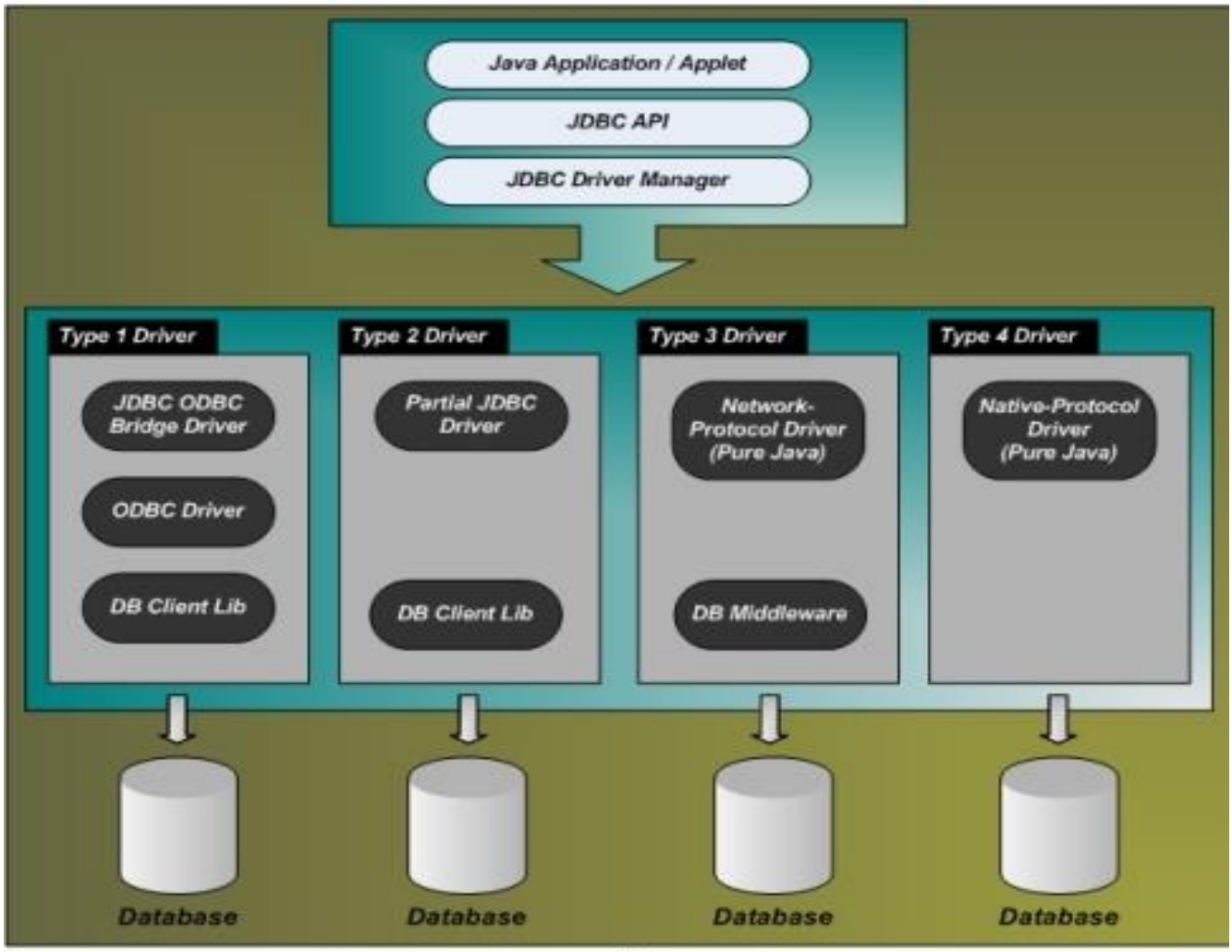
- ✓ *In present application To load the driver class into application we are using fallowing code,*

  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

  The above statements loads driver class and it register driver class into DriverManager service with the help of static block present in JdbcOdbcDriver class.

The Drivers can be divided into the following four types.

1. *Type–1 Driver****JDBC-ODBC bridge driver***
2. *Type–2 Driver* ***Native API, partly Java driver***
3. *Type–3 Driver* ***JDBC-Net, pure Java driver***
4. *Type–4 Driver* ***Thin driver***

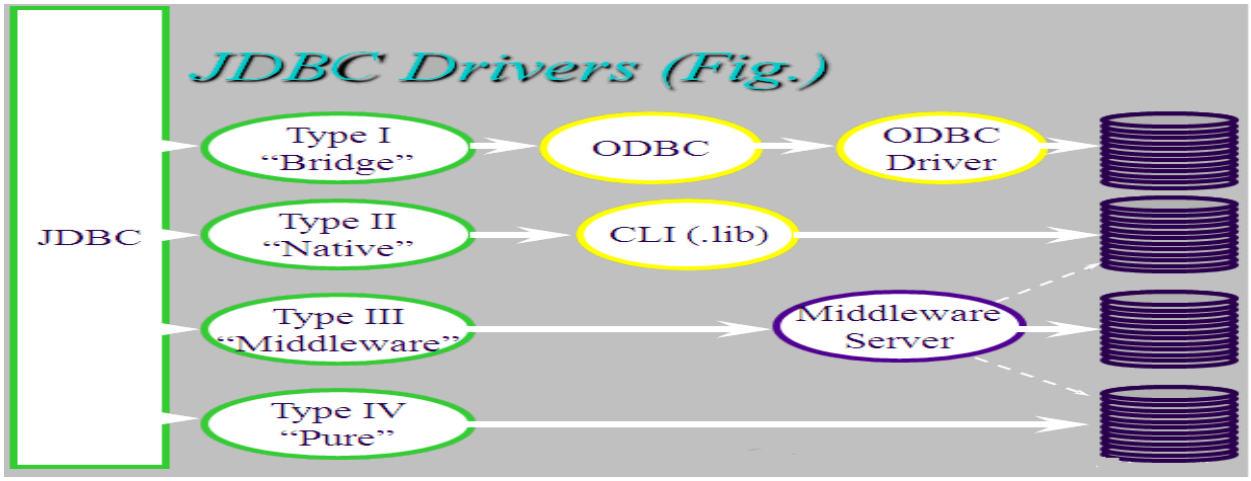


JDBC Architecture

Types of JDBC Drivers:

- *Type 1: JDBC-ODBC Bridge Driver*
- *Type 2: Native-API/Partly Java Driver*
- *Type 3: Network Protocol Driver*
- *Type 4: Native Protocol Driver*

JDBC Architecture: Type Drivers



1. ***Type-1 Driver:***
   - ✓ *This driver the access to database using ODBC driver. And this driver is introduced by sun micro systems with the interdependency on Microsoft product ODBC driver.*

     Advantages :
     - *Type-1 Driver is highly recommended for Standalone applications.*

     Disadvantages :
     - *If we want to use Type-1 Driver in our Jdbc applications then we must install the Microsoft provided ODBC native library in our machine.*
     - *In case of Type-1 Driver, to interact with database we have to perform 2 types of conversions i.e. from **Java to Odbc** and **Odbc to the respective database**.Due to this reason Type-1 driver is slower Driver,it will reduce performance of the Jdbc applications.*
     - *Type-1 Driver is not suitable for web applications.*
     - *Type-1 Driver is suggestible for simple Jdbc applications, but not suggestible for complex Jdbc applications.*

2. ***Type-2 driver:-***
   - ✓ *If we want to use Type-2 Driver in our Jdbc applications then we have to install thedatabase vendor provided Native library.*
   - ✓ *When compared with Type-1 Driver Type-2 Driver is more portable Driver because itshould not require Microsoft provided Native library. When compared with Type-1 Driver Type-2 Driver is faster Driver because it should not require two time conversions.*
   - ✓ *Type-2 Driver is suggestible for only standalone applications, but not suggestible forweb applications.*

3. ***Type-3 driver:-***
   - ***a.*** *Type-3 Driver is also called as **Middleware Database Access Driver** or **NetworkDriver.***
   - *b. Type-3 Driver will provide very good environment to interact with multiple number of databases from multiple clients at a time.*
   - *c. This driver translates the JDBC calls into the middleware vendors protocols, which is then translated to a DBMS protocol by a middleware server. The middleware provides connectivity to many different databases.*

4. ***Type-4 driver:-***
   - ***a.*** *Type-4 Driver is also called as **Thin Driver** or **Pure Java Driver.***
   - ***b.*** *Type-4 Driver was designed purely on the basis of Java technology.*
   - ***c.*** *Type-4 Driver is more portable Driver when compared with Type-1, Type-2 and Type-3 Drivers because Type-4 Driver should not require Odbc Native library, Database vendor provided Native library and Application Server provided Middleware components.*
   - *d. Type-4 Driver is frequently used Driver in Application Development.*
   - *e. Type-4 Driver is recommended for any type of Java, J2EE applications i.e. both standalone applications and enterprise applications.*
   - *f. Type-4 Driver is faster Driver when compared with all the remaining Drivers because Type-4 Driver should not require 2 times conversions in order to interact with database from Java applications.*

Type-1 driver:-

Driver class name ***sun.jdbc.odbc.JdbcOdbcDriver***
Driver URL pattern ***jdbc:odbc:DSNname***
Example : jdbc:odbc:rattan

In the aboveexample we are using DSN name is: ***Ratan***

To create the DSN name we have to fallow the following steps

Start--------→Control Panel------→system and security---→Administrative Tools-----→Data Sources (ODBC)

<---------Click the finish<-----Select a driver<------click add button <------user DSN

Provide DSN name (provide any name)---→Click on ok button------→ok----------→ok

To load and register the driver

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

To provide connection to the particular data base

Connection con=DriverManager.getConnection("jdbc:odbc:first","system","manager");

Type-4 driver:-

Driver class name ***oracle.jdbc.driver.OracleDriver***
Driver URL pattern ***jdbc:oracle:thin:@localhost:1521:xe*** *where xe=logical name of the database*
localhost represent the current machine in the project level using ip address
If we are using express edition the logical name is ***xe***
If we are using enterprise edition the logical name is ***orcl***
It is possible to change the logical name of the database

To load and register the driver

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

To provide connection to the particular database

Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");

jdbc:oracle:thin:@localhost:1521:x

- **jdbc** is the API
- **oracle** is the database
- **thin** is the driver
- **localhost** is the server name on which oracle is running, we may also use IP address
- **1521** is the port number and
- **XE** is the Oracle service name.

Exception handling:- *There are three types of exception in java*

a. *Checked Exception [compiler is able to check] ---->developer mistakes*
b. *Unchecked exception[compiler is unable to check] ---->end user mistakes*
c. *Error [caused due to lack of system resources]*

Example *: In below example we are using throws keyword to delegate the exception but here after exception code is not executed.*

```
class Test
{       void m1() throws ArithmeticException
        {       System.out.println("connection opening="+10/0);
                System.out.println("after exception code database closing");
        }
        public static void main(String[] args) throws ArithmeticException
        {       new Test().m1();
        }
}
```

Example *: In below example method is delegating exception and caller method is handling exception but after exception code is not executed.*

```
class Test
{       void m2() throws ArithmeticException
        {       System.out.println("connection opening="+10/0);
                System.out.println("after exception code database closing");
        }
        void m1()
        {       try{ m2(); }       catch(ArithmeticException ae){ae.printStackTrace();}
        }
        public static void main(String[] args)
        {       new Test().m1();
        }
}
```

<u>Note:</u> In above two examples after exception code is resource releasing (like database closing operations) that code is not executed hence we will get the resource problems.

Example :- *To overcome above limitations use try-catch-finally to release the resources.*

```
class Test
{       void m2()
        {       try{System.out.println(10/0);}
                catch(ArithmeticException ae){ae.printStackTrace();}
                finally{  System.out.println("after exception code ...database closing operations");}
        }
        void m1()
        {       m2();
        }
        public static void main(String[] args)
        {       new Test().m1();
        }
}
```

Steps to design a first application:

Step 1: Load the driver.

- ✓ *Driver is a java class given by database vendors in the form of **jar** file.*

- ✓ *To load the driver .class file byte code into memory user **forName()** method, it is a static method present in Class class hence access this method by using class name.*
  public static java.lang.Class forName(java.lang.String) throws ClassNotFoundException;

- ✓ *forName() method throws ClassNotFoundException and it is checked exception so handle the checked exception by using try-catch blocks or throws keyword.*
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
  ***Jar file location** : C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib*
  oracle 10g -----> ojdbc14.jar
  oracle 11g -----> ojdbc6.jar
  oracle 12c -----> ojdbc7.jar
  mysql -----> mysql-connector.jar

Step 2:- provide the connection between java applications to database.

- ✓ *To get the connection use getConnection() method of DriverManager class.*
  public static java.sql.Connection getConnection(String, String, String) throws SQLException;
  public static java.sql.Connection getConnection(url, username, password) throws SQLException;
  DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");

- ✓ *getConnection() method throws SQLException() and it is a checked exception so must handle that exception by using try-catch blocks or throws keyword.*

***ex :** In below example we are using type-4 driver*

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class TestDb {
        public static void main(String[] args) throws ClassNotFoundException, SQLException
        {       Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection connection =
                DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                System.out.println("connection created successfully.....");
        }
}
```

*In above example we are using type-4 oracle driver & this driver implementation are given database vendors in the in the form of jar file **(ojdbc6.jar)** build path the jar file. If we are using command prompt we have to set the class-path to jar file.*

Step 3: write the SQL query.

String q1= "create table emp(eno number primary key,ename varchar2(10),esal number)";

Step 4: execute the query by using statement object.

To execute the queries, Statement object is providing three methods,

1) executeUpdate(_) method:

In Jdbc applications, executeUpdate(_) method can be used to execute updation group of SQL queries like create, insert, update,delete, drop ..etc & it returns int value as return value.

public int executeUpdate(String sql_query)throws SQLException

Type-1		***Type-4***
Create	*---> -1*	*Create--->0*
Drop	*---> -1*	*Drop--->0*
Insert	*---> 1*	*Insert--->1*
update	*---> 5 (updated rows)*	*Update ->5 (updated rows)*

2) executeQuery( ) method:

a. *executeQuery( ) method can be used to execute selection group SQL queries in order to retrieve the data from database.*
b. *To store the data which is coming from database we are using ResultSet object.*

public ResultSet executeQuery(String sql_query)throws SQLException

ex : *ResultSet rs=st.executeQuery("select * from emp");*

3) execute() method:

- *execute( ) method can be used to execute both selection group SQLqueries and updation group SQL queries.*

public boolean execute(String sql_query)throws SQLException

boolean b2=statement.execute("update emp1 set esal=esal+500 where esal<10000");
*boolean b1=statement.execute("select * from emp1");*

Step 5: Release the resources

To close the resources use close() method.

public abstract void close() throws java.sql.SQLException;

executeQuery()	executeUpdate()	execute()
This method is used to execute the SQL statements which retrieve some data from the database.	This method is used to execute the SQL statements which update or modify the database.	This method can be used for any kind of SQL statements.
This method returns a ResultSet object which contains the results returned by the query.	This method returns an int value which represents the number of rows affected by the query. This value will be the 0 for the statements which return nothing.	This method returns a boolean value. TRUE indicates that query returned a ResultSet object and FALSE indicates that query returned an int value or returned nothing.
This method is used to execute only select queries.	This method is used to execute only non-select queries.	This method can be used for both select and non-select queries.
Ex : SELECT	Ex : DML → INSERT, UPDATE and DELETE DDL → CREATE, ALTER	This method can be used for any type of SQL statements.

Build path vs class path :

- ✓ *The build path is working only for eclipse IDE to configure the jar file.*
- ✓ *The class path is working only for command to run the application.*
- ✓ *The build path is not working for command prompt to run & class path environmental variables are not working to run the applications in eclipse.*

Three ways to load the jar file:-

The oracle jar file location is

C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

1-Way to load jar file : *paste the* ***ojdbc14.jar*** *file in jre/lib/ext folder*

2-way to load jar file

set classpath:

There are two ways to set the classpath:

temporary

permanent

How to set the temporary classpath:-

search the ojdbc14.jar file then open command prompt and write:

C:>set classpath=c:\folder\ojdbc14.jar;.;

How to set the permanent classpath:

C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

Environment variables setup :

d. From the desktop, right click the Computer icon.

e. Choose Properties from the context menu.

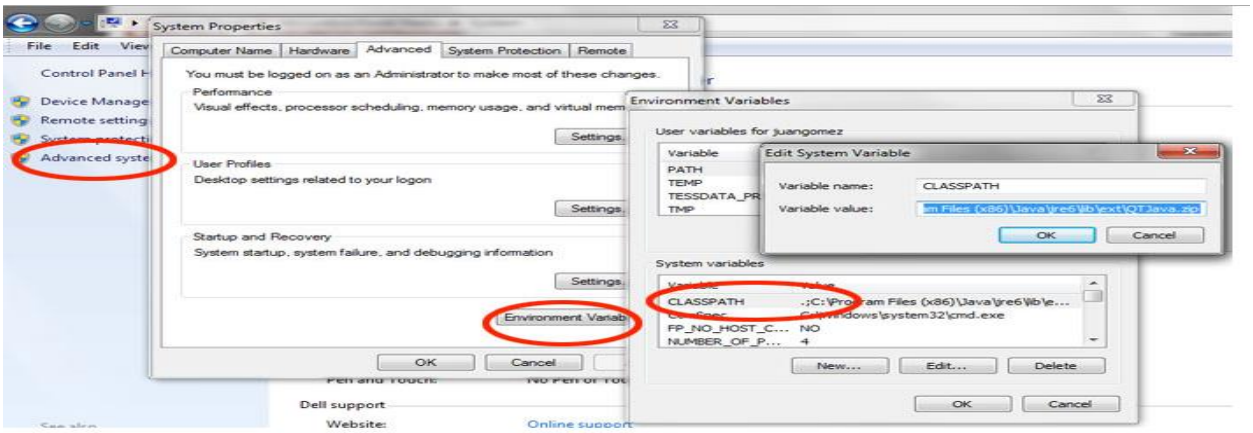
f. Click the Advanced system settings link.

g. Click Environment Variables. In the section user Variables, click new In

variable name : classpath

variable value : paste the path to ojdbc14.jar.

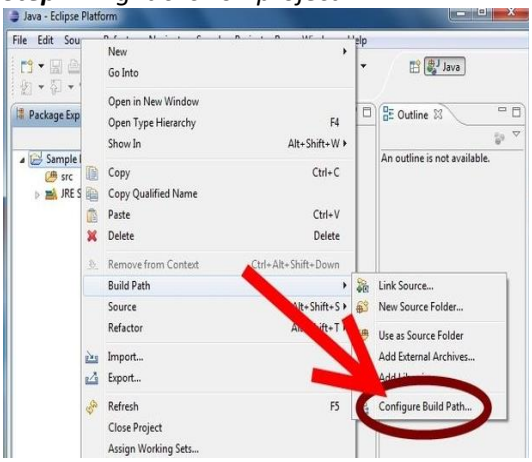
h. Click OK. Close all remaining windows by clicking OK.



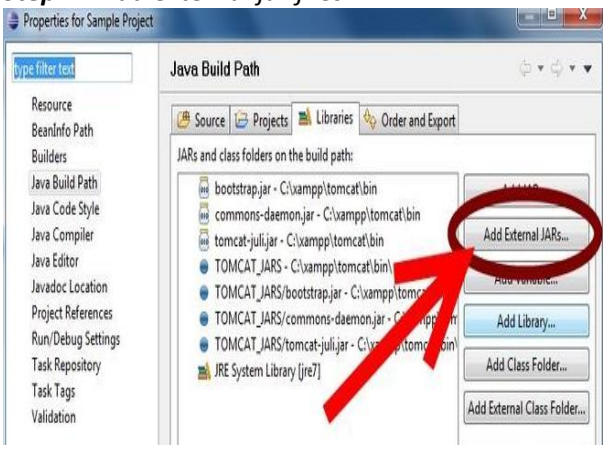
The above 2-ways are not recommended because in real-time projects we are using eclipse IDE to develop the application so we will use build path to set the jar file.

3- way to load the jar file : ***Build path***

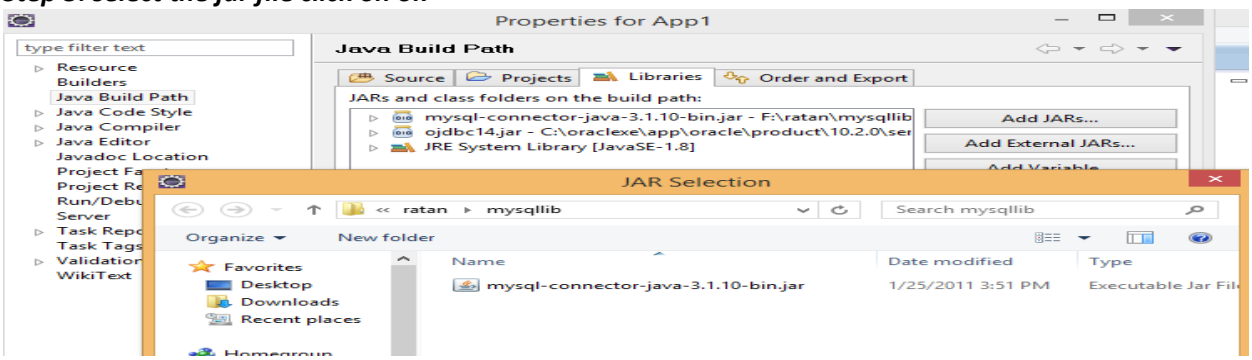
Step 1: *Right click on project*



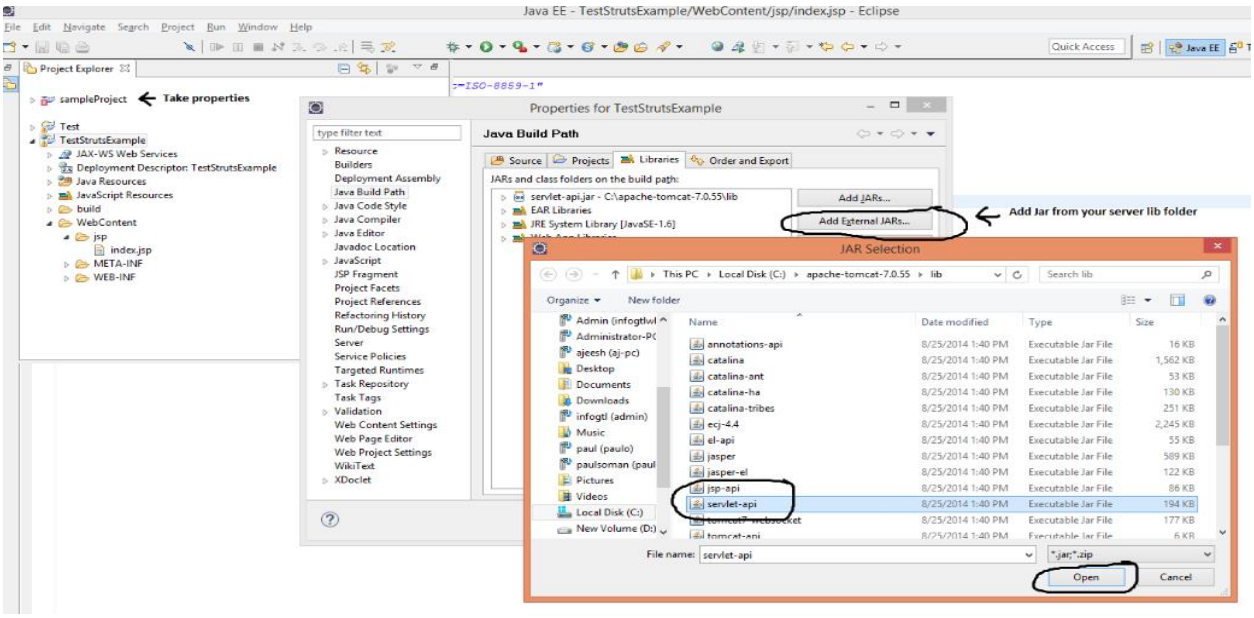
Step 2: *Add external jar files*



Step 3: select the jar file click on ok



All steps in single window:


Java EE - TestStrutsExample/WebContent/jsp/index.jsp - Eclipse
Take properties
Properties for TestStrutsExample
Java Build Path
Add JARs...
Add External JARs...
Add Jar from your server lib folder
JAR Selection
servlet-api
Open
Cancel

App -1: Application with throws keyword : Table creation process

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException {
		//step 1: load the driver
		Class.forName("oracle.jdbc.driver.OracleDriver");

		//step 2: create the connection
		Connection connection =
		DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");

		//step 3: write the query
		String q1="create table emp(eid number primary key,ename varchar2(30),esal number)";

		//step 4: process the query
		Statement statement = connection.createStatement();
		int x = statement.executeUpdate(q1);
		System.out.println("table created successfully..."+x);

		//step 5: relaese the resources
		statement.close();
		connection.close();
		System.out.println("Connection closed successfully.....");
	}
}
```

- ✓ *In above example we have to build path the jar file based on which version of database we are using.*
- ✓ *In above example we are using throws, in this case when exception raised program terminated abnormally there may be chance of resources are not released.*
- ✓ *To overcome above problem to release the resources normally use the **finally** block this code always executed to release the resources.*

Note: *In application level when we have resource releasing option it is recommended to use finally block.*

App 2: Application with try-catch-finally blocks : Data insertion process

Step 1: Declare the resources.
Step 2: Write the application logics in try block.
Step 3: in catch block handle the exception.
Step 5: Release the resources by using finally block.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class TestDb {
	public static void main(String[] args) {
		//Resource declaration
		Connection connection=null;
		Statement statement=null;
		try {
			Class.forName("oracle.jdbc.driver.OracleDriver");
    connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
			System.out.println(connection);
			System.out.println("Connection created successfully");

			String q1="insert into emp values(111,'ratan',10000)";
			String q2="insert into emp values(222,'durga',20000)";
			String q3="insert into emp values(333,'anu',30000)";
			statement = connection.createStatement();
			statement.executeUpdate(q1);
			statement.executeUpdate(q2);
			statement.executeUpdate(q3);
			System.out.println("Values are inserted successfully.....");
		}
		catch(ClassNotFoundException|SQLException e)
		{	e.printStackTrace();
		}
		finally {
			try {	if(statement!=null)statement.close();
				if(connection!=null)connection.close();
			}
			catch (SQLException e) {e.printStackTrace(); }
		}
		System.out.println("Connection Closed Successfully.....");
	}
}
```

- ✓ *Release the resources by using finally block because this code is executed always in normal & abnormal terminations.*
- ✓ *Before releasing the resources check resources are available or not.*

App 3: try with resources: java7 version : Data retrieval process.

- ✓ *When we declare the resources by using try block once the try block is completed, the resources are automatically released.*
- ✓ *If the resource throws checked exception catch block mandatory.(in below example resource throws checked exceptions so catch block mandatory)*
- ✓ *If the resource throws unchecked exception catch block is optional.*

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class TestDb {
        public static void main(String[] args) {
                try(Connection connection =
                DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                                Statement statement = connection.createStatement();)
                {
                        String q = "select * from emp";
                        ResultSet set =  statement.executeQuery(q);
                        while(set.next())
                        {
                        System.out.println(set.getInt(1)+" "+set.getString("ename")+" "+set.getFloat(3));
                        }
                        System.out.println("values are retrieved successfully…. ");
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
                System.out.println("Connection closed Successfully.....");
        }
}
```

Different types of Drivers:

Type -1 Driver :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("jdbc:odbc:first","system","system");
```

Type -3 Driver :

```
Class.forName("ids.sql.IDSServer");
Connection con=DriverManager.getConnection("jdbc:ids://localhost:12/conn?dsn='accdsn'");
```

Type -4 Driver :

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ram");
```

Type -5 Driver :

```
Class.forName("com.ddtek:jdbc.oracle.OracleDriver");
Connectioncon = DriverManager.getConnection
                ("jdbc:datadirect:oracle://localhost:1521:servicename="orcl","system","ratan");
```

Application- 4 : Application with try-catch block. CURD operations.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class TestDb2 {
	public static void main(String[] args) {
		//Resource Declaration
		Connection connection=null;
		Statement statement=null;
		ResultSet set=null;

		try
		{	Class.forName("oracle.jdbc.driver.OracleDriver");
	connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
			System.out.println("table created successfully....="+a);

			statement = connection.createStatement();
		String q1="create table emp(eid number primary key,ename varchar2(30),esal number)";
			int a = statement.executeUpdate(q1);
			System.out.println("table created successfully....="+a);

			String q2="insert into emp values(111,'ratan',10000)";
			String q3="insert into emp values(222,'anu',20000)";
			String q4="insert into emp values(333,'durga',30000)";
			statement.executeUpdate(q2);
			statement.executeUpdate(q3);
			statement.executeUpdate(q4);
			System.out.println("insertion process completed.....");

			String q5="select * from emp";
			 set = statement.executeQuery(q5);
			while(set.next())
			{
		System.out.println(set.getInt(1)+"---"+set.getString("ename")+"--"+set.getDouble(3));
			}
			System.out.println("Data retrieved completed.....");

			String q6="update emp set esal=esal+100 where esal>10000";
			int b = statement.executeUpdate(q6);
			System.out.println("updated records="+b);

			Thread.sleep(20000);//20 sec
```

```
                String q7="drop table emp";
                int c = statement.executeUpdate(q7);
                System.out.println("table droped successfully....="+c);
            }
            catch(ClassNotFoundException | SQLException | InterruptedException e)
            {       System.out.println("exception raised...="+e);
            }
            finally
            {       try{ if(set!=null) set.close();}
                    catch(SQLException e){e.printStackTrace();}

                    try{  if(statement!=null) statement.close();}
                    catch(SQLException e){e.printStackTrace();}

                    try{  if(connection!=null) connection.close();}
                    catch(SQLException e){e.printStackTrace();}
            }
            System.out.println("Connection closed successfully.....");
        }
}
```

Application 5 : *In below example we are using type-1 driver hence we have to set DSN name.*

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class Test {
        public static void main(String[] args)
                throws ClassNotFoundException, SQLException,InterruptedException {

        System.out.println("*****connection creation process******");
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection connection=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
                Statement statement=connection.createStatement();

                System.out.println("*****Table creation process******");
                String query1="create table emp(eid number,ename varchar2(24),esal number)";
                int a=statement.executeUpdate(query1);
                System.out.println("table is create successfully="+a);

                System.out.println("**********Table dropping process**********");
                String query7="drop table emp11";
                Int b = statement.executeUpdate(query7);
                System.out.println("table dropped successfully="+b);
        }
}
```

Output:- if we are using type-1 driver (we have to set DSN name use dsn name)

******connection creation process*******
******table creation process*******
table is create successfully = -1
table dropped successfully = -1

Output:- if we are using type-1 driver (we have to set DSN name use dsn name)

******connection creation process*******
******table creation process*******
table is create successfully = 0
table dropped successfully = 0

Application-6: Check the application with execute() method

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class TestDb
{		public static void main(String[] args) throws ClassNotFoundException, SQLException
		{	Class.forName("oracle.jdbc.driver.OracleDriver");
			Connection connection =DriverManager.getConnection
					("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
			System.out.println("Connection created successfully.....");

			Statement statement = connection.createStatement();

			String q1="select * from emp";
			String q2="update emp set esal=esal+100 where esal>500";

			boolean status = statement.execute(q2);		//line-1
			if(status)
			{	System.out.println("Selection statement....");
				ResultSet set =  statement.getResultSet();
				while(set.next())
				{	System.out.println(set.getInt(1)+"---"+set.getString(2)+
									"--"+set.getDouble(3))
				}
			}
			else
			{	int a = statement.getUpdateCount();
				System.out.println("records are updated="+a);
			}

			Set.close();
			statement.close();
			connection.close();
			System.out.println("connection is closed");
		}
}
```

In the line-1 pass the updating group sql statement check the output.
In the line-1 pass the selection group sql statement check the output.

Java.util.Scanner : Taking input from end-user

*Scanner class present in **java.util** package and it is introduced in 1.5 versions & it is used to take dynamic input from the end-user.*

to get int value	*---->*	*s.nextInt()*
to get float value	*---->*	*s.nextFloat()*
to get String value	*---->*	*s.next()*
to get single line	*---->*	*s.nextLine()*
to close the input stream	*---->*	*s.close()*

```
import java.util.*;
class Test
{       public static void main(String[] args)
        {       Scanner s=new Scanner(System.in);

                System.out.println("enter emp no");
                int eno=s.nextInt();

                System.out.println("enter emp name");
                String ename=s.next();

                System.out.println("enter emp salary");
                float esal=s.nextFloat();

                System.out.println("*****emp details*****");
                System.out.println("emp no----->"+eno);
                System.out.println("emp name---->"+ename);
                System.out.println("emp sal------>"+esal);

                s.close();
        }
}
```

ex : *The \s represents whitespace.*

```
import java.util.*;
public class Test
{               public static void main(String args[])
                {       String input = "java 10 version";

                        Scanner s = new Scanner(input).useDelimiter("\\s");
                        System.out.println(s.next());
                        System.out.println(s.nextInt());
                        System.out.println(s.next());

                        s.close();
                }
}
```

Example :- ***Application taking dynamic input from the keyboard by using scanner class.***

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;

public class TestDb {
	public static void main(String[] args)
				throws ClassNotFoundException, SQLException, InterruptedException   {

		Class.forName("oracle.jdbc.driver.OracleDriver");
				Connection connection =
		DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

		Statement statement = connection.createStatement();
		Scanner scanner = new Scanner(System.in);

		while(true)
		{	System.out.println("enter emp id");
			int eid = scanner.nextInt();

			System.out.println("enter emp name");
			String ename = scanner.next();

			System.out.println("enter emp sal");
			double esal = scanner.nextDouble();

			String q2="insert into emp values("+eid+",'"+ename+"',"+esal+")";
			System.out.println(q2);

			statement.executeUpdate(q2);
		System.out.println("values are inserted successfully do you want one more(yes/no)");
			String option = scanner.next();
			if(option.equals("no"))
			{  break;
			}
		}
		connection.close();
		scanne.close();
		System.out.println("connection  closed successfully.....");
	}
}
```

Java.sql.PreparedStaement:

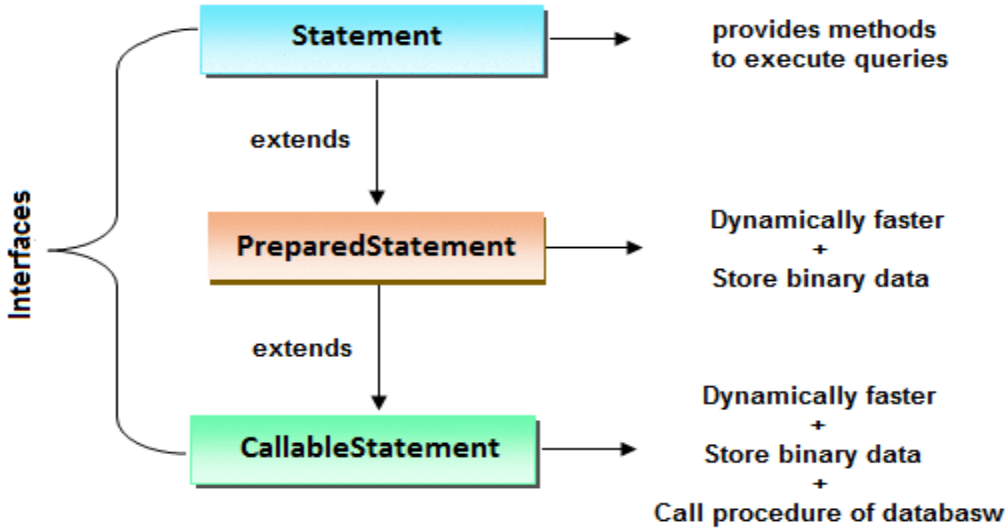
1) *It is sub interface of statement interface used to execute parameterized query.*

2) *Whenever we are executing the query the database side the fallowing operations will be performed.*
   a. *Query tokenization*
   b. *Query parsing*
   c. *Query optimization*
   d. *Query execution*

3) *If we using Statement interface to execute query every time the above operations (query compilation) are performed it effects on performance of the application.*

4) *To overcome the above limitation use preparedstatement. When we use the preparedstatement the query compilation done only once it improves the performance of the application.*

5) *In case of preparedstatement the query compilation done only once then DBE prepare query plan with unique id, later just set the data execute the query it improves performance of the application.*



*Note : if we want to execute the query only once simple use statement object but if we want execute the same query **'n'** number of times use prepared statement.*

Step 1: Create the Preparedstatement object.

public PreparedStatement prepareStatement(String query)throws SQLException

ex: PreparedStatement pst=con.prepareStatement("insert into emp values(?,?,?)");

- *In PreparedStatement only '?' symbol are allow, no other symbols are allowed.*
- *'?' is only for replacing value but not for table name or column names.*
- *'?' symbol are not allowed in DDL operation.*
- *'?' symbol is called parameter or replacement operator or place-resolution operator.*
- *Its value will be set by calling the setter methods of PreparedStatement.*

Step 2: Set values to the positional parameters

To set values to the positional parameters we will use setter method of PreparedStatement.
public void setXxx(int param_index, xxx value)

```
preparedStatement.setInt(1, 111);
preparedStatement.setString(2, "ratan");
preparedStatement.setDouble(3, 10000.45);

preparedStatement.setInt(1, eid);
preparedStatement.setString(2, ename);
preparedStatement.setDouble(3, esal);
```

Step 3: execute the query
preparedStatement.executeUpdate();

step 4: release the resource.
preparedStatement.close();

Application-1: PreparedStatement performs insert operations

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Scanner;

public class TestDb3 {
	public static void main(String[] args) throws ClassNotFoundException, SQLException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection = DriverManager.getConnection
					("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

 PreparedStatement preparedStatement = connection.prepareStatement("insert into emp values(?,?,?)");
		Scanner scanner = new Scanner(System.in);

		while(true)
		{	System.out.println("enter emp id");
			int eid = scanner.nextInt();

			System.out.println("enter emp name");
			String ename = scanner.next();

			System.out.println("enter emp sal");
			double esal = scanner.nextDouble();

			//setting the values to preparedstatement parameterized query
			preparedStatement.setInt(1, eid);
			preparedStatement.setString(2, ename);
			preparedStatement.setDouble(3, esal);
			preparedStatement.executeUpdate();

			System.out.println("values are inserted sucessfully....");
			System.out.println("Do you want one more record...(yes/no)");
			String option = scanner.next();
			if(option.equals("no"))
				break;
		}

		preparedStatement.close();
		scanner.close();
		connection.close();
		System.out.println("connection closed successfully....");
	}
}
```

Application-2 : PreparedStatement with updating query.

```
PreparedStatement preparedStatement =
        connection.prepareStatement("update emp set esal=esal+? where esal>?");

    preparedStatement.setInt(1, 100);
    preparedStatement.setInt(2, 2000);

    int a = preparedStatement.executeUpdate();
    System.out.println(a);
```

Application-3 :- PreparedStatement with select query.

```
    PreparedStatement preparedStatement =
        connection.prepareStatement("select * from emp where esal>?");

    preparedStatement.setInt(1, 2000);

    ResultSet set = preparedStatement.executeQuery();
    while(set.next())
    {       System.out.println(set.getInt(1));
    }
```

Example:-working with MySql data base

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("com.mysql.jdbc.Driver");
		Connection connection =
		DriverManager.getConnection("jdbc:mysql://localhost:3306/vishnu","root","root");
		System.out.println("connection created successfully with mysql.....");

		Statement statement = connection.createStatement();

		String q1="select * from emp";
		ResultSet set =  statement.executeQuery(q1);

		while(set.next())
		{	System.out.println(set.getInt(1)+"--"+set.getString(2)+"--"+set.getDouble(3));
		}
		System.out.println("operations are completed");
	}
}
```

In above we are using mysql driver class so this driver class implementation given by mysql database vendor in the form of the jar file

Mysql-connector.jar

In eclipse ide must set the build path configurations otherwise in project folder create the lib folder manually and copy the jar file and paste the jar file in lib folder the refresh the project.

The jar file location is : https://dev.mysql.com/downloads/connector/j/ click download button.

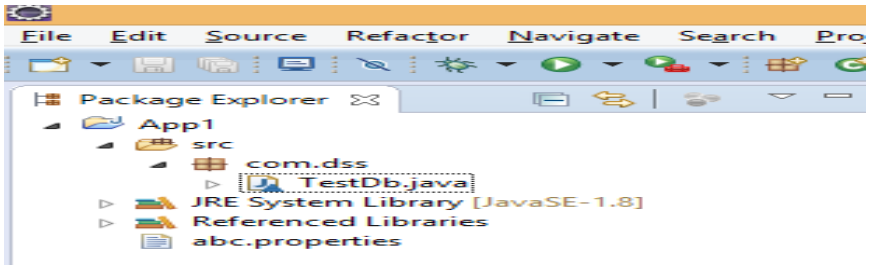
Example:Reading data from properties file.

In JDBC application the JDBC properties like username,password,driver..etc the properties are frequently changing, for every change doing modification in every source file is not recommended.

To overcome above problem to get the flexibility of modifications use properties file. when we do the modifications on properties file those modifications are reflected in entire project.

Working with properties file:-

- ✓ *To get the flexibility of modifications use properties file.*
- ✓ *Property file is a normal text file it recognize the data in the form of key=value pairs.*
- ✓ *Properties file always read from secondary memory.*



Create the properties file in inside the project folder directly.

abc.properties

```
username=system
password=ratan
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:xe
```

Test.java

```
package com.dss;
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
class TestDb
{               public static void main(String[] args)
                                throws ClassNotFoundException, SQLException, IOException
                {       FileInputStream inputStream = new FileInputStream("abc.properties");
                        Properties properties = new Properties();
                        properties.load(inputStream);

                        Class.forName(properties.getProperty("driver"));
Connection connection =DriverManager.getConnection(properties.getProperty("url"),
                properties.getProperty("username"), properties.getProperty("password")););
                        System.out.println("Connection created successfully.....");
                        connection.close();
                        System.out.println("connection is closed");
                }
}
```

Java.sql.ResultSet:-

1. *ResultSet is an interface present in the java.sql package and it is used to hold the values which are coming from data base.*
2. *When we create the ResultSet object the resultset cursor is pointing to before first record.*
3. *The default ResultSet object is CONCUR_READ_ONLY and TYPE_FORWARD_ONLY .*

ResultSet's Type:-

Forward cursor:-

ResultSet.TYPE_FORWARD_ONLY (default):

By using this ResultSet it is possible to moves forward only from the first row to the last row via the next() method.

Scrollable cursor:-

ResultSet.TYPE_SCROLL_SENSITIVE:

The ResultSet reflects changes made by others to the underlying data source while it remains opened.

ResultSet.TYPE_SCROLL_INSENSITIVE:

The ResultSet does NOT reflect change made by others to the underlying data source while it is opened, hence, insensitive to changes.

ResultSet's Concurrency:-

ResultSet.CONCUR_READ_ONLY (default):

The ResultSet is read-only. You cannot update the underlying database via the ResultSet.

ResultSet.CONCUR_UPDATABLE:

The ResultSet object can be updated via the updateXxx() methods. Change in ResultSet is reflected in the underlying database.

While creating statement object we have to specify the result set type. And type is the first argument & concurrency is the second argument.
Type & concurrency is the static constants of ResultSet so access the static constants by using ResultSet.

Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

Example:-The default ResultSet object is forward only and read only.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521","system","ratan");
		System.out.println("connection created successfully with Oracle.....");

		Statement statement = connection.createStatement();

		String q1="select * from emp";
		ResultSet set =  statement.executeQuery(q1);
		while(set.previous())
		{	System.out.println(set.getInt(1)+"--"+set.getString(2)+"--"+set.getDouble(3));
		}

		System.out.println("operations are completed");
		  }
	}
```

The default ResultSet object is forward only and read only hence it is not possible to perform previous() method we will get the exception.

Exception in thread "main" java.sql.SQLException: Invalid operation for forward only resultset : previous

But the same application if we are using Mysql the output will be printed it means the behavior of the ResultSet object is changed from database to database but the generic way it is always recommended to set the type & concurrency based on the operation we are performing.

ResultSet Cursor methods:- *when we create ResultSet object is first created, the cursor is positioned before the first row*.

Next():-

Moves the cursor forward one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned after the last row.

public abstract boolean next() throws java.sql.SQLException;

Previous():

Moves the cursor backward one row. Returns true if the cursor is now positioned on a row and false if the cursor is positioned before the first row.

public abstract boolean previous() throws java.sql.SQLException;

First():-

Moves the cursor to the first row in the ResultSet object. Returns true if the cursor is now positioned on the first row and false if theResultSet object does not contain any rows.

public abstract boolean first() throws java.sql.SQLException;

Last():-

Moves the cursor to the last row in the ResultSet object. Returns true if the cursor is now positioned on the last row and false if the ResultSet object does not contain any rows.

public abstract boolean last() throws java.sql.SQLException;

beforeFirst():-

Positions the cursor at the start of the ResultSet object, before the first row. If the ResultSet object does not contain any rows, this method has no effect.

public abstract void beforeFirst() throws java.sql.SQLException;

afterLast():

Positions the cursor at the end of the ResultSet object, after the last row. If the ResultSet object does not contain any rows, this method has no effect.

public abstract void afterLast() throws java.sql.SQLException;

absolute(int row):- Positions the cursor on the row specified by the parameter row.

Example :- ResultSet object basic operations.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

		String q1="select * from emp";
		ResultSet set =  statement.executeQuery(q1);

		set.afterLast();
		while(set.previous())
		{	System.out.println(set.getInt(1)+"--"+set.getString(2)+"--"+set.getDouble(3));
		}

		set.first();
		System.out.println(set.getInt(1));

		set.last();
		System.out.println(set.getInt(1));

		set.absolute(3);
		System.out.println(set.getInt(1));

		System.out.println("operations are completed");
	}
}
```

Example :-ResultSet object performing updation operations.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestDb {
        public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
                Class.forName("com.mysql.jdbc.Driver");
                Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/vishnu","root","root");
                System.out.println("connection created successfully with mysql.....");

Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

                String q1="select * from emp";
                ResultSet set =  statement.executeQuery(q1);

                set.absolute(2);            //moving cursor to particular record to perform updation
                set.updateInt(1, 22222);
                set.updateRow();


                System.out.println("operations are completed");
        }
}
```

Example : Inserting data into ResultSet by using moveToInsertRow method :-

To insert column values into the insert row. An updatable ResultSet object has a special row associated with it that serves as a staging area for building a row to be inserted. The following code fragment moves the cursor to the insert row, builds a three-column row, and inserts it into resultsets and into the data source table using the method insertRow.

```
                String q1="select * from emp";
                ResultSet set =  statement.executeQuery(q1);

                set.moveToInsertRow();
                set.updateInt       (1, 55);
                set.updateString    (2, "ratan");
                set.updateFloat  (3,2000);
                set.insertRow();

                set.beforeFirst();
```

Example : Java.sql.ResultSetMetaData

It is used to get the metadata of the particular table like

a. *Number of column in table*
b. *Data type of the column*
c. *Size of the column…etc*

To get ResultSetMetaData object use getMetaData method of ResultSet object,

public abstract java.sql.ResultSetMetaData getMetaData() throws java.sql.SQLException;

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

import com.mysql.jdbc.ResultSetMetaData;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("com.mysql.jdbc.Driver");
		Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/vishnu","root","root");
		System.out.println("connection created successfully with mysql.....");

Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

		String q1="select * from emp";
		ResultSet set =  statement.executeQuery(q1);
		 java.sql.ResultSetMetaData metaData = set.getMetaData();

		 System.out.println(metaData.getColumnCount());
		 System.out.println(metaData.getColumnTypeName(2));
		 System.out.println(metaData.getColumnDisplaySize(2));
		 System.out.println(metaData.getColumnName(2));

		System.out.println("operations are completed");
	}
}
```

Example :- Java.sql.DataBaseMetaData:-

DataBaseMetaData is used to get the metadata of the particular table .Database username, Drivername,atabase version……..etc

abc.properties:

```
username=system
password=ratan
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:xe
```

TestDb.java

```
package com.dss;
import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;
class TestDb
{	public static Connection createConnection()
	{	Connection connection = null;
		try {	Properties properties = new Properties();
			properties.load(new FileInputStream("abc.properties"));
			Class.forName(properties.getProperty("driver"));
			connection = DriverManager.getConnection(properties.getProperty("url"),
			properties.getProperty("username"),properties.getProperty("password"));
		} catch (IOException | ClassNotFoundException | SQLException e) {
				e.printStackTrace();		}
		return connection;
	}
}
```

TestDb1.java

```
package com.dss;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.SQLException;
public class TestDb1 {
	public static void main(String[] args) throws SQLException {
		Connection connection = TestDb.createConnection();
		DatabaseMetaData metaData = connection.getMetaData();
		System.out.println("Database name : "+metaData.getDatabaseProductName());
		System.out.println("Database version : "+metaData.getDatabaseProductVersion());
		System.out.println("Database useranme: "+ metaData.getUserName());
		System.out.println("Database url:"+metaData.getURL());
		System.out.println("Database driver:"+metaData.getDriverName());
		System.out.println("Database driver version:"+metaData.getDriverVersion());
		connection.close();
		System.out.println("connection closed successfully.....");
	}
}
```

Steps to design standalone applications:- [not required]
Step 1:- prepare the component and add the components to the frame
Step 2:- set the particular layout to the frame.
Step 3:-conversion of static component into the dynamic component.(by adding listeners)

Example 1:-

```
import java.sql.*;
import java.awt.*;
import java.awt.event.*;
class SearchFrame extends Frame implements ActionListener
{       Label l;
        TextField tf;
        Button b;
        Connection con;
        Statement st;
        ResultSet rs;

        SearchFrame()
        {       try
                {       this.setVisible(true);
                        this.setSize(500,400);
                        this.setBackground(Color.pink);
                        this.setTitle("JDBC - AWT Application");
                        this.setLayout(new FlowLayout());
                        l=new Label("productId");
                        tf=new TextField(15);
                        b=new Button("Search");
                        b.addActionListener(this);
                        this.add(l);
                        this.add(tf);
                        this.add(b);
                        Class.forName("oracle.jdbc.driver.OracleDriver");

        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
                        st=con.createStatement();
                }
                catch (Exception e)
                {       e.printStackTrace();
                }
        }
        public void actionPerformed(ActionEvent ae)
        {       try
                {       rs=st.executeQuery("select * from emp where eno='"+tf.getText()+"'");
                        repaint();
                }
                catch (Exception e)
                {       e.printStackTrace();
                }
```

```
	}
	public void paint(Graphics g)
	{	try
		{
		Font f=new Font("arial",Font.BOLD,30);
		g.setFont(f);
		boolean b=rs.next();
			if(b==true)
			{	g.drawString("emp id....."+rs.getInt(1),50,100);
				g.drawString("emp Name....."+rs.getString(2),50,150);
				g.drawString("emp sal....."+rs.getInt(3),50,200);
			}
			else
			{	g.drawString("emp does not exists",50,150);
			}
		}
		catch (Exception e)
		{	e.printStackTrace();
		}
	}
}
class Test
{	public static void main(String[] args)
	{	new SearchFrame();
	}
}
```

Example:-

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
class MyFrame extends Frame implements ActionListener
{       Button b1,b2;
        Connection con;
        ResultSet rs;
        Statement st;
        String label;
        MyFrame()
        {       try{
                this.setVisible(true);
                this.setSize(500,500);
                this.setBackground(Color.red);
                b1=new Button("NEXT");
                b2=new Button("PREVIOUS");
                this.setLayout(new FlowLayout());
                this.add(b1);
                this.add(b2);
                b1.addActionListener(this);
                b2.addActionListener(this);
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                con=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
        st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
                rs=st.executeQuery("select * from emp");
                }
                catch(Exception e)
                {       System.out.println(e);
                }
        }
        public void actionPerformed(ActionEvent e)
        {       label=e.getActionCommand();
                repaint();
        }
        public void paint(Graphics g)
        {       try{
                if (label.equals("NEXT"))
                {       boolean b=rs.next();
                        if (b==true)
                        {       g.drawString("emp number"+rs.getInt(1),50,100);
                                g.drawString("emp name"+rs.getString(2),50,200);
                                g.drawString("emp sal"+rs.getInt(3),50,300);
                        }
                        else
                        {       g.drawString("no record",50,500);
                        }
                }
```

```
                if (label.equals("PREVIOUS"))
                {       boolean b=rs.previous();
                        if (b==true)
                        {       g.drawString("emp number"+rs.getInt(1),50,100);
                                g.drawString("emp name"+rs.getString(2),50,200);
                                g.drawString("emp sal"+rs.getInt(3),50,300);
                        }
                        else
                        {       g.drawString("no record",50,500);
                        }
                }
                }
                catch(Exception e)
                {       System.out.println(e);          }
        }
};
class Test
{       public static void main(String[] args)
        {               MyFrame f=new MyFrame();
        }
};
```

Transaction Management:-

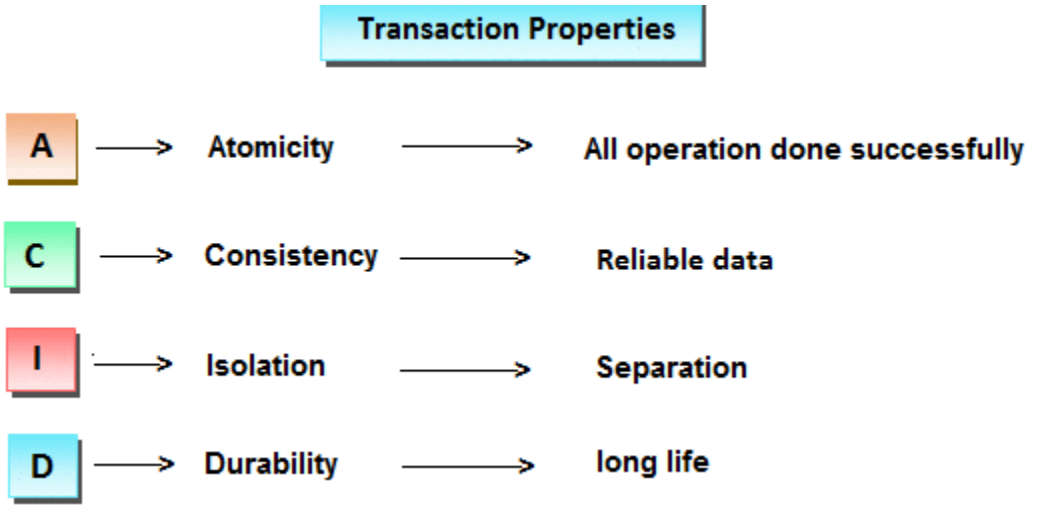
Transaction represents a single unit of work.

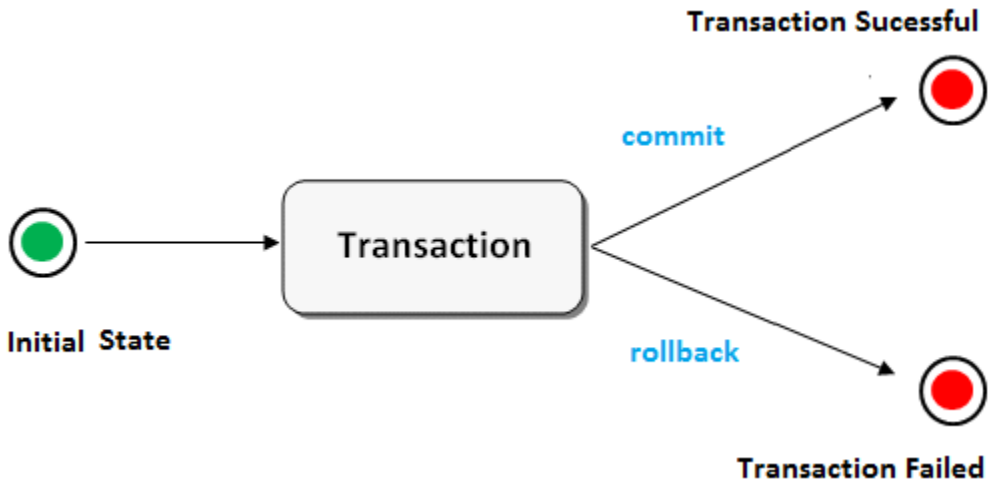
The ACID properties describes the transaction management well.

ACID stands for

Atomicity
Consistency
Isolation
Durability.

Atomicity *means either all successful or none.*
Consistency *ensures bringing the database from one consistent state to another consistent state.*
Isolation *ensures that transaction is isolated from other transaction.*
Durability *means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.*





Suppose a movie ticket booking at online is a transaction. This task contains four operation.
Verify the seats
Reserve the seats
Payment
Issue tickets

If all the above four operations are done successfully then a transaction is finished successfully. In the middle, if any one operation is failed then all operation are canceled and finally a transaction is failed.

Types of Transaction:

1. *Local Transaction*
2. *Distributed or global transaction*

Local Transaction:- *A local transaction means, all operation in a transaction are executed against one database.*

For example; If transfer money from first account to second account belongs to same bank then transaction is local transaction.

Global Transaction:- *A global transaction means, all operations in a transaction are executed against multiple database.*

For Example; If transfer money from first account to second account belongs to different banks then the transaction is a global transaction.

Note: *Jdbc technology performs only local transactions. For global transaction in java we need either EJB or spring framework.*

Note: *The operation in a transaction management may be executed on same table or different table but database should be same.*

Note*: In transaction management DDL(create ,alter drop) commands are not allowed.*

Java.sql.SavePoint :-

To manage the transactions in JDBC we have to use SavePoint interface. And it is introduced in JDBC3.0 verison.

By default when we run the query in database the database is permanently updated because the database auto commit mode is by default true.

To perform the transaction we have to set auto commit is false then it is possible to manage the transaction by using commit and rollback operations.

To change connection's auto commit mode we have to use the following method from Connection.

public void setAutoCommit(boolean b)throws SQLException

If b==true then the connection will be in auto commit mode else the connection will be in non-auto commit mode.

Example : con.setAutoCommit(false);

If we change connection's auto commit mode then we have to perform either commit or rollback operations to complete the transactions.

To perform commit and roll back operations we have to use the following methods from Connection.

public void commit()throws SQLException

public void rollback()throws SQLException

Note: *In case of connection's non-auto commit mode, when we submit SQL query to the connection then connection will send that SQL query to Database Engine and make the Database Engine to execute that SQL query and store the results on to the database table temporarily. In this case, Database Engine may wait for commit or rollback signal from client application to complete the transactions.*

Note:*Multiple savepoints can exist within a single transaction. Savepoints are useful for implementing complex error recovery in database.*

It is possible to place multiple savepoints in single source file but it is possible to rollback up to only one savepoint at a time.

Example:-

To do transaction management in Jdbc, we need to follow the below steps.

Step 1: Disable auto commit mode of Jdbc
Step 2: Put all operation of a transaction in try block.
Step 3: If all operation are done successfully then commit in try block, otherwise rollback in catch block.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

class TestDb
{       public static void main(String[] args) throws ClassNotFoundException, SQLException
        {       Class.forName("oracle.jdbc.driver.OracleDriver");
        Connection con
        =DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                System.out.println("connection created successfully.....");

                Statement stmt=con.createStatement();
                con.setAutoCommit(false);

                try
                {               stmt.executeUpdate("insert into emp values(111,'ratan',10000)");
                                stmt.executeUpdate("update emp set esal=esal+100 where esal>1000");
                                stmt.executeUpdate("delete from emp where eid=222");
                                con.commit();
                                System.out.println("Transaction is success");
                }//end of try
                catch (Exception e)
                {               con.rollback();
                                System.out.println("Trasaction is failed");
                }
                con.close();
                System.out.println("connection is closed");
        } //end of main
} //end of class
```

Example :

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Savepoint;
import java.sql.Statement;

class TestDb
{       public static void main(String[] args) throws ClassNotFoundException, SQLException
        {
                Savepoint savepoint = null;
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection con
=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                System.out.println("driver is loaded");
                Statement stmt=con.createStatement();
                con.setAutoCommit(false);
                try
                {               stmt.executeUpdate("insert into emp values(555,'ratan',10000)");
                                savepoint = con.setSavepoint();
                                stmt.executeUpdate("update emp set esal=esal+100 where esal>1000");
                                stmt.executeUpdate("delete from emp where eid===111");//error
                                con.commit();
                                System.out.println("Transaction is success");
                }//end of try
                catch (Exception e)
                {               con.rollback(savepoint);
                                System.out.println("Trasaction is failed");
                }
                con.close();
                System.out.println("connection is closed");
        } //end of main
} //end of class
```

RowSet interface:-(Introduced in JDBC 3.0)

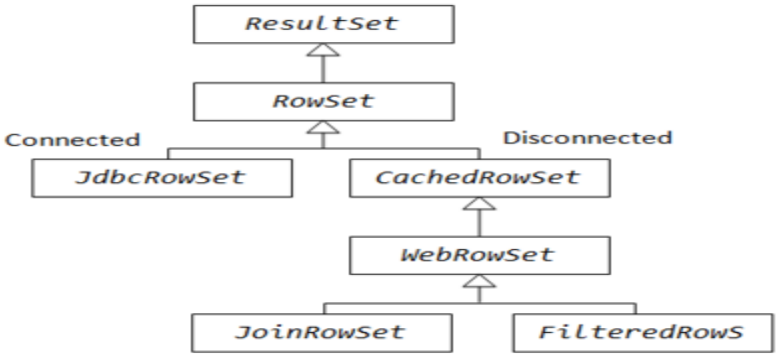
It is used to store the table data coming from database but is flexible to use compare to ResultSet object.
The implementation classes of RowSet interface is,

- ✓ *JdbcRowSet*
- ✓ *CachedRowSet*
- ✓ *WebRowSet*
- ✓ *JoinRowSet*
- ✓ *FilteredRowSet*

The all classes are present in javax.sql package

The advantages of using RowSet are given below:

- ✓ *It is easy and flexible to use*
- ✓ *It is Scrollable and Updatable by default.*



Example-1 :-

```
package com.dss;
import java.sql.SQLException;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;
public class TestDb {
	public static void main(String[] args) throws SQLException, ClassNotFoundException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
		rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
		rowSet.setUsername("system");
		rowSet.setPassword("ratan");

		rowSet.setCommand("select * from emp");
		rowSet.execute();
		  while (rowSet.next()) {
	System.out.println("Emp Id: " + rowSet.getInt(1));
	System.out.println("Emp Name: " + rowSet.getString(2));
	System.out.println("Emp Salary: " + rowSet.getDouble(3));
	System.out.println("*****************************");
    }
	}
}
```

RowSet Object creation ways :

```
JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();//single line code project level
//2-lines of code
RowSetFactory rowSetFactory = RowSetProvider.newFactory();
JdbcRowSet rowSet = rowSetFactory.createJdbcRowSet();
```

Example -2:

```
package com.dss;
import java.sql.SQLException;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;
public class TestDb {
        public static void main(String[] args) throws SQLException, ClassNotFoundException {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet();
                rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
                rowSet.setUsername("system");
                rowSet.setPassword("ratan");

                rowSet.setCommand("select * from emp");
                rowSet.execute();
                rowSet.afterLast();
                  while (rowSet.previous()) {
        System.out.println("Emp Id: " + rowSet.getInt(1));
        System.out.println("Emp Name: " + rowSet.getString(2));
        System.out.println("Emp Salary: " + rowSet.getDouble(3));
        System.out.println("*****************************");
    }
                   rowSet.absolute(2);
                   System.out.println("Emp Id: " + rowSet.getInt(1));
        }
}
```

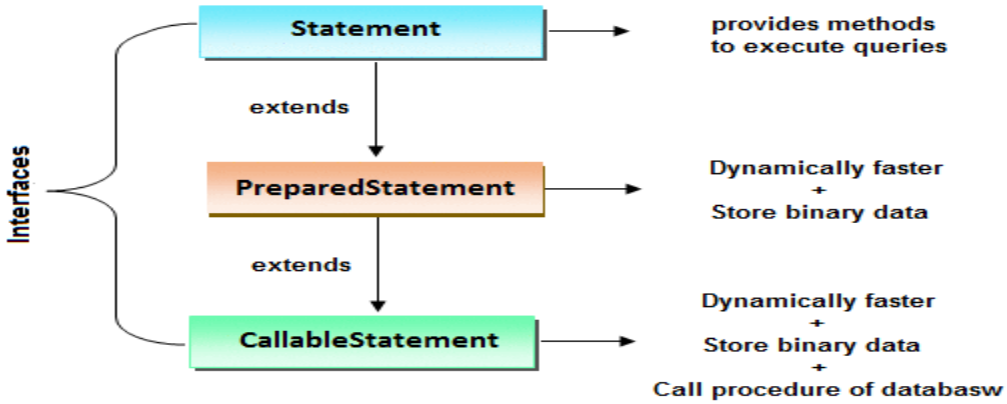
Example 3:-

```
                rowSet.setCommand("select * from emp");
                rowSet.execute();
                System.out.println("EmpId \t EmpName \t EmpSal");
                while (rowSet.next()) {
        System.out.println(rowSet.getInt(1) + "\t" + rowSet.getString(2) + "\t" + rowSet.getDouble(3));
                                }
```

Output:-

```
EmpId   EmpName       EmpSal
111             ratan           20000.0
222             anu             40000.0
333             dileep          50000.0
```

Java.sql.CallableStatement:



- *This interface used to execute stored procedures and functions of PL/SQL.*
- *Procedures & functions are precompiled queries available at database side. It will takes the input & produce the output.*
- *We can write business logics in data base with procedures and functions it will improve the performance because these are pre compiled.*
- *To get the employee salary based on eid then creates the procedure or function to take the eid as a input and returns salary as a output.*

Stored procedures:-

1. *Stored procedures are pre compiled queries available at database side.*
2. *Procedure will take the input & produce the output.*
3. *Stored procedure return value is optional.*
4. *Procedure can have both input and output parameters(IN & OUT)*
5. *Procedure is able to call functions.*

Syntax:-

```
create or replace procedure procedure_name([param-list])
as
---------- Global declarations
BEGIN
---------- Database logic
END procedure_name;
/
 (press enter to save and compile the procedure)
```

Functions:-

1. *Functions are pre compiled queries available at database side.*
2. *Functions will take the input & produce the output.*
3. *Function must return the value.*
4. *Function can have only IN(input) parameter.*
5. *From the functions we are unable to call procedures.*

Syntax:-

create or replace function function_name([param-list]) return data_type
as
---------- Global declarations
BEGIN
---------- Database logic
return value;
END function_name;

/ (press enter to save and compile the function)

IN :- *it is a input value to procedure or function call & set the input value by using setXXX().*
OUT : - it is a output value of procedure or function & get the output value by using getXXX().
INOUT :- it is parameter of both input and output values set the input value by using setXXX() and get the value by using getXXX().

Stored Procedure (SP)	**Function (UDF – User Defined Function)**
SP can return zero , single or multiple values.	Function can return one value which is mandatory.
We can use transaction in SP.	We can't use transaction in UDF.
SP can have input/output parameter.	Only input parameter.
We can called function from SP.	We can't call SP from function.
We can't use SP in SELECT/ WHERE/ HAVING statement.	We can use UDF in SELECT/ WHERE/ HAVING statement.
We can use exception handling using Try-Catch block in SP.	We can't use Try-Catch block in UDF.

SQL Data Types	JDBC Typecodes	Standard Java Types	Oracle Extension Java Types
CHAR	java.sql.Types.CHAR	java.lang.String	oracle.sql.CHAR
VARCHAR2	java.sql.Types.VARCHAR	java.lang.String	oracle.sql.CHAR
LONG	java.sql.Types.LONGVARCHAR	java.lang.String	oracle.sql.CHAR
NUMBER	java.sql.Types.NUMERIC	java.math.BigDecimal	oracle.sql.NUMBER
NUMBER	java.sql.Types.DECIMAL	java.math.BigDecimal	oracle.sql.NUMBER
NUMBER	java.sql.Types.BIT	boolean	oracle.sql.NUMBER
NUMBER	java.sql.Types.TINYINT	byte	oracle.sql.NUMBER
NUMBER	java.sql.Types.SMALLINT	short	oracle.sql.NUMBER
NUMBER	java.sql.Types.INTEGER	int	oracle.sql.NUMBER
NUMBER	java.sql.Types.BIGINT	long	oracle.sql.NUMBER
NUMBER	java.sql.Types.REAL	float	oracle.sql.NUMBER
NUMBER	java.sql.Types.FLOAT	double	oracle.sql.NUMBER
NUMBER	java.sql.Types.DOUBLE	double	oracle.sql.NUMBER
RAW	java.sql.Types.BINARY	byte[]	oracle.sql.RAW
RAW	java.sql.Types.VARBINARY	byte[]	oracle.sql.RAW
LONGRAW	java.sql.Types.LONGVARBINARY	byte[]	oracle.sql.RAW
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE
DATE	java.sql.Types.TIME	java.sql.Time	oracle.sql.DATE
TIMESTAMP	java.sql.Types.TIMESTAMP	javal.sql.Timestamp	oracle.sql.TIMESTAMP

Steps to design Application :

Step 1: Create the callable statement object by using prepareCall() method of Connection interface.

public CallableStatement prepareCall(String pro_cal)throws SQLException

```
CallableStatement cst=connection.prepareCall("{call getSal(?,?)}");
```

When JVM encounters the above instruction JVM will pick up procedure call and send to Database Engine, where Database Engine will parse the procedure call and prepare a query plan with the positional parameters, as a result CallableStatement object will be created at Java application.

Step 2:- *If we have IN type parameters in CallableStatement object then set values to IN type parameters.*

public void setXxx(int param_position, xxx value)

Where xxx may be byte, short, int and so on.

ex: callableStatement.setInt(1, 111);

Step 3:- *If we have OUT type parameter in CallableStatement object then we have to register OUT type parameter with a particular data type.*

public void registerOutParameter(int param_position, int data_type)

Where data_type may be the constants from Types class : BYTE, SHORT, INTEGER,FLOAT ...etc

ex: callableStatement.registerOutParameter(2,Types.FLOAT);

Step 4:- Make Database Engine to pick up the values from Query plan and to execute the respective procedure or function.

public void execute()throws SQLException

ex : callableStatement.executeUpdate();

Step 5:- Get the values from OUT type parameters available in CallableStatement object.

After executing the respective procedure or function the respective values will be stored in OUT type parameters in CallableStatement object from stored procedure or functions. Toaccess the OUT type parameter values we have to use the following method.

public xxx getXxx(int param_position) Where xxx= byte, short, int...etc

ex: callableStatement.getFloat(2)

abc.properties

```
username=system
password=ratan
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:xe
```

JDBC code to create the connection: (Reading data from properties file)

```
package com.dss;

import java.io.FileInputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class TestCon {
	public static Connection createConnection()
	{	Connection connection=null;
		try
		{	FileInputStream inputStream = new FileInputStream("abc.properties");
			Properties properties = new Properties();
			properties.load(inputStream);

			Class.forName(properties.getProperty("driver"));
			connection = DriverManager.getConnection(properties.getProperty("url"),
			properties.getProperty("username"),properties.getProperty("password"));
		}
		catch (ClassNotFoundException|SQLException | IOException e) {
			e.printStackTrace();
		}
		return connection;
	}
}
```

Note: in below example we are using the above created connection.

Example -1

Procedure Creation :

```
create or replace procedure getSal(id IN number, sal OUT number)
as
BEGIN
select esal into sal from emp where eid=id;
END getSal;
/
```

Jdbc code to call the procedure:

Step 1: *call the procedure*
Step 2: *set the input to procedure*
Step 3: *register the output parameter*
Step 4: *execute the procedure; the result will be stored in register parameter*
Step 5: *Check the result & release the resources*

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Types;

public class TestDb3 {
	public static void main(String[] args) throws SQLException {
		Connection connection =  TestCon.createConnection();

		CallableStatement callableStatement = connection.prepareCall("{call getSal(?,?)}");
		callableStatement.setInt(1, 111);
		callableStatement.registerOutParameter(2,Types.FLOAT);
		callableStatement.executeUpdate();
		System.out.println("Result="+callableStatement.getFloat(2));

		callableStatement.close();
		connection.close();
		System.out.println("Connection closed successfully....");
	}
}
```

Example -2:

Procedure Creation :

```
create or replace procedure insert11(id IN number,name IN varchar2, sal IN number)
as
BEGIN
insert into emp values(id,name,sal);
END ;
/
```

Jdbc code executing procedure:

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Types;

public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Connection connection =  TestCon.createConnection();

		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

	CallableStatement callableStatement = connection.prepareCall("{call insert11(?,?,?)}");

	callableStatement.setInt(1, 111);
	callableStatement.setString(2, "ratan");
	callableStatement.setInt(3, 50000);

	callableStatement.executeUpdate();
	System.out.println("operations are completed");

	connection.close();
	System.out.println("connection is closed");
	}
}
```

Example 3:

procedure creation

```
create or replace procedure getDetails(id IN number, o1 OUT number,o2 OUT varchar,o3 OUT number)
as
BEGIN
select eid,ename,esal into o1,o2,o3 from emp where eid=id;
END getDetails;
/
```

Jdbc code executing procedure :

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;

public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

CallableStatement callableStatement = connection.prepareCall("{call getDetails(?,?,?,?)}");

callableStatement.setInt(1, 999);
callableStatement.registerOutParameter(2, Types.INTEGER);
callableStatement.registerOutParameter(3, Types.VARCHAR);
callableStatement.registerOutParameter(4,Types.FLOAT);
callableStatement.executeUpdate();

System.out.println(callableStatement.getInt(2));
System.out.println(callableStatement.getString(3));
System.out.println(callableStatement.getFloat(4));

System.out.println("operations are completed");

	connection.close();
System.out.println("connection is closed");
	}
}
```

Example 4:

Procedure Creation :

```
create or replace procedure getEmps(emps OUT SYS_REFCURSOR, sal IN number)
as
BEGIN
open emps for
select * from emp where esal>sal;
END getEmps;
/
```

Jdbc code calling procedure :

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;

public class TestDb3 {
	public static void main(String[] args) throws SQLException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
	DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

		CallableStatement callableStatement =
				connection.prepareCall("{call getEmps(?,?)}");
		callableStatement.setDouble(2, 10000);
		callableStatement.registerOutParameter(1, oracle.jdbc.OracleTypes.CURSOR);
		callableStatement.execute();
		ResultSet rs=(ResultSet)callableStatement.getObject(1);
		System.out.println("EID   ENAME   ESAL");
		System.out.println("--------------------");
		while (rs.next())
		{	System.out.println(rs.getInt(1)+"   "+rs.getString(2)+"   "+rs.getFloat(3));
		}
		callableStatement.close();
		connection.close();
		System.out.println("Connection closed successfully....");
	}
}
```

Example : ***CallableStatement vs BatchUpdation***

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;

public class TestDb1 {
	public static void main(String[] args) throws SQLException {

		Connection connection = TestDb.createConnection();
		CallableStatement callableStatement = connection.prepareCall("{call insert11(?,?,?)}");

		callableStatement.setInt(1, 111);
		callableStatement.setString(2, "ratan");
		callableStatement.setInt(3, 1000);
		callableStatement.addBatch();

		callableStatement.setInt(1, 222);
		callableStatement.setString(2, "anu");
		callableStatement.setInt(3, 2000);
		callableStatement.addBatch();


		callableStatement.executeBatch();
		System.out.println("operations are completed");

		connection.close();
		System.out.println("connection is closed");
	}
}
```

Example -5

Execution of functions:

```
create or replace function getAvg(id1 IN number, id2 IN number) return number
as
sal1 number;
sal2 number;
BEGIN
select esal into sal1 from emp where eid=id1;
select esal into sal2 from emp where eid=id2;
return (sal1+sal2)/2;
END getAvg;
/
```

Jdbc code executing function :

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;

public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

CallableStatement callableStatement = connection.prepareCall("{?=call getAvg(?,?)}");
callableStatement.setInt(2,999);
callableStatement.setInt(3,888);
callableStatement.registerOutParameter(1, Types.FLOAT);

callableStatement.executeUpdate();

System.out.println("Result="+callableStatement.getFloat(1));
System.out.println("operations are completed");

		connection.close();
System.out.println("connection is closed");
		}
}
```

Example-6:

Function creation :

```
create or replace function getEmployees(sal IN number) return SYS_REFCURSOR
as
Employees SYS_REFCURSOR;
BEGIN
open Employees for
select * from emp where esal>sal;
return Employees;
END getEmployees;
/
```

Jdbc code executing function :

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Types;

public class TestDb {
        public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                System.out.println("connection created successfully.....");

CallableStatement callableStatement = connection.prepareCall("{?=call getEmployees (?)}");
callableStatement.setFloat(2,20000);
callableStatement.registerOutParameter(1,oracle.jdbc.OracleTypes.CURSOR);

callableStatement.execute();
ResultSet set=(ResultSet) callableStatement.getObject(1);
System.out.println("EID   ENAME   ESAL");
System.out.println("--------------------");
while (set.next())
{       System.out.println(set.getInt(1)+"   "+set.getString(2)+"   "+set.getFloat(3));
}
connection.close();
System.out.println("connection is closed");

        }
}
```

Batch Updations :

Example: Application without batch updation.

In below example if we are executing 5-queries,

- *5 times we are sending request to database*
- *5-times we are receiving response.*

In above context if the number of queries are increased number of network round trips are increased it decreases performance of the application.

If the number of round trips is increased between an application and database, then it will reduce the performance of an application. To overcome these problems we use Batch Processing.

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Scanner;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");
		Scanner scanner = new Scanner(System.in);

		Statement statement = connection.createStatement();

		String q1="create table emp2(eid number, ename varchar2(30), esalary number)";
		String q2="insert into emp2 values(11,'vishwada',10000)";
		String q3="insert into emp2 values(12,'ajay',15000)";
		String q4="update emp2 set esalary=esalary+500 where esalary>15000";
		String q5="drop table emp2";

		statement.executeUpdate(q1);
		statement.executeUpdate(q2);
		statement.executeUpdate(q3);
		statement.executeUpdate(q4);
		statement.executeUpdate(q5);

		System.out.println("operations are completed");
	}
}
```

Example :-

- ✓ *If the number of round trips is increased between an application and database, then it will reduce the performance of an application. To overcome these problems we use Batch Processing.*
- ✓ *In above context if the number of queries are increased number of network round trips are increased it decreases performance of the application.*

  To overcome above limitation use batch updation concept.
- ✓ *In batch updating take the all the queries into single unit like batch by using addBAtch() method.*

  public abstract void addBatch(java.lang.String) throws java.sql.SQLException;
- ✓ *Execute the batch by using executeBatch() method of Statement interface,*

  public abstract int[] executeBatch() throws java.sql.SQLException;

when we execute the query by using executeBatch() at a time the batch is send to database side then database engine will execute all queries at a time and it will send the result to jdbc application in the form of int[] contains all the results.

Note: *If we include selection group SQL query in a batch then JVM will raise an Exception like ava.sql.BatchUpdateException: invalid batch command: invalid SELECT batch command.*

```
package com.dss;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

		Statement statement = connection.createStatement();
		String q1="create table emp2(eid number, ename varchar2(30), esalary number)";
		String q2="insert into emp2 values(11,'vishwada',10000)";
		String q3="insert into emp2 values(12,'ajay',15000)";
		String q4="update emp2 set esalary=esalary+500 where esalary>15000";
		String q5="drop table emp2";
		statement.addBatch(q1);
		statement.addBatch(q2);
		statement.addBatch(q3);
		statement.addBatch(q4);
		statement.addBatch(q4);
		int[] a = statement.executeBatch();
		for(int aa : a)
		{	System.out.println(aa);		}
		System.out.println("operations are completed");
	}
}
```

Example :- PreparedStatement with batch updations

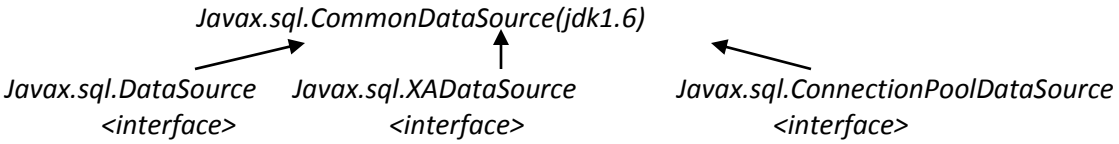
```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
InterruptedException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

		PreparedStatement preparedStatement =
	connection.prepareStatement("update emp set ename=? where eid=?");
		preparedStatement.setString(1, "Chitanya");
		preparedStatement.setInt(2, 999);
		preparedStatement.addBatch();

		preparedStatement.setString(1, "Nalanda");
		preparedStatement.setInt(2, 888);
		preparedStatement.addBatch();

		int[] a = preparedStatement.executeBatch();
		for(int aa:a)
		{		System.out.println(aa);
		}
		connection.close();
		System.out.println("connection is closed");
	}
}
```

Example :- CallableStatement vs BatchUpdation

```
package com.dss;

import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.SQLException;

public class TestDb1 {
	public static void main(String[] args) throws SQLException {

		Connection connection = TestDb.createConnection();
		CallableStatement callableStatement = connection.prepareCall("{call insert11(?,?,?)}");

		callableStatement.setInt(1, 111);
		callableStatement.setString(2, "ratan");
		callableStatement.setInt(3, 1000);
		callableStatement.addBatch();

		callableStatement.setInt(1, 222);
		callableStatement.setString(2, "anu");
		callableStatement.setInt(3, 2000);
		callableStatement.addBatch();


		callableStatement.executeBatch();
		System.out.println("operations are completed");

		connection.close();
		System.out.println("connection is closed");
	}
}
```

Connection Pooling:-

- ✓ *The connection pool mechanism used to establish the pool of connections before client make the request.*
- ✓ *The connection pool contains logical connection.*
- ✓ *The connections are retrieved from the pool use the connection finally return back to the pool.*
- ✓ *Connection pooling means that connections are reused rather than created each time a connection is requested.*
- ✓ *The advantages of connection pooling is improves the performance.*



Physical connection :-

- ✓ *These type of connections are established either using the driverManager or DataSource.*
- ✓ *The connection established directly to database when the client is requested & it is destroyed when the operations are completed.*
- ✓ *Connection.close() destroyed the connection.*

Logical connection:-

- ✓ *Logical or pooled connections those connection objects are created & maintained by pool manager.*
- ✓ *When you send the request to connection pool just it will give the logical connection if the connections are available.*
- ✓ *If the connections are reached to maximum limit the request is queued.*
- ✓ *Once the connection use is completed then it is return back to pool. But not destroyed or garbage collected.*
- ✓ *The connection objects destruction & maintained by pool manager.*
- ✓ *Connection.close() method return the connection to pool.*

Javax.sql.DataSource:-

1. *For local transaction.*
2. *Must be implemented by Third-Party CP vendor such as DBCP, C3P0, Proxool, etc*
   a. *Java.sql.Connection**ds.getConnection();** //Gets Logical connection from CP.*
3. *con.close(); //Returns logical connection back to CP.*

Javax.sql.XADataSource:-

1. *For Distributed Transactions*
2. *Must be implemented by Third-Party CP vendor.*
3. *Javax.sql.XAConnection ds.getXAConnection(); //Gets logical connection from cp*
4. *con.close(); //Returns logical connection back to CP.*

Javax.sql.ConnectionPoolDataSource:-

1. Must be implemented by JDBC Driver Vendor such as Oracle, MSSQL, MYSQL, etc.
2. *Javax.sql.PooledConnection cpds.getPooledConnection();*
   //Establish Physical connections with DB.

3. *pc.close(); //Physical connection with DB is closed.*

In general in Jdbc applications, when we have a requirement to perform database operations we will establish the connection with the database from a Java application, at the end of the application we will close the connection i.e. destroying Connection object.

In Jdbc applications, every time establishing the connection and closing the connection may increase burden to the Jdbc application, it will reduce the performance of the jdbc application.
In the above context, to improve the performance of Jdbc applications we will use an alternative called as Connection Pooling.

In Connection pooling at the time of application startup we will prepare a fixed number of Connection objects and we will keep them in a separate base object called ***Pool object.***

In Jdbc applications, when we have a requirement to interact with the database then we will get the Connection object from Pool object and we will assign it to the respective client application. At the end of the Jdbc application we will keep the same Connection object in the respective Pool object without destroying.
The above mechanism will improve the performance of the application is called as ***Connection Pooling.***

Example :-Connection with BasicDataSource provided by c3p0 vendor.

```
package com.dss;

import java.sql.Connection;
import java.sql.SQLException;

import org.apache.commons.dbcp.BasicDataSource;

public class TestDb2 {

	public static void main(String[] args) throws SQLException {
		BasicDataSource dataSource = new BasicDataSource();
		dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
		dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
		dataSource.setUsername("system");
		dataSource.setPassword("ratan");
		dataSource.setMaxActive(10);
		Connection connection=null;
		for(int i=0;i<100;i++){
		connection = dataSource.getConnection();
		System.out.println(connection);}
		connection.close();
	}
}
```

To run the above application we need two jar files

a. *commons-dbcp-1.0.jar*
b. *commons-pool-1.5.2 jar*

If we want to implement Connection pooling in Jdbc application we have to use the following steps.

Step 1: *Prepare DataSource object.*

DataSource is an object, it is able to manage all the Jdbc parameter which are required to establish the connections.

To represent DataSource object Java API has provided a predefined interface i.e. javax.sql.DataSource.

DataSource is an interface provided by Jdbc API, but whose implementation classes are provided by all the database vendors.

With the above convention Oracle has provided an implementation class to DataSource interface in ojdbc6.jar file i.e. oracle.jdbc.pool.OracleConnectionPoolDataSource.

Ex: *OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();*

Step 2: *Set the required Jdbc parameters to DataSource object.*

To set the Jdbc parameters like Driver url, database username and password to the DataSource object we have to use the following methods.

public void setURL(String driver_url)

public void setUser(String user_name)

public void setPassword(String password)

example :-

```
ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");
ds.setUser("system");
ds.setPassword("venkat");
```

Step 3: *Get the PooledConnection object.*

PooledConnection is an object provided by DataSource, it can be used to manage number of Connection objects.

To represent PooledConnection object Jdbc API has provided a predefined interface i.e. javax.sql.PooledConnection.

To get PooledConnection object we have to use the following method from DataSource.

public PooledConnection getPooledConnection()

Example: *PooledConnection pc=ds.getPooledConnection();*

Step 4: *Get Connection object from PooledConnection.*

To get Connection object from PooledConnection we have to use the following method.

public Connection getConnection()

Example: *Connection con=pc.getConnection();*

Step 5: *After getting Connection prepare Statement or preparedStatement or CallableStatement and perform the respective database operations.*

Example: *Statement st=con.createStatement();*

Example:-

```
import java.sql.*;
import javax.sql.*;
import oracle.jdbc.pool.*;
public class ConnectionPoolDemo
{       public static void main(String[] args) throws Exception
        {       OracleConnectionPoolDataSource ds=new OracleConnectionPoolDataSource();
                ds.setURL("jdbc:oracle:thin:@localhost:1521:xe");
                ds.setUser("system");
                ds.setPassword("ratan");
                PooledConnection pc=ds.getPooledConnection();

                Connection con=pc.getConnection();
                Statement st=con.createStatement();
                ResultSet rs=st.executeQuery("select * from emp");
                System.out.println("EID ENAME ESAL");
                System.out.println("------------------------------");
                while (rs.next())
                {       System.out.println(rs.getInt(1)+" "+rs.getString(2)+""+rs.getFloat(3));
                }
        }
}
```

Jdbc new features:-

Present version of jdbc is JDBC 4.0

JDBC 3.0 new features

a. RowSet interface
b. SavePoint interface in transaction management.

JDBC 4.0 new features

a. Automatic loading of driver.
b. Sub classes of SQLException.

Storing image into oracle database :-

- ✓ *You can't insert picture into database directly but we can store binary data of picture.*
- ✓ *To insert the imge into database we need BLOB data type.*

- ✓ *We can store the imgae into database by using PreparedStatement object.*
- ✓ *In JDBC only PreparedStatement supports the binary data transfer between java application to data base by providing setBinaryStream() method.*
- ✓ *Jdbc supports only gif or jpeg or png type of images to insert or read from a database.*

public abstract void setBinaryStream(int parameter-index , java.io.InputStream, long file-size) throws java.sql.SQLException;

Note:*In java forward slash (/) allowed but not backword slash at time of writing the path.*

Table creation process :-

create table imgtable1(name varchar2(20),image blob);

Example :

```
package com.dss;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
FileNotFoundException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
	DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

		File f=new File("Capture.PNG");
		FileInputStream fis=new FileInputStream(f);
	PreparedStatement pst=connection.prepareStatement("insert into imgtable values(?,?)");

		pst.setString(1,"ratan");
		pst.setBinaryStream(2,fis,(int)f.length());
		pst.executeUpdate();
		System.out.println("Image inserted Successfully");

		connection.close();
	}
}
```

Reading image form database :-

- ✓ *The image is stored in databse in binary format hece we must convert binary data to image format.*
- ✓ *When we select the image from databse it is stored in ResultSet object.*
- ✓ *From the resultset we need to read the binary data and we need to store data in InputStream object.*

Example :-

```
package com.dss;

import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class TestDb {
        public static void main(String[] args) throws ClassNotFoundException, SQLException,
IOException {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                System.out.println("connection created successfully.....");
                PreparedStatement pstmt=connection.prepareStatement("select photo from imgtable
where name=?");

                pstmt.setString(1,"ratan");
                // read the data from database
                ResultSet rs=pstmt.executeQuery();
                rs.next();
                InputStream is=rs.getBinaryStream(1);
                rs.close();
                //writing image to hard disk
                FileOutputStream fos=new FileOutputStream("f:/img001.gif");
                int k;
                while((k=is.read())!=-1)
                {       fos.write(k);
                }
                System.out.println("picture is ready open F:drive");
                fos.close();
                connection.close();
        }
}
```

Storeing file into database :-

- ✓ *To store the data in databse use clob data type.*
- ✓ *To set the data to table use setCharacterStream() method of preparedStatement object.*

Creation of table :

Create table ftable(id number,name clob);

Example :

```
package com.dss;

import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class TestDb {
	public static void main(String[] args) throws ClassNotFoundException, SQLException,
IOException {
		Class.forName("oracle.jdbc.driver.OracleDriver");
		Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
		System.out.println("connection created successfully.....");

		PreparedStatement preparedStatement = connection.prepareStatement("insert into
ftable values(?,?)");
		preparedStatement.setInt(1, 111);

		File f = new File("MyServlet.txt");
		FileReader reader = new FileReader(f);
		preparedStatement.setCharacterStream(2,reader, (int)f.length());

		preparedStatement.executeUpdate();

		System.out.println("data inserted successfully in database.....");

		preparedStatement.close();
		connection.close();
		System.out.println("connection closed successfully.....");
	}
}
```

Reading the file from database :-

```
package com.dss;

import java.io.FileWriter;
import java.io.IOException;
import java.io.Reader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class TestDb {
        public static void main(String[] args) throws ClassNotFoundException, SQLException,
IOException {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                System.out.println("connection created successfully.....");

PreparedStatement pstmt=connection.prepareStatement("select name from ftable where id=?");

                pstmt.setInt(1,111);
                ResultSet rs=pstmt.executeQuery();
                rs.next();
                Reader is=rs.getCharacterStream(1);
                rs.close();
                FileWriter fw=new FileWriter("f:/abc.txt");
                int k;
                while((k=is.read())!=-1)
                {       fw.write(k);
                }
                System.out.println("file is ready to open F:drive");
                fw.close();

                connection.close();
                System.out.println("connection closed successfully.....");
        }
}
```

1. *What is JDBC?*
2. *What is the present & initial version of java?*
3. *What is the difference between JDBC & ODBC?*
4. *What are the layers in web application?*
5. *What do you mean by deployment?*
6. *What do you mean by client & server?*
7. *What are the client side technologies & server side technologies?*
8. *What is the difference between technology & framework?*
9. *What do you mean by persistence logic?*
10. *What is the JDBC driver?*
11. *JDBC applications are standalone applications or web application?*
12. *What are the steps to design first application?*
13. *When we will get SQLException?*
14. *SQLException is checked or unchecked exception?*
15. *How many ways are there to handle checked exception?*
16. *What do you mean by factory method?*
17. *What is the difference instance method & static method?*
18. *When we will use extends & implements ?*
19. *How to take the input from the keyboard?*
20. *What is the difference between try-catch blocks and throws keyword?*
21. *How many drivers are available explain?*
22. *How to load the driver class into memory?*
23. *How to register driver into DriverManager service?*
24. *What do you mean by DriverManager service?*
25. *How to get Connection?*
26. *What are the key interfaces of JDBC?*
27. *What is the type-1 driver class-name & url pattern?*
28. *How to load the driver?*
29. *What is the purpose of Statement object?*
30. *What is the difference between DDL & DML & TCL?*
31. *Who will execute the database queries?*
32. *Explain stored procedures and functions in oracle?*
33. *What is the difference between class-path & bud path?*
34. *What is the logical name of the database and is it possible to change the logical name or not?*
35. *Explain about IN OUT INOUT parameters?*
36. *For every query execution 4-steps are performed what are those?*
37. *What is the purpose of the PreparedStatement?*
38. *What is the difference between Statement & PreparedStatement?*
39. *How to store the data coming from database?*
40. *What is the purpose of database?*
41. *What is the jar file of Oracle driver implementation?*
42. *What is the jar file of MySql driver implementation?*
43. *What is the oracle driver class-name & url pattern?*
44. *What is the mysql driver class-name & url pattern?*
45. *What is the difference between class-path & build path?*
46. *What is the purpose of DSN?*
47. *How to set the DSN name?*
48. *What are the statements present in JDBC?*

49. *What is the purpose of ResultSet object?*
50. *What is the purpose of CallableStatement?*
51. *Mainly how many methods are there to execute the query?*
52. *Which method is used to store all the type of queries in database?*
53. *What is the difference between executeUpdate() & executeQuery()?*
54. *What is the difference between executeQuery() & execute() ?*
55. *What are the return types of executeUpdate() & executeQuery() & execute()?*
56. *What do you mean by metadata?*
57. *How to set null values to PreaparedStatement ?*
58. *In preparedstatement what do you mean by **?** like parameter query?*
59. *What are the types of resultSet?*
60. *How to execute the parameterized query?*
61. *What is the default behavior of ResultSet object?*
62. *How to specify the resultSet cursor scrollable nature?*
63. *What is the difference between scrll sensitive & insensitive ?*
64. *How to move the cursor to last & first & absolute positions?*
65. *What is the initial position of the Resultset cursor?*
66. *What is the purpose of ResultSetMetadata?*
67. *What is the purpose of databasemetadata object?*
68. *How to execute the stored procedures & functions of Oracle?*
69. *Why we need batch updations?*
70. *What do you mean by batch processing and what is the advantages of batch processing?*
71. *Is it possible to all select query in batch updation?*
72. *What are the common database Exceptions?*
73. *How to get the database server details in java program?*
74. *How to get the metadata of the table?*
75. *How to specify the resultset object is scrollable and updatable?*
76. *By using which class we are specifying types in JDBC?*
77. *What is the purpose of the properties file?*
78. *How to read the data from properties file?*
79. *What is the JDBC transaction management?*
80. *How to rollback & commit the transaction?*
81. *Autocommit mode by default true or false?*
82. *By using which interface we are managing transaction?*
83. *What is the ACID properties?*
84. *How many types of transaction in JDBC?*
85. *By using jdbc is it possible to perform global transaction or not?*
86. *Is it possible to rollback create,drrop queries?*
87. *What is the purpose of SavePoint interface?*
88. *Is it possible to set more than one savepoint in single source file?*
89. *Is it possible to roll back the transaction upto more than one savepoint?*
90. *What are the benefits of JDBC 4.0 version?*
91. *If you want execute the same query more times then use Statement or Preparedstatement & why?*
92. *What is the connection pooling?*
93. *What is the purpose of RowSet interface?*
94. *What is the difference between RowSet & ResultSet?*
95. *In JDBC 4.0 loading the driver optional or mandatory?*

96. *What is the default behavior of RowSet interface?*
97. *What is the difference between java.util.Date & java.sql.Date?*
98. *What is the BLOB and CLOB data types in JDBC?*
99. *How to store the image in database?*
100. *How to store word document in database.*

Oracle data types and corresponding setter methods:-

Oracle Datatype	***setXXX()***
CHAR	*setString()*
VARCHAR2	*setString()*
NUMBER	*setBigDecimal(), setBoolean(), setByte(), setShort() setInt(), setLong(),setFloat(), setDouble().*
INTEGER	*setInt()*
FLOAT	*setDouble()*
CLOB	*setClob()*
BLOB	*setBlob()*
RAW	*setBytes()*
LONGRAW	*setBytes()*
DATE	*setDate(), setTime(), setTimestamp()*

SQL Data Types	JDBC Typecodes	Standard Java Types	Oracle Extension Java Types
CHAR	java.sql.Types.CHAR	java.lang.String	oracle.sql.CHAR
VARCHAR2	java.sql.Types.VARCHAR	java.lang.String	oracle.sql.CHAR
LONG	java.sql.Types.LONGVARCHAR	java.lang.String	oracle.sql.CHAR
NUMBER	java.sql.Types.NUMERIC	java.math.BigDecimal	oracle.sql.NUMBER
NUMBER	java.sql.Types.DECIMAL	java.math.BigDecimal	oracle.sql.NUMBER
NUMBER	java.sql.Types.BIT	boolean	oracle.sql.NUMBER
NUMBER	java.sql.Types.TINYINT	byte	oracle.sql.NUMBER
NUMBER	java.sql.Types.SMALLINT	short	oracle.sql.NUMBER
NUMBER	java.sql.Types.INTEGER	int	oracle.sql.NUMBER
NUMBER	java.sql.Types.BIGINT	long	oracle.sql.NUMBER
NUMBER	java.sql.Types.REAL	float	oracle.sql.NUMBER
NUMBER	java.sql.Types.FLOAT	double	oracle.sql.NUMBER
NUMBER	java.sql.Types.DOUBLE	double	oracle.sql.NUMBER
RAW	java.sql.Types.BINARY	byte[]	oracle.sql.RAW
RAW	java.sql.Types.VARBINARY	byte[]	oracle.sql.RAW
LONGRAW	java.sql.Types.LONGVARBINARY	byte[]	oracle.sql.RAW
DATE	java.sql.Types.DATE	java.sql.Date	oracle.sql.DATE
DATE	java.sql.Types.TIME	java.sql.Time	oracle.sql.DATE
TIMESTAMP	java.sql.Types.TIMESTAMP	javal.sql.Timestamp	oracle.sql.TIMESTAMP

JDBC DATA TYPES

JDBC TYPE	JAVA TYPE
BIT	boolean
TINY INT	byte
SMALL INT	short
INTEGER	int
BIG INT	long
REAL	Float
FLOAT DOUBLE	Double
BINARY VARBINARY LONGVARBINARY	byte[]
CHAR VARCHAR LONGVARCHAR	String

JDBC TYPE	JAVA TYPE
NUMERIC DECIMAL	BigDecimal
DATE	java.sql.Date
TIME TIMESTAMP	java.sql.Timestamp
CLOB	Clob
BLOB	Blob
ARRAY	Array
DISTINCT	Mapping of underlying type
STRUCT	Struct
REF	Ref
JAVA_OBJECT	Underlying java class

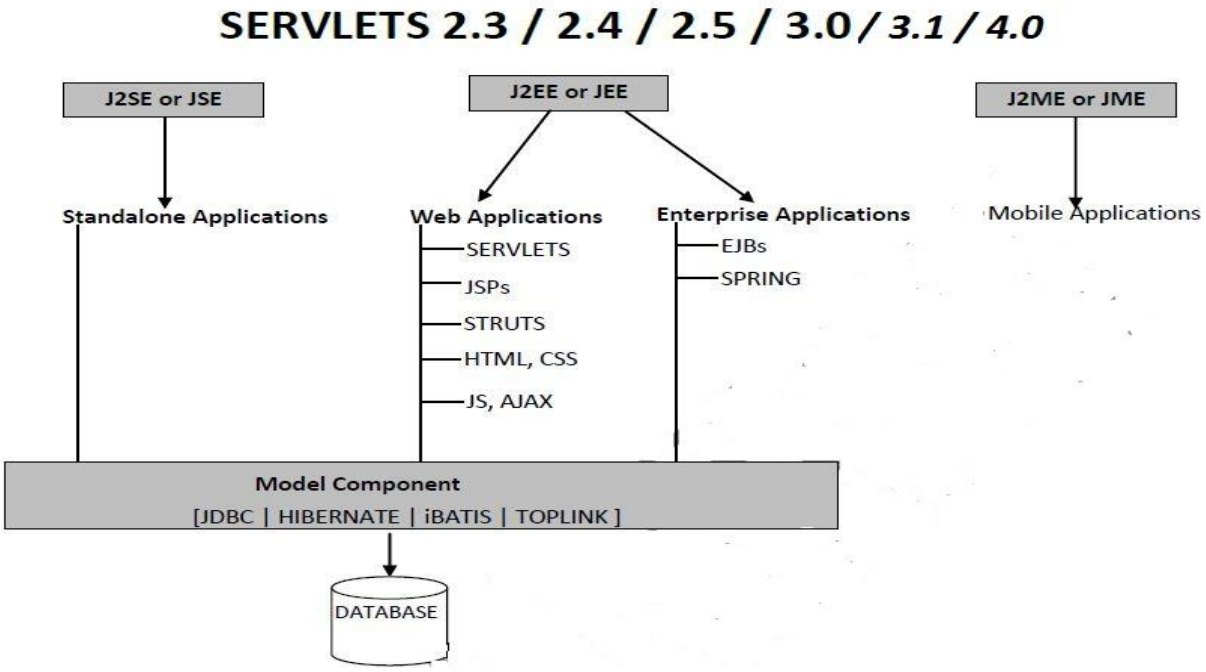
Servlets

- ✓ *Servlets is a first web technology used to develop the web application.*
- ✓ *Servlet is an object executed at server side.*
- ✓ *Servlets is a server side technology used to write the programing at server side.*

- ✓ *Servlet is a part of the J2EE & these are executed by Servlet container (web container).*
- ✓ *The present version of the servlet is **Servlet 4.0** introduced in 2017 may compatable with jdk8.*
- ✓ *The servlets predefined support in the form of fallowing packages.*
  - ***javax.servlet***
  - ***javax.servlet.http***
  - ***javax.servlet.annotation***

Servlet Versions History:

Servlet API version	Released	JSR Number	Platform	Important Changes
Servlet 4.0	Sep 2017	369	Java EE 8	HTTP/2
Servlet 3.1	May 2013	340	Java EE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket)[7]
Servlet 3.0	December 2009	315	Java EE 6, Java SE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Servlet 2.5	September 2005	154	Java EE 5, Java SE 5	Requires Java SE 5, supports annotation
Servlet 2.4	November 2003	154	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Servlet 2.3	August 2001	53	J2EE 1.3, J2SE 1.2	Addition of `Filter`
Servlet 2.2	August 1999	902, 903	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Servlet 2.1	November 1998	NA	Unspecified	First official specification, added `RequestDispatcher`, `ServletContext`
Servlet 2.0		NA	JDK 1.1	Part of Java Servlet Development Kit 2.0
Servlet 1.0	June 1997	NA		



CGI(Common Gateway Interface):-

- ✓ *Before servlet CGI(Common Gateway Interface) scripting language was popular at server side programming.*
- ✓ *In CGI the separate process will be created for each request. Creating & destroying process for every request is costly. If number of requests increases the performance of the system goes down, due to this the CHI fail to deliver scalable applications.*
- ✓ *Two processes never share same address space hence there is no chance of concurrent problems in CGI applications.*
- ✓ *CGI programming is platform dependent we can write code in different languages C,CPP,Perl..etc*
- ✓ *CGI uses high memory & low performance*

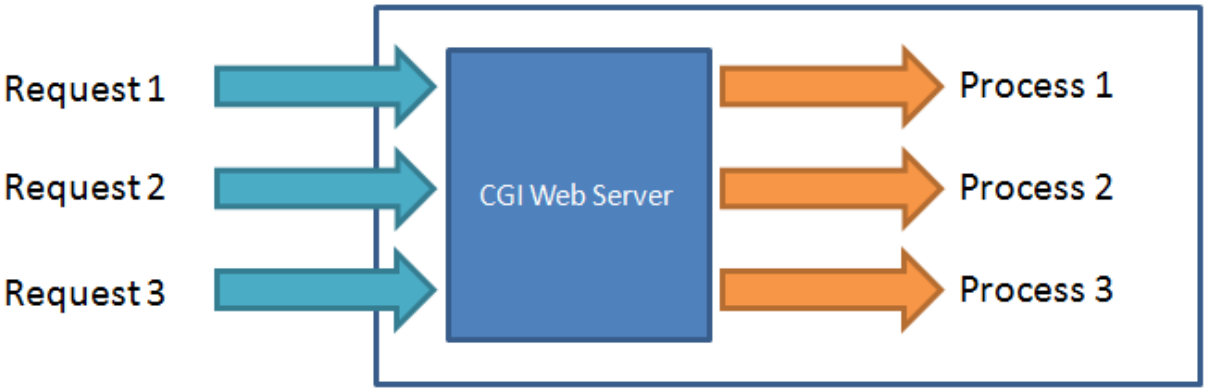
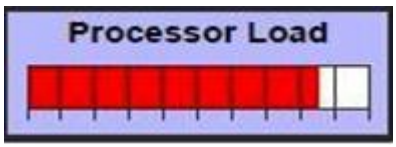
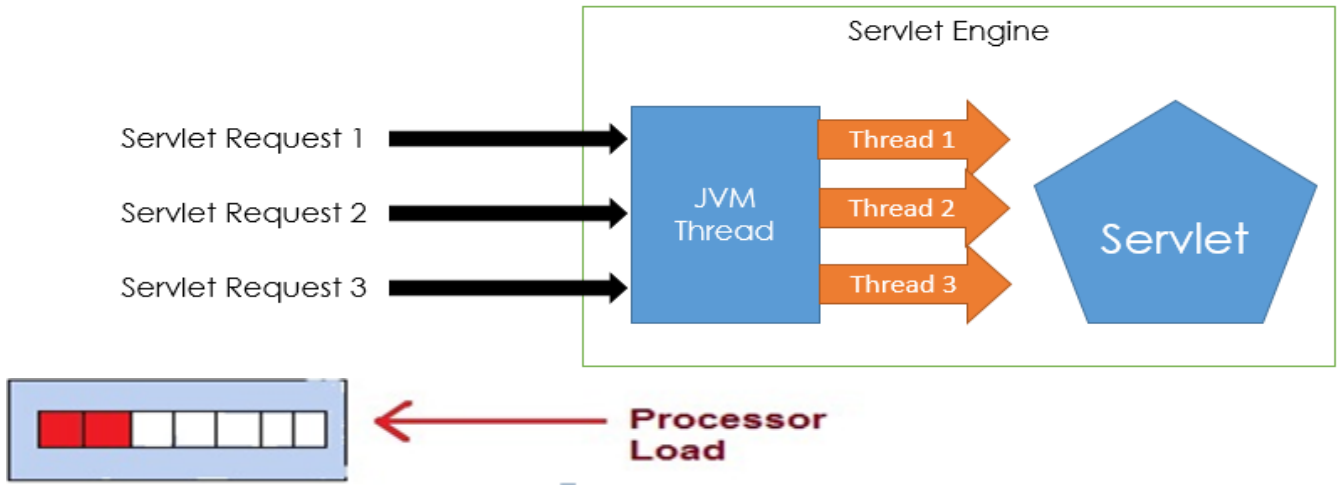


FIG : CGI Request Processing

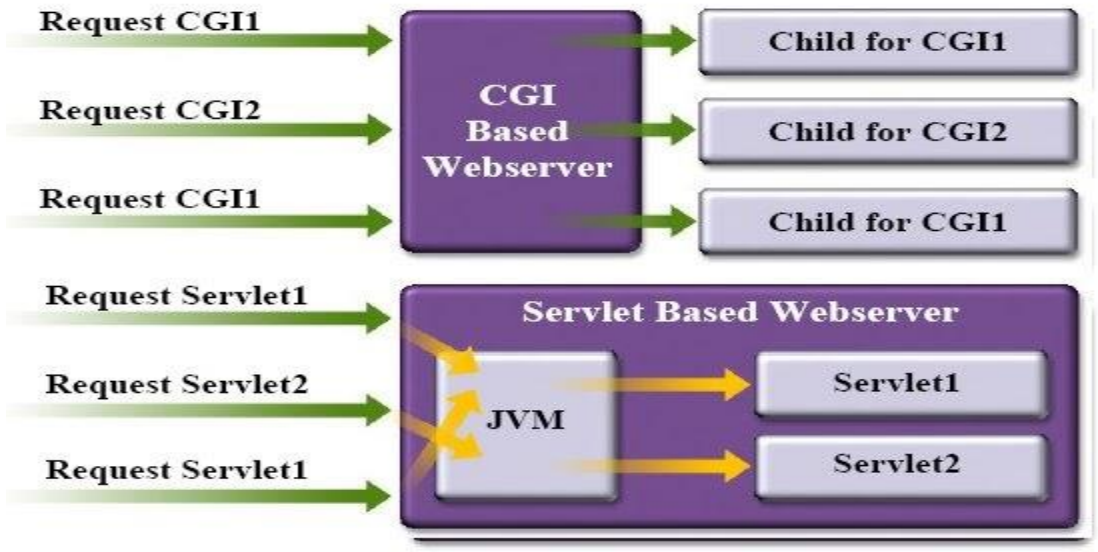


Advantages of using Servlets:-

1) *Less response time because each request runs on separate thread but not process.*
    a. *Threads have lot of benefits over process like,*
        i. *Threads are sharing common memory area.*
        ii. *The thread is light weight.*
        iii. *Costs of communication between threads are low.*
2) *Creating & destroying new thread for every request is not expensive, hence even number of requests are increases the performance may not goes down. Due to this servlets can deliver scalable products.*
3) *Since all threads are crated on common servlet object, there may be chance of concurrence problems in servlet applications, hence appropriate steps to taken.*
4) *Servlets are robust & object oriented.*
5) *Servlets and platform and system independent, the web application developed with Servlet can be run on any standard web container such as Tomcat, JBoss, Glassfish servers and on operating systems such as Windows, Linux, Unix, Solaris, Mac etc.*



CGI vs. Servlets:



Technology vs. Framework:

Technologies : J2SE J2EE J2ME
Frameworks : struts, spring , hibernate,jsf…etc

- ✓ *Frameworks are developed based on technologies source code.*
  Jdbc ---> hibernate
  Servlets ---> structs
  Jsp ---> jsf
  EJB's ---> Spring

- ✓ *The framework will support all the features of technologies,*
  - *It supports extra features.*
  - *It overcomes the limitations of technologies.*

- ✓ *The technologies will give good performance when compare to technologies.*

- ✓ *When we develop the application by using technology, the code is duplicated & length increased but when we develop the application by using framework it reduce the code duplication & length of the code because framework having more predefined support.*

- ✓ *The frameworks are flexible to use.*

Different types of applications:-

- ✓ *Standalone applications*
  Developed by using j2se
  Contains main method called by JVM
  Used extension .java .class ***.jar***

- ✓ *Web-applications*
  Developed by using j2ee(servlets & jsp) technology : structs JSF framework
  No-main method contains life cycle methods(init() service() destroy()) called by server
  Used extension ***.war***

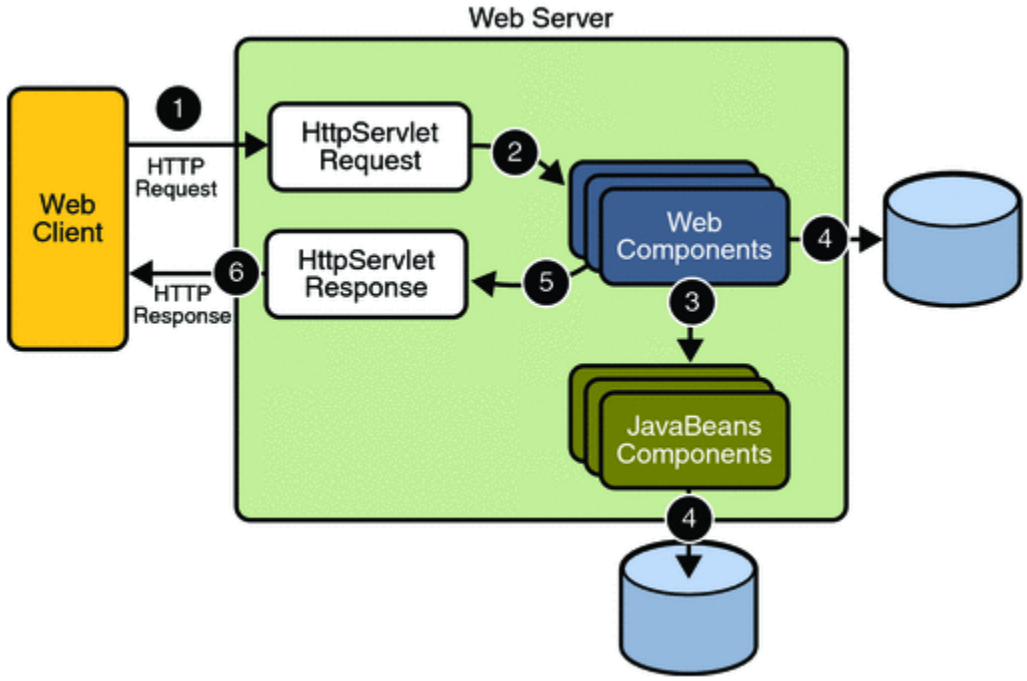
- ✓ *Enterprise applications*
  Developed by using EJB technology & spring frame work
  Used extension ***.ear***

- ✓ *Distributed applications*
  Developed by using RMI , web-services v, Micro services
  Used extension ***.aar*** *application archive file*

- ✓ *Mobile applications*
  Developed by using android ,IOS
  Different types of mobile apps
  - *Native apps: Installed & doesn't required internet Conn to run : candy crush*
  - *Hybrid apps : installed & required internet connection to run : facebook*
  - *Web apps : run through web browser*

  Used extension ***.apk****android application package*

Web applications:-

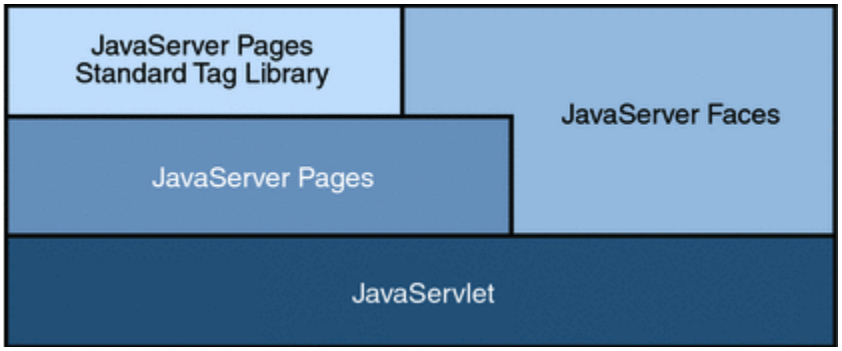


- ✓ *In above diagram the web components are either Java servlets, JSP pages..etc*

The six steps are listed below.

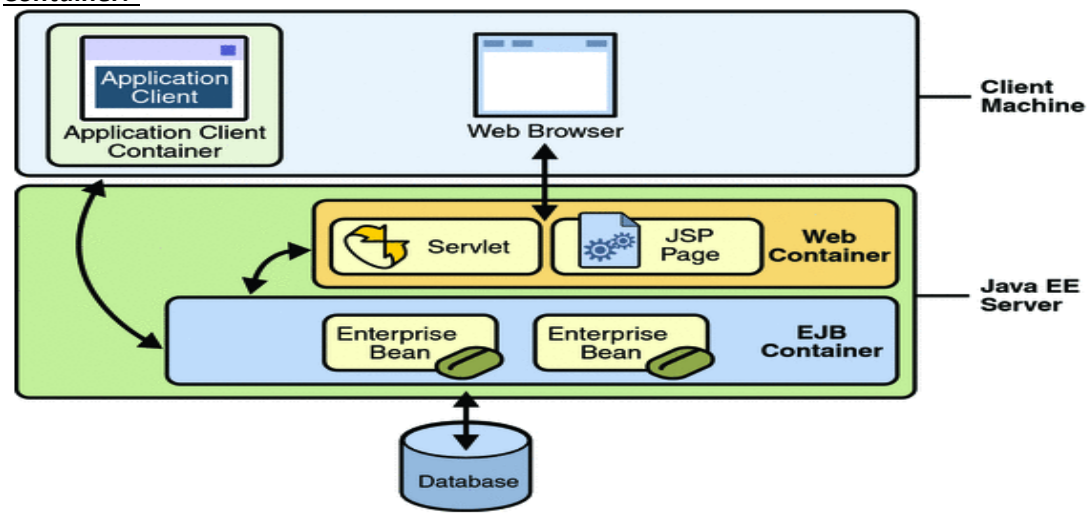
1. *Client sends the Http request to the web server.*
2. *The HttpServletRequest object is delivered to the web component.*
3. *The web components are interacting with database to generate dynamic content.*
4. *The web components are interacting with java beans to generate dynamic content.*
5. *The web componenet generate the response in HttpServletResponse object.*
6. *The web server converts the HttpServletResponse into HTTP response and returns it to the client.*

Java web applications technologies:-

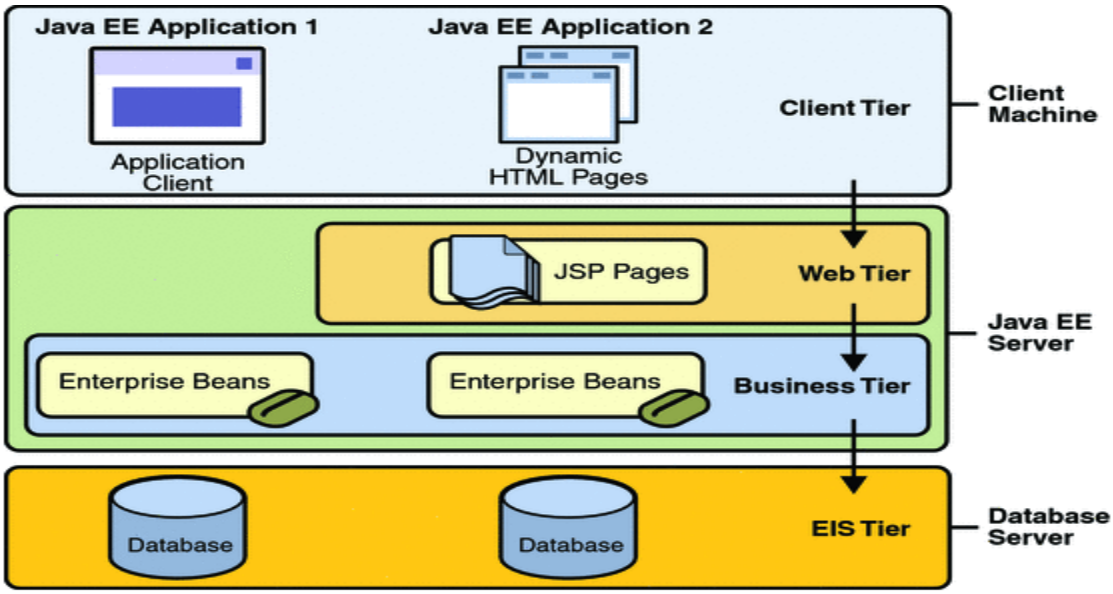


Notice that Java Servlet technology is the foundation of all the web application technologies.

Container:-



Distributed Multitier Java EE application:-



- ✓ *Client-tier components run on the client machine.*
- ✓ *Web-tier components run on the Java EE server.*
- ✓ *Business-tier components run on the Java EE server.*
- ✓ *Enterprise information system (EIS)-tier software runs on the EIS server.*

Servers & Vendors:

Server-name	***vendor***	***version***	***symbol***
Apache Tomcat	*Apache*	*9.0*	
Glassfish	*(Sun micro systems) Oracle*	*4.x*	
JBOSS	*Red Hat*	*7.x*	
Web logic	*Oracle*	*WebLogic 12cR2*	
Web sphere	*IBM*	*9.0*	

There are two types of servers

1. *Web server*
2. *Application server*

Web server:

Web server contains web container it support web application

Web container = servlet container + jsp engine

Servlet container is also known as : ***CATALINA , servlet engine***

Jsp engine is also known as : ***jasper***

Ex: jetty

Application Server:

Application server contains both web container & EJB container support both web-applications & EJB applications

Ex: glassfish, web logic , tomcat ..etc

Tomcat installation:
Step 1: download the tomcat latest version

Folder name	***Bescription***
Bin	*to start & stop the server use : **tomcat8.5.exe***
Config	

Protocols:

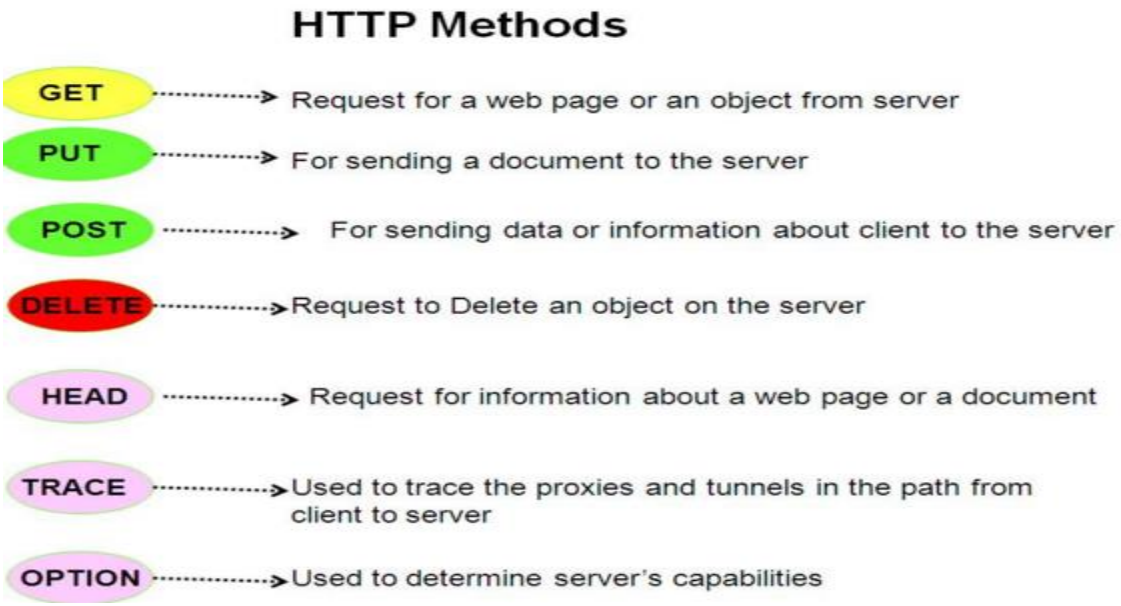
There are two types of protocols

1. *Transport protocol : transport purpose : Http , Https ,FTP*
2. *Messaging protocol : talking about data : SOAP*

HTTP (Hyper Text Transfer Protocol):-

- *HTTP is the foundation of data communication for the World Wide Web.*
- *HTTP functions as a request–response protocol in the client–server computing model.*
- *Client & server use HTTP protocol to provide the secure socket connection.*
- *It is similar to other internet protocols such as SMTP(Simple Mail Transfer Protocol) and FTP(File Transfer Protocol).*
- *HTTP protocol is stateless means each request is considered as the new request.*

There are seven Http request types but commonly used request types are get & post.



Request Format vs. Response Format:

When we send the request to server by using client first Http protocol will trap that request it will prepare request format contains header & body fields

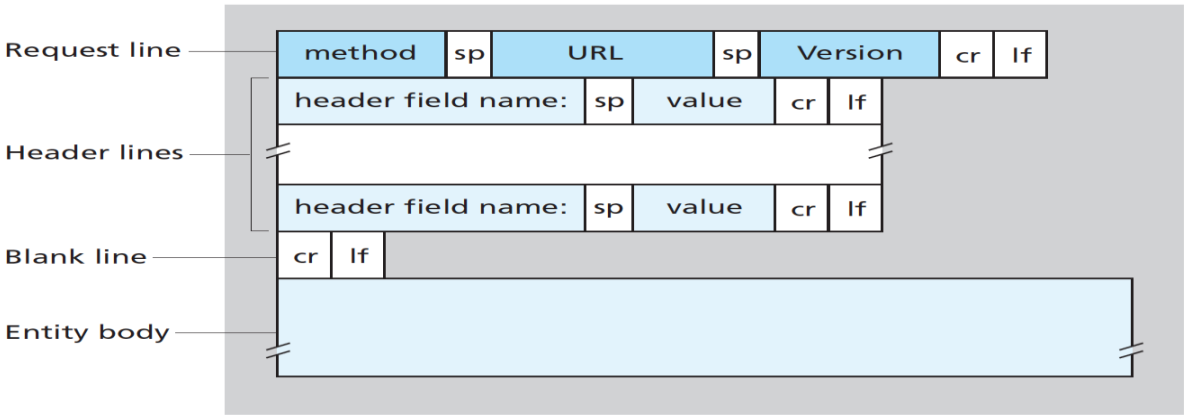
The header field contains request message headers

The body field contains the actual requested details.

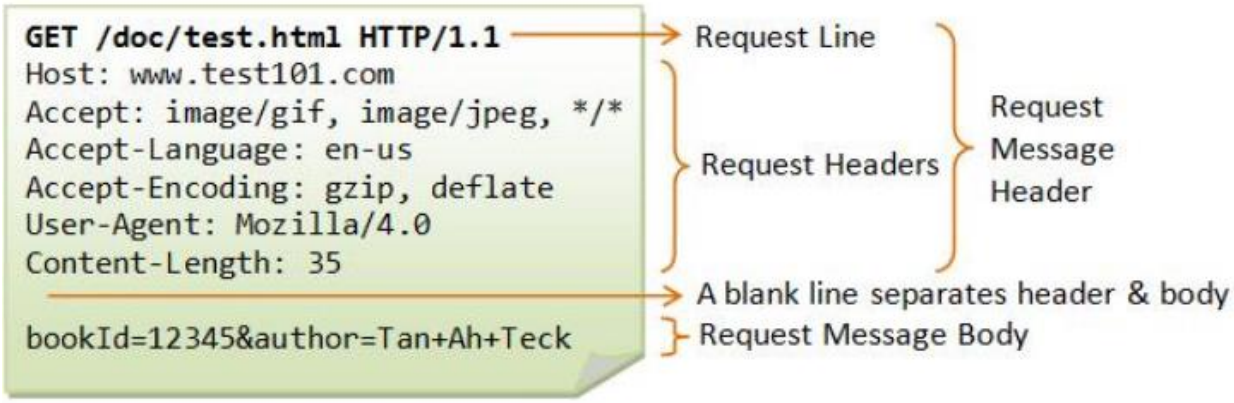
When the server is sending response to client browser first Http protocol will take that response the it will prepare response format contains header & body fields.

The header filed contains response message headers.

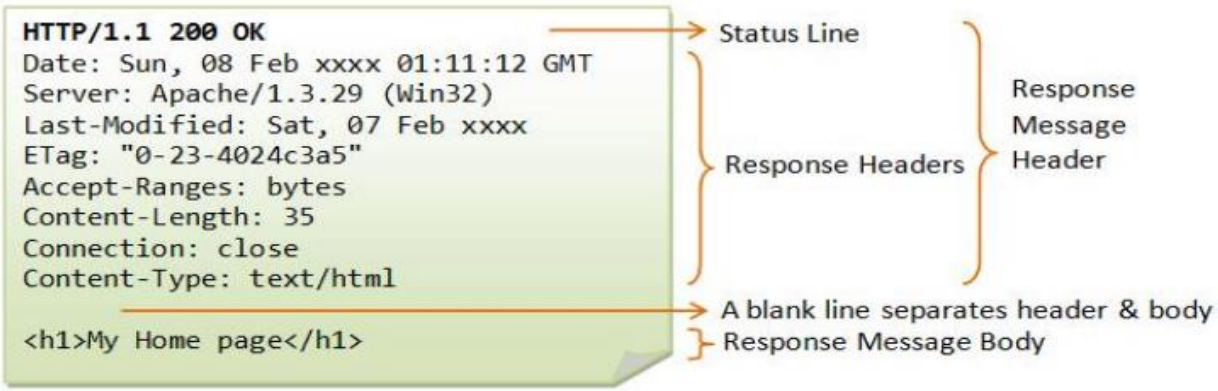
The body field contains actual response.



Request Format:



Response Format:

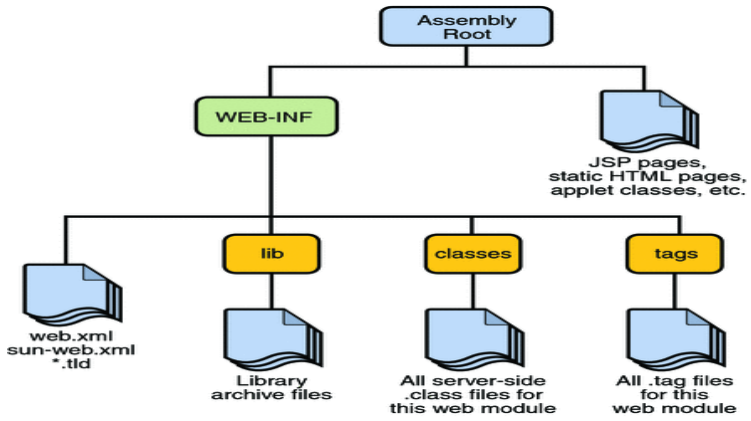


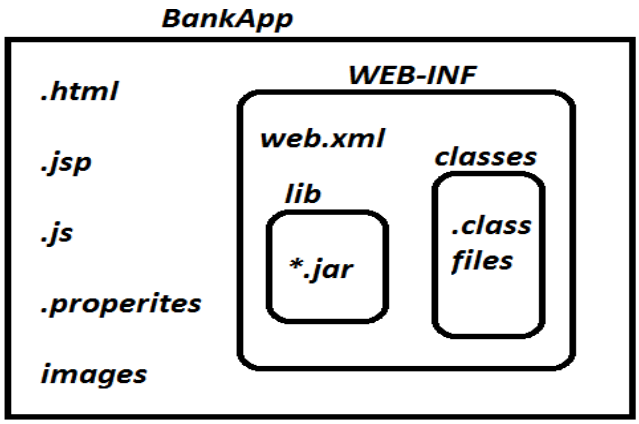
Steps to design a first application:

Step 1	*Prepare the web application directory structure.*
Step 2	*Prepare the form pages like html pages. (login.html, registration.html)*
Step 3	*Prepare the servlets.*
Step 4	*Prepare the web.xml file (Deployment Descriptor file).*
Step 5	*Deploy the project into server access the application by using url.*

Step 1: Web Application Directory Structure

If we want to design any web application then we have to prepare the following directory structure at server machine place the different web resources in different folders.





Web application directory structure contains two areas,

1. *Public area*
2. *Private area*

Public area:

The area which is in outside of WEB-INF folder and inside the application folder is treated as public area. If we deploy any resource under public area then client is able to access that resource by using its name directly.

Private area:

In web application directory structure, WEB-INF folder and its internal is treated as private area. If we deploy any resource under private area then client is unable to access that resource by using its name directly. So access the private area elements by using url pattern specified in web.xml

Step 2: Prepare the from pages.

```
<html>
<body>
<form method=”get” action=". /LoginServlet">
        User name     :     <input type="text" name="uname"/><br/>
        Password      :     <input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

- ✓ *Form tag method attribute* ***<form method=”get”>*** *used to represent which type of request we are sending from client browser like get,post,put….etc*
- ✓ *Form tag action attribute* ***<form action=”url”>*** *used to represent url pattern of the particular servlet used to provide the mapping between form page & servlet.*
- ✓ *At server side to read the form page data use logical name of the text filed* ***name="uname"****.*

Step 3: Prepare Web Resources (Servlets)

There are three approaches to create a servlets in java

1) Implementing Servlet interface

```
class MyServlet implements Servlet
{
}
```

2) Extending GenericServlet (abstract class)

```
class MyServlet  extends GenericServlet
{
}
```

3) Extending HttpServlet (abstract class)

```
class MyServlet extends HttpServlet
{
}
```

Servlet predefined support in the form of two packages

1. *Javax.servlet*
2. *Javax.servlet.http*

Servlet & GenricServlet present in ***javax.servlet*** *package*
HttpServlet class present in ***javax.servlet.http package.***

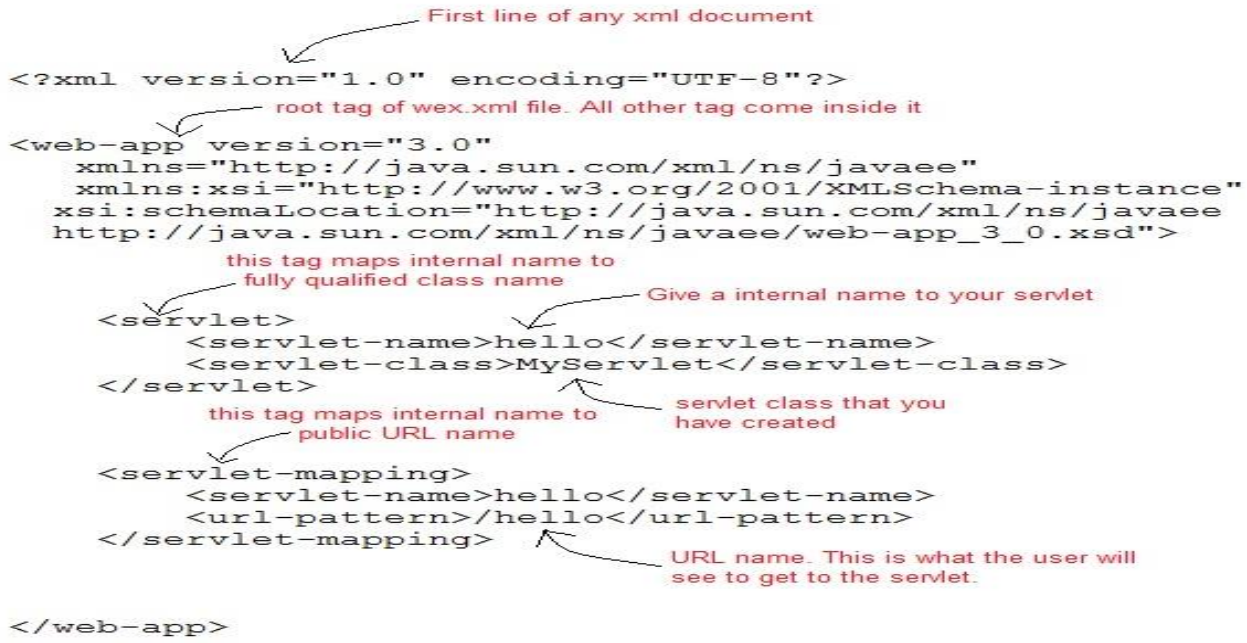
Step 4: Deployment Descriptor file or web.xml or mapping file.

- ✓ *It is used to provide the mapping between html pages to corresponding servlets.*
- ✓ *Deployment Descriptor is web.xml file, it can be used to provide the metadata about the present web application required by the container in order to perform a particular server side action.*

In web applications, web.xml file include the following configuration

Welcome Files Configuration	***<welcome-file>***
Load On Startup Configuration	***<load-on-startup>***
Display Name Configuration	***<display-name>***
Description	***<Description>***
Servlets Configuration	***<servlet>*** ***<servlet-mapping>***
Filters Configuration	***<Filter>*** ***<FilterMapping>***
Jsp configuration	***<jsp-file>***
Listeners Configuration	***<Listener>***
Context Parameters Configuration	***<context-param>***
Initialization Parameters Configuration	***<init-param>***
Session Time Out Configuration	***<session-timeout>***
Error Page Configuration	***<error-code>***
Tag Library Configuration	***<taglib-uri>***

If we want more info about web.xml check this web-site
https://docs.oracle.com/cd/E13222_01/wls/docs81/webapp/web_xml.html#1013555



web.xml:

```
<web-app>
  <servlet>
        <servlet-name>logical_name</servlet-name>
        <servlet-class>fully qualified name of servlet</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>logical_name</servlet-name>
        <url-pattern>urlpattern_name</url-pattern>
  </servlet-mapping>
</web-app>
```

ex:

```
<web-app>
    <servlet>
        <servlet-name>aaa</servlet-name>
        <servlet-class>com.dss.LoginServlet</servlet-class>
   </servlet>
   <servlet-mapping>
        <servlet-name>aaa</servlet-name>
        <url-pattern>/login</url-pattern>
   </servlet-mapping>
</web-app>
```

Step 4: Start the Server & access the web application by using url.

Open web browser and type the complete URL on address bar.

http://localhost:8888/app1/firstservlet

Application -1 : Implementing Servlet interface.

Servlet interface Predefined implementations: *servlet interface contains five methods*

```
public interface Servlet
{  public abstract void init(ServletConfig servletconfig)throws ServletException;
   public abstract ServletConfig getServletConfig();
   public abstract void service(ServletRequest servletrequest, ServletResponse servletresponse)
                                                                    throws ServletException, IOException;
   public abstract String getServletInfo();
   public abstract void destroy();
}
```

public void init(ServletConfig config) throws ServletException

- ✓ *The init() method is used to perform initialization of servlet.*
- ✓ *Life cycle methods, automatically invoked by the servlet container.*
- ✓ *The servlet container automatically calls this method immediately after servlet is being instantiated. While calling init() method servlet container has to create* ***ServletConfig*** *object then passing that object to init() method argument.*
- ✓ *The servlet initialization done only once so this method will be executed only once.*

public void service(ServletRequest req, ServletResponse resp)throws ServletException, IOException

- ✓ This method will be executed for every client request.
- ✓ This method contains following information:
  - *I. Reading Input data*
  - *II. Either Perform business logic or Invoke business logic.*
  - *III. View navigation (Calling response page)*
- ✓ *To call this method container has to create two objects****(Request,Response)*** *here*
  - o *Request object contains the requested details coming from client side.*
  - o *Response object contains actual response displayed in client browser.*

Public void destroy()

This method will be executed only once just before the servlet is removed from the container. The main purpose of this method is to perform cleanup activities.

Public ServletConfig getServletConfig()

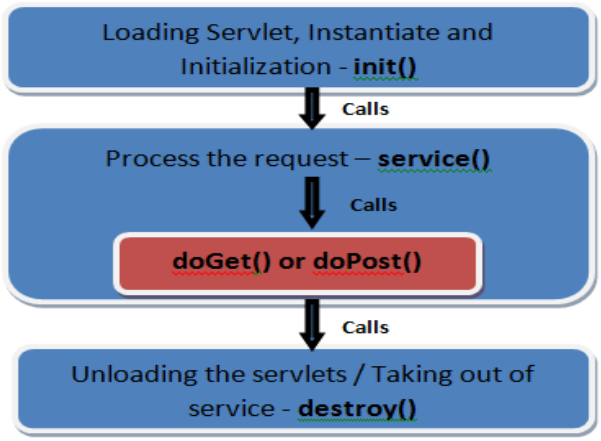
This method return ServletConfig object it contains configuration details of servlet. Every servlet is associated with ServletConfig object.(ServletConfig object specific to servlet)

Public String getServletInfo()

Returns basic information about servlet such as author, version, copyright, etc.

The servlet interface contains 5-methods , but three are life cycle methods

1. *Init()*
2. *Service()*
3. *Destroy()*



Specifying response content type:

To specify response content type to the client we have to use setContentType(---) method from ServletResponse.

It is a HTTP header to provide the description about what you are sending to browser window.

*public void setContentType(String MIME_TYPE) [**Multipurpose Internet Mail Extensions**]*

where MIME_TYPE may be

a. *text/html,*
b. *text/xml,*
c. *application/pdf,*
d. *application/msword*
e. *image/jpeg, img/jpg and so on.*

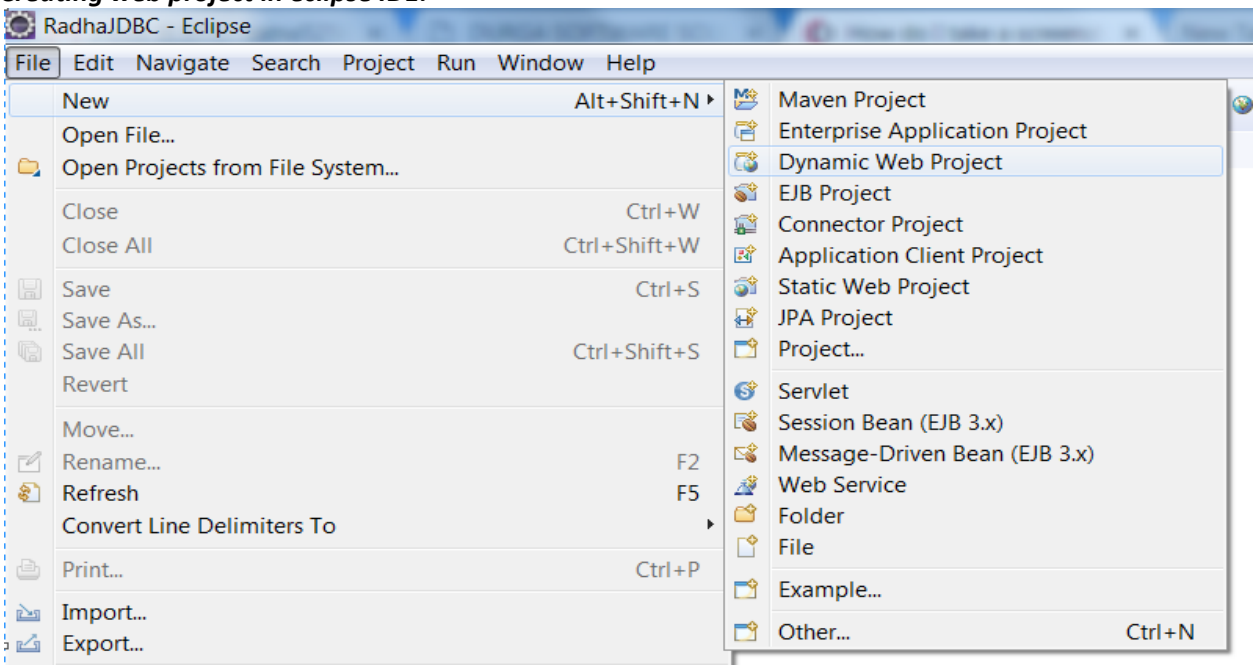
ex: ***response.setContentType("text/html");*** *The default content type in servlets is text/html.*

Adding response into response object:

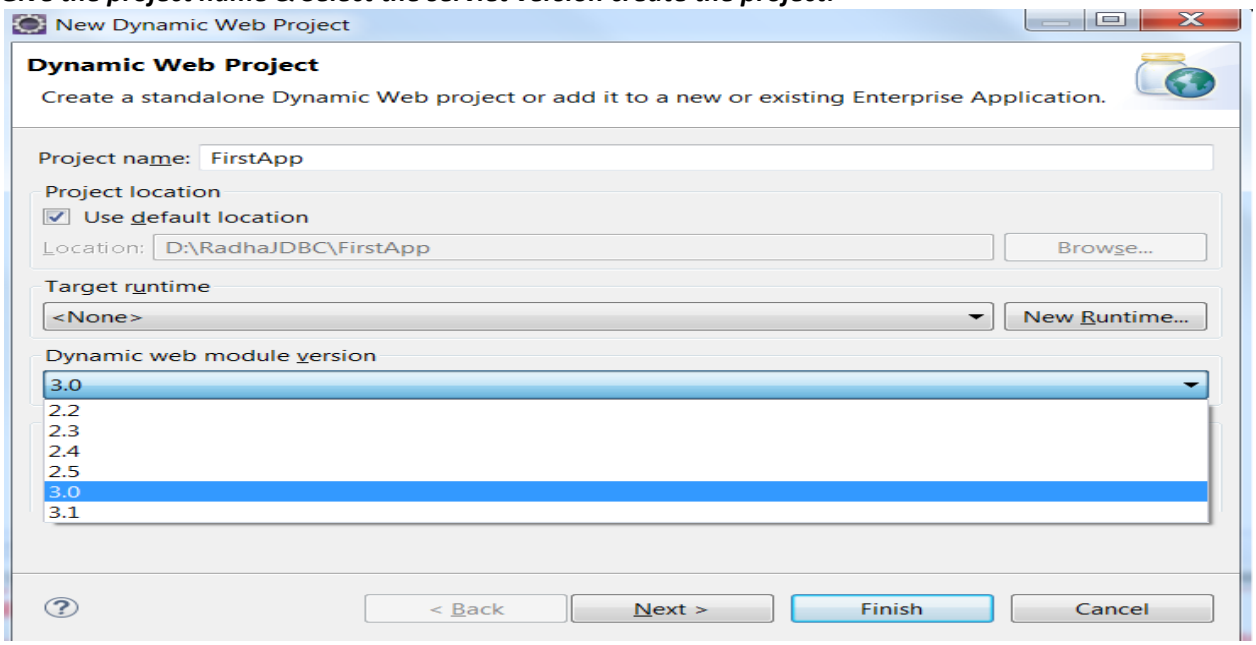
Initially the response object is empty hence to add the response into response object use PrintWriter object.

ex : *PrintWriter writer = response.getWriter();*
Writer.println("hi Ratan how r u….");

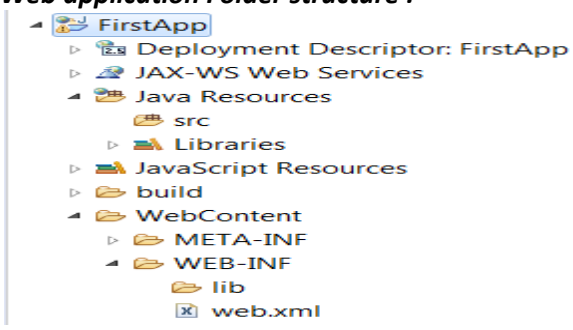
Creating web project in eclipse IDE:



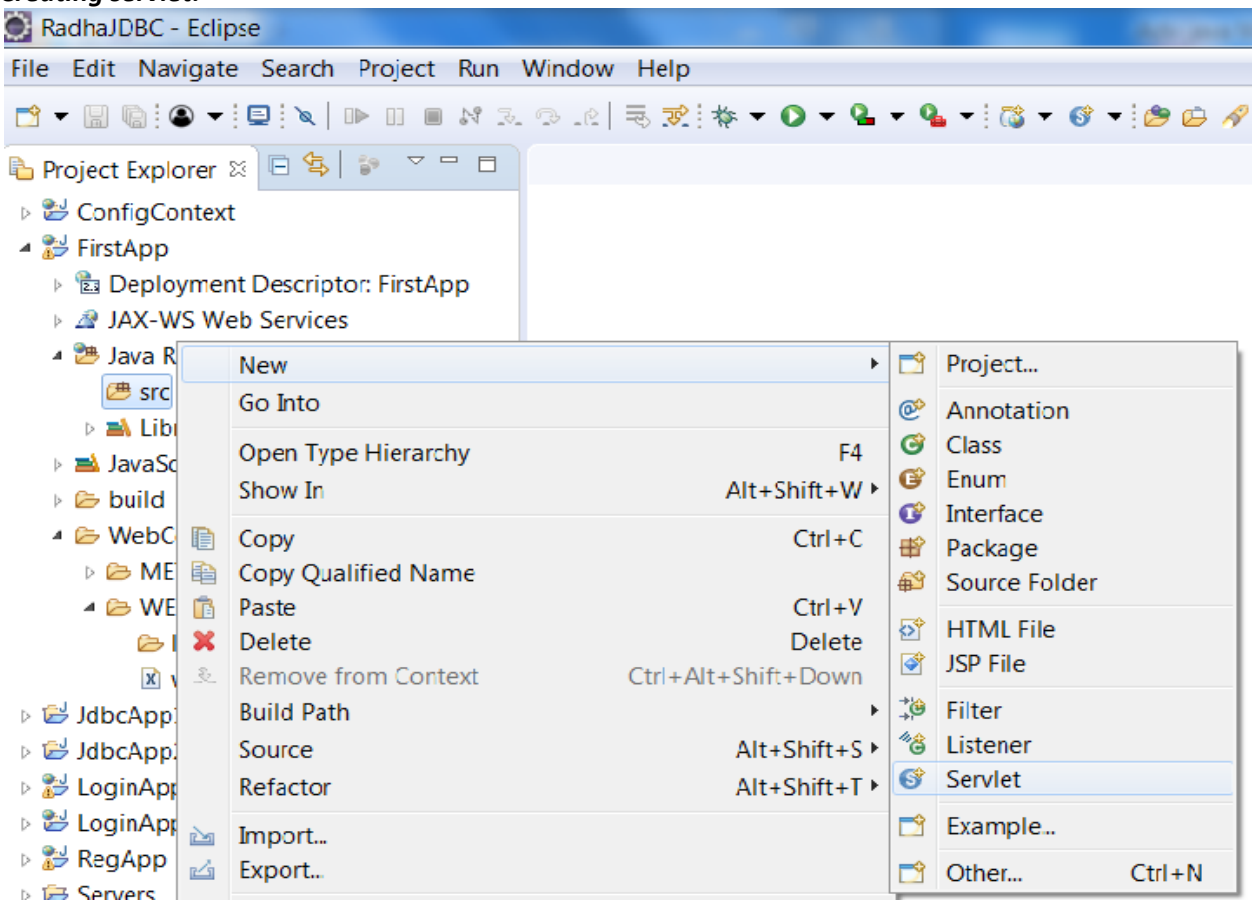
Give the project name & Select the servlet version create the project:



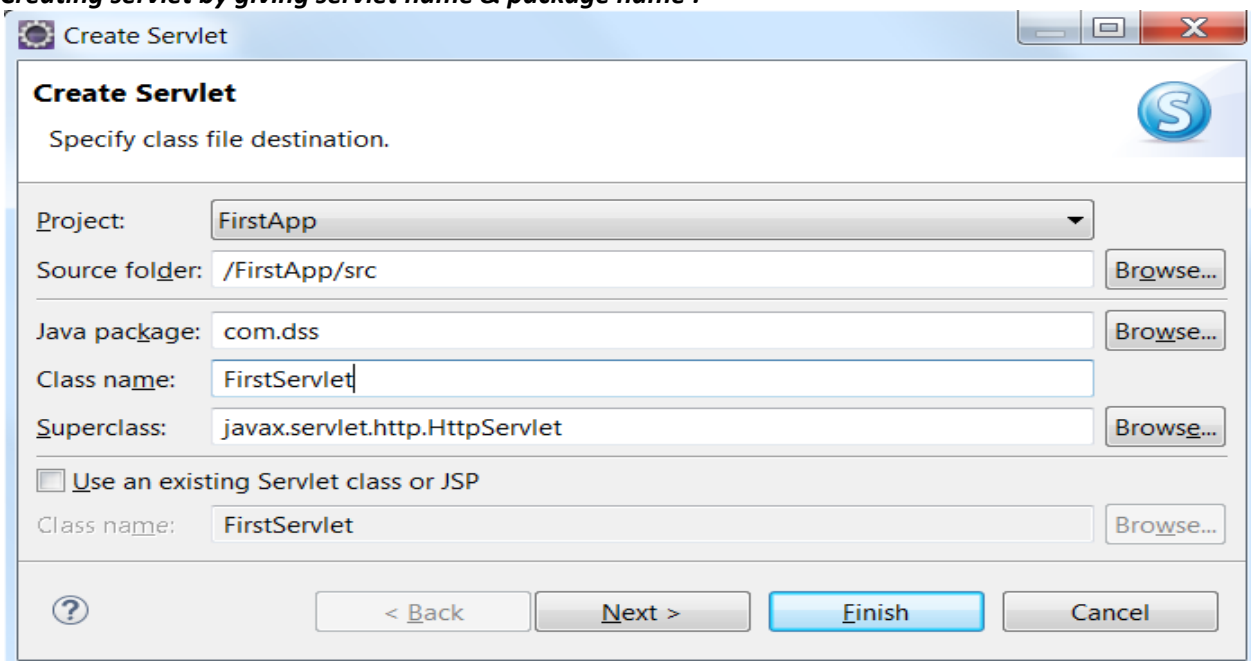
Web application Folder structure :



Creating servlet:



Creating servlet by giving servlet name & package name :



Approach 1:- *Take user defined class which implements Servlet Interface*

MyServlet.java:

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Servlet;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class FirstServlet implements Servlet {

	public void init(ServletConfig config) throws ServletException {
		System.out.println("init method");
	}
	public void service(ServletRequest request, ServletResponse response) throws ServletException,
	IOException {
		System.out.println("service method");
		response.setContentType("text/html");
		PrintWriter writer=response.getWriter();
		writer.println("Hi this is First Servlet Appliation");
	}

	public void destroy() {    System.out.println("destroy method");   }

	public ServletConfig getServletConfig() {
		System.out.println("getServletConfig method");
		return null;
	}
	public String getServletInfo() {
		System.out.println("getServletinfo method");
		return null;
	}
}
```

Web.xml:-

```
<web-app>
	<servlet>
		<servlet-name>FirstServlet</servlet-name>
		<servlet-class>  com.dss.FirstServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>FirstServlet</servlet-name>
		<url-pattern>/FirstServlet</url-pattern>
	</servlet-mapping>
</web-app>
```

Observation : Run the project

Right click on project --> run as --> Run on server --> we will get browser window type url

http://localhost:9999/ServletApp1/FirstServlet

http	*:*	*protocol*
localhost	*:*	*domain name*
9999	*:*	*port number*
ServletApp1	*:*	*project name*
FirstServlet	***:***	*Resource name (url pattern)*

Observation:

I above approach the developer must override 5-methods if required or not. But to write the logics of servlets we are using only service() method.

Observation:

When we send the request to user defined servlet class the container will create the object of servlet in this process it uses default constructor.

Inside the class if we are declaring argument constructor while instantiation the container will generate exception

```
public class FirstServlet implements Servlet {
   public FirstServlet(int a) {
   }
}
```

javax.servlet.ServletException: Error instantiating servlet class [com.dss.FirstServlet]

Observation:

*When we send the first request to Servlet the servlet initialization **(init())** done only once.*

*When we send second request directly it will dorequest processing **(service())***

Based on above two pointes we can decide the response time of both servlets is differing.

To give the same response time to every request performs servlet initialization during server startup.

*To perform servlet initialization during server startup use **<load-on-startup>** configuration in web.xml*

```
<servlet>
   <servlet-name>FirstServlet</servlet-name>
   <servlet-class>com.dss.FirstServlet</servlet-class>
   <load-on-startup>1</load-on-startup>
</servlet>
```

<load-on-startup> *is taking positive number it represents priority, if we are passing other than positive number the load-on-startup configuration ignored.*

Without <load-on-startup> :
During First Req
Loading Servlet
Instantiating Servlet
Initializing Servlet
service() method
From Second Request Onwards:
service() method

With <load-on-startup>1</load-on-startup>
During Server startup (or) At Application Deployment time i.e., before first request:
Loading Servlet
Instantiating Servlet
Initializing Servlet
From First Request onwards:
Service() method

Application 2: Creating servlet by extends GenericServlet

```
public interface Servlet
{       init()  serfvice() destroy()  getServletInfo()  getServletConfig()
}
abstract class GenericServlet implements Servlet
{       4-impl completed        1-abstract method service()
}
class MyServlet extends GenericServlet
{       Override 1-method service()
}
```

MyServlet.java:-

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.GenericServlet;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class SecondServlet extends GenericServlet {
        private static final long serialVersionUID = 1L;
          public SecondServlet() {
                super();
        }
   public void service(ServletRequest request, ServletResponse response)
                                                throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();

        writer.println("This is my SecondServlet Application");
        }
}
```

Web.xml:-

```
<web-app>
        <servlet>
                <servlet-name>SecondServlet</servlet-name>
                <servlet-class>com.dss.SecondServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>SecondServlet</servlet-name>
                <url-pattern>/SecondServlet</url-pattern>
        </servlet-mapping>
</web-app>
```

Application 3: creating servlet by extending HtpServlet

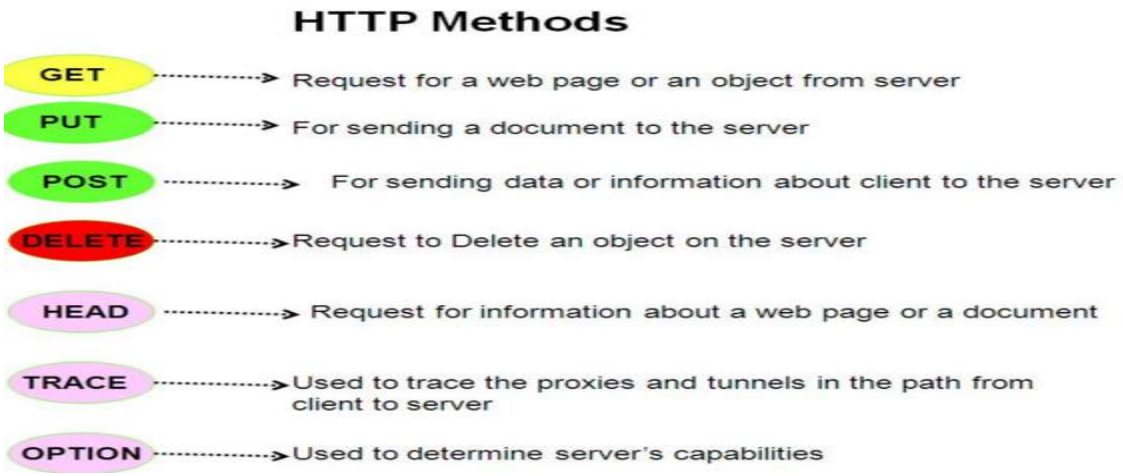
Limitation of second approach:

GenericServlet approach is protocol independent it means from the client side if we sending any type of request but at server side only service() method will be executed.

Advantage of third approach:

HttpServlet is protocol dependent it means it can handle only http type of request these are7 request types. Based on request type the corresponding method will be executed.

public interface Servlet
{ init() serfvice() destroy() getServletInfo() getServletConfig()
}
abstract class GenericServlet implements Servlet
{ 4-impl completed 1-abstract method ***service()***
}
Abstract Class HttpServlet extends GenericServlet
{ All methods implementations completed
}
class MyServlet extends HttpServlet
{ Override http request methods (total 7-Request types)
}



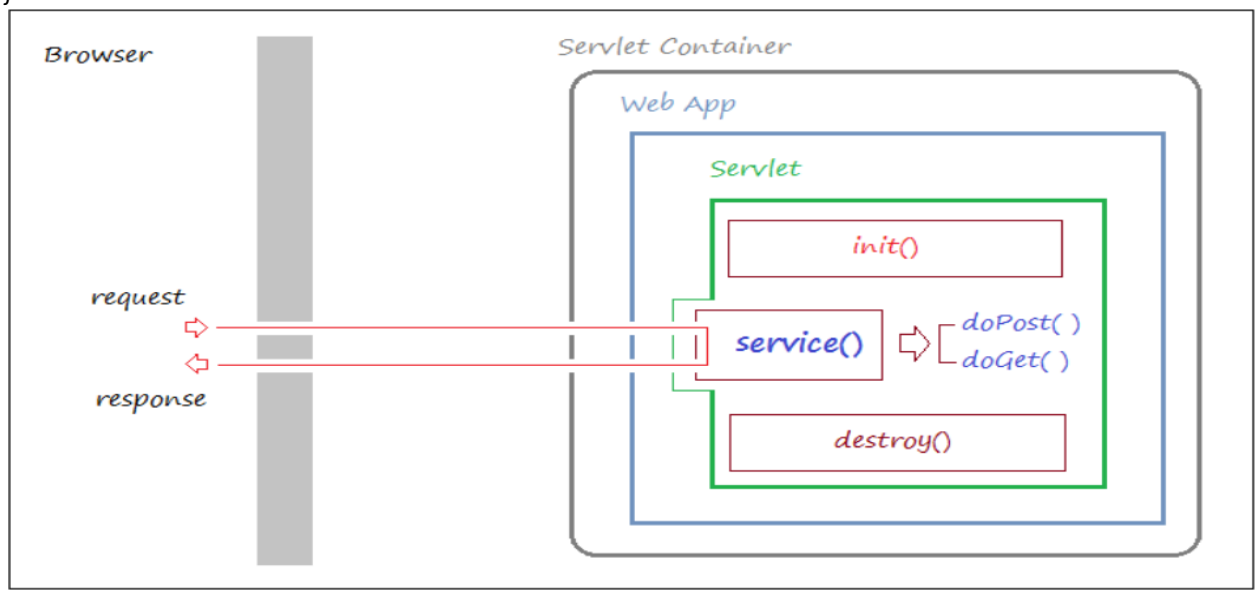
Note: *The default request type is* ***get*** *request.*
Note: *To specify the post request use form tag method attribute* ***<form method="post">***
Note: *HttpServlet is abstract class contains zero abstract methods.*

HttpServlet contains two service() methods :

1. *Pulbic service : Called by server : Converting normal req,res to http req,res*
2. *Protected service : Called by public service : Identifying req type forwarding req to method.*

Internal implementation of HttpServlet interface:

```
public abstract class HttpServlet extends GenericServlet
{       // converting normal request types to http request types then calling protected service
   public void service(ServletRequest req, ServletResponse res)  throws ServletException, IOException
   {
      HttpServletRequest request;
      HttpServletResponse response;
      try
      {  request = (HttpServletRequest)req;
         response = (HttpServletResponse)res;
      }
      catch(ClassCastException e)
      {  throw new ServletException("non-HTTP request or response");
      }
      service(request, response);
   }
// identifying request type then forwarding request to particular method
protected void service(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
   {  String method = req.getMethod();
      if(method.equals("GET")){            doGet(req, resp);          }
       else if(method.equals("HEAD")){    doHead(req, resp);         }
       else if(method.equals("POST")){         doPost(req, resp);    }
      else if(method.equals("PUT")){        doPut(req, resp);         }
      else if(method.equals("DELETE")){      doDelete(req, resp);  }
      else if(method.equals("OPTIONS")){        doOptions(req, resp);}
      else if(method.equals("TRACE")){       doTrace(req, resp);   }
      else
        String errMsg = lStrings.getString("http.method_not_implemented");
   }
}
```



```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.Ht1tpServletResponse;

publicclass HttpServletDemo extends HttpServlet {
        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer=response.getWriter();
                writer.print("<b>hi ratan how r u</b>");
                writer.print("<b>this is HttpServletDemo GET request example</b>");
        }
}
```

Case : doPost() : *To specify the post request use form tag method attribute* ***<form method="post">***

form.html:

```
<html>
<body>
<form method="post" action="./HttpServletDemo">
Click here to generate post request: <input type="submit" value="click">
</form>
</body>
</html>
```

```
        protectedvoid doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer=response.getWriter();
                writer.print("<b>hi ratan how r u</b>");
                writer.print("<b>this is HttoServletDemo  POST request example</b>");
        }
```

Web.xml:

```
<web-app>
        <servlet>
                <servlet-name>HttpServletDemo</servlet-name>
                <servlet-class>  com.dss.HttpServletDemo</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>HttpServletDemo</servlet-name>
                <url-pattern>/HttpServletDemo</url-pattern>
        </servlet-mapping>
</web-app>
```

Matching the url pattern :

1. Exact Match Method:

In Exact Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with forward slash("/") and pattern name may be anything.

```
<servlet-mapping>
   <servlet-name>FirstServlet</servlet-name>
   <url-pattern>/FirstServlet</url-pattern>
 </servlet-mapping>
```

ex:	http://localhost:9999/ServletApp1/FirstServlet	*Valid*
	http://localhost:9999/ServletApp1/abc/FirstServlet	*Invalid*
	http://localhost:9999/ServletApp1/FirstServlet/abc	*Invalid*

***Note**: in this approach the particular url pattern belongs to particular servlet.*

2. Directory Match Method:

In this approach define the URL pattern in web.xml file, it must be prefixed with forward slash("/") and it must be terminated with "".*

Case 1:

```
 <servlet-mapping>
   <servlet-name>FirstServlet</servlet-name>
   <url-pattern>/abc/*</url-pattern>
 </servlet-mapping>
```

ex:

http://localhost:9999/ServletApp1/abc/xyz
http://localhost:9999/ServletApp1/xyz/abc
http://localhost:9999/ServletApp1/abc/xyz/123

Case 2:

```
 <servlet-mapping>
   <servlet-name>FirstServlet</servlet-name>
   <url-pattern>/abc/xyz/*</url-pattern>
 </servlet-mapping>
```

ex:

http://localhost:9999/ServletApp1/abc/xyz
http://localhost:9999/ServletApp1/xyz/abc
http://localhost:9999/ServletApp1/abc/xyz/abc

***Note:** Multiple requested url patterns are trapped by same servlet with specific suffix(directory name) use this approach.*

3. Extension Match Method:

In Extension Match Method, we have to define an URL pattern in web.xml file, it must be prefixed with "" and it must be terminated with a particular extension.*

```
<servlet-mapping>
   <servlet-name>FirstServlet</servlet-name>
   <url-pattern>*.do</url-pattern>
 </servlet-mapping>
```

ex:	http://localhost:9999/ServletApp1/ratan.do	*valid*
	http://localhost:9999/ServletApp1/afdas/dsad.do	*valid*
	http://localhost:9999/ServletApp1/afdas/dsad/123.do	*valid*
	http://localhost:9999/ServletApp1/123.html	*valid*

***Note:** All the requested url patters are trapped by single servlet use this approach.*

ServletRequest<<interface>>

For every request, the Servlet container creates ServletRequest object.
It contains complete information about HTTP Request text message.

Methods:

a) getParameter(String name)
b) Enumeration getParameterNames()
c) getContentLength()
d) getContentType() : (MIME types)
e) getInputStream()
f) getLocales() : (en_US)
g) getProtocol() : http
h) getScheme() : (http/1.1)
i) getServerName() : (localhost)
j) getServerPort() : (9090)

ServletResponse<<interface>>

The Servlet container creates ServletResponse object. It is used to send HTTP Response message to web browser.

Methods:

a) getOutputStream() : For sending binary data
b) getWriter() : For sending character data
c) setContentType(String type) : Specifying MIME type of the response.

Servlets Flow of Execution:

When we start the server the main job of container is to recognize each and every web application and to prepare ServletContext object to each and every web application.

While recognizing web application container will recognize web.xml file under WEB-INF folder then perform loading, parsing and reading the content of web.xml file.

While reading the content of web.xml file, if container identifies any context data in web.xml file then container will store context data in ServletContext object at the time of creation.

After the server startup when we send a request from client to server protocol will pick up the request tehn perform the following actions.

1. *Protocol will establish virtual socket connection between client and server as part of the server IP address and port number which we specified in the URL.*
2. *Protocol will prepare a request format having request header part and body part, where header part will maintain request headers and body part will maintain request parameters provided by the user.*
3. *After getting the request format protocol will carry request format to the main server.*

Upon receiving the request from protocol main server will check whether the request data is in well-formed format or not, if it is in well-formed then the main server will bypass request to container.

Upon receiving the request from main server container will pick up application name and resource name from request and check whether the resource is for any html page or Jsp page or an URL pattern for a servlet.

If the resource name is any html page or Jsp page then container will pick up them application folder and send them as a response to client.

If the resource name is an URL pattern for a particular servlet available under classes folder then container will go to web.xml file identifies the respective servlet class name on the basis of the URL pattern.

After identifying the servlet class name from web,xml file container will recognize the respective servlet .class file under classes folder then perform the following actions.

Step 1: Servlet Loading:

Here container will load the respective servlet class byte code to the memory.

Step 2: Servlet Instantiation: Here container will create a object for the loaded servlet.

Step 3: Servlet Initialization:

Here container will create ServletConfig object and access init(_) method by passing ServletConfig object reference.

Step 4: Creating request and response objects(Request Processing):

After the servlet initialization container will create a thread to access service(_,_) method, for this container has to create request and response objects.

Step 5: Generating Dynamic response:

By passing request and response objects references as parameters to the service(_,_) method then container will access service(_,_) method, executes application logic and generate the required response on response object.

Step 6: Dispatching Dynamic response to Client:

When container generated thread reaching to the ending point of service(_,_) method then container will keep the thread in dead state, with this container will dispatch the dynamic response to main server from response object, where main server will bypass the response to the protocol.

When protocol receives the response from main server then protocol will prepare response format with header part and body part, where header part will manage all the response headers and body part will manage the actual dynamic response.

After getting response format protocol will carry that response format to the client.

Step 7: Destroying request and response objects:

When the response is reached to the client protocol will terminate the virtual socket connection between client and server, with this container destroy the request and response objects.

Step 8: Servlet Deinstantiation:

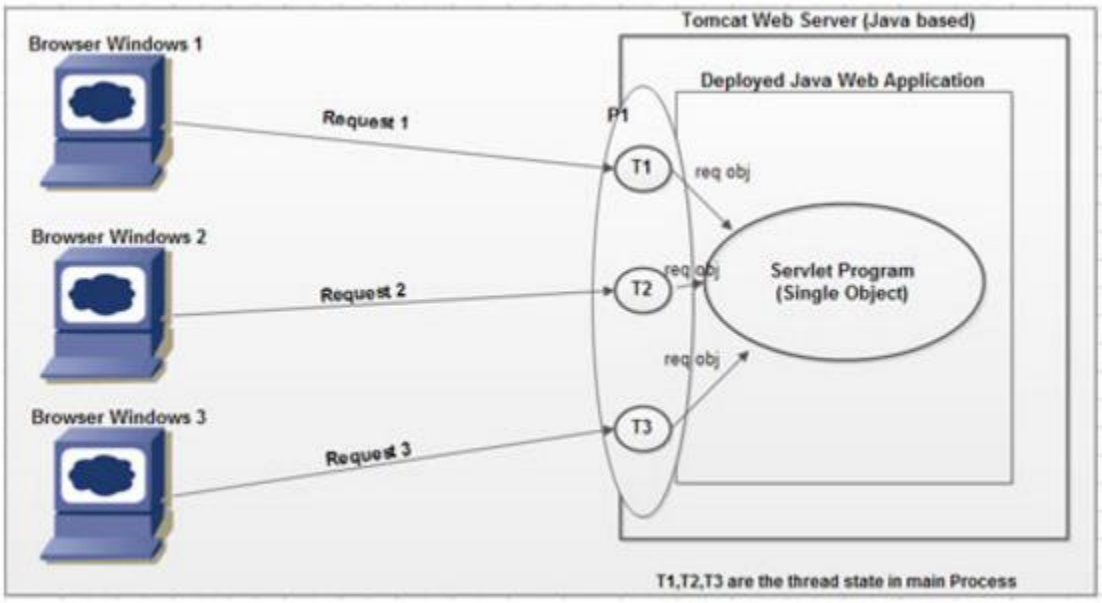
When container destroy request and response objects then container will go to the waiting state depends on the container implementation, if container identifies no further request for the same resource then container will destroy servlet object.

Note: In servlet execution, container will destroy ServletConfig object just before destroying the servlet object.

Step 9: Servlet Unloading:

After servlet deinstantiation container will eliminate the loaded servlet byte code from operational memory.

Step 10: Destroying ServletContext object:



In the above servlet life cycle, all the objects like request, response and ServletConfig are destroyed before servlet deinstantiation, but still Servlet object is available in memory.

In general ServletContext object will be destroyed at the time of server shut down.

First Approach to Design Servlets (Implementing Servlet interface):

If we want to design a servlet with this approach then we have to take an user defined class it should be an implementation class to Servlet interface.

Where the purpose of init(_) method to provide servlets initialization.

Note: In servlets execution, to perform servlets initialization container has to access init(_) method. To access init(_) method container has to create ServletConfig object.

ServletConfig is an object, it will manage all the configuration details of a particular servlet like logical name of servlet, initialization parameters and so on.

In servlet, the main purpose of service(_,_) method is to accommodate the complete logic and to process the request by executing application logic.

Note: In servlet, service(_,_) method is almost all same as main(_) method in normal Java application.

When we send a request from client to server then container will access service(_,_) method automatically to process the request, where the purpose of getServletConfig() method is to get ServletConfig object at servlet initialization.

Where getServletInfo() method is used to return generalized description about the present servlet.

Where destroy() method can be used to perform servlet reinstantiation.

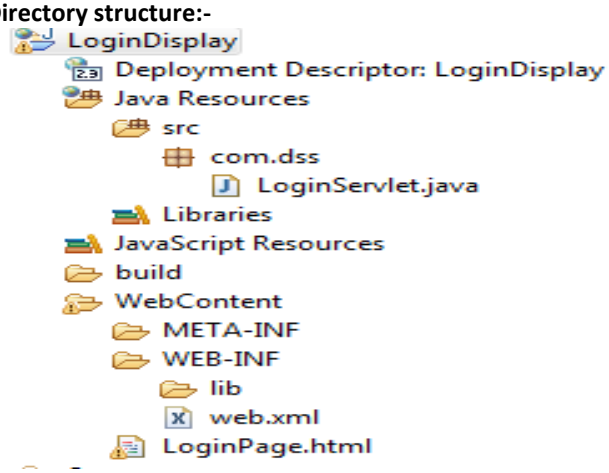
To prepare application logic in service(_,_) method we have to use the following conventions.

If we want to access the above servlet then we have to provide the following URL at client browser.

http://localhost:8080/loginapp/login

Application : Display login details

Directory structure:-



login.html:

```
<html>
<head>
<h1><center>Login page</center>
</head>
<body>
<form method="get" action="./LoginServlet">
<pre>
user name<input type="text" name="uname"/>
password<input type="password" name="upwd"/>
<input type="submit" value="login"/>
</pre>
</form>
</body>
</html>
```

Web.xml:

```
<web-app>
	<welcome-file-list>
		<welcome-file>login.html</welcome-file>
	</welcome-file-list>
	<servlet>
		<servlet-name>LoginServlet</servlet-name>
		<servlet-class>com.dss.LoginServlet</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>LoginServlet</servlet-name>
		<url-pattern>/LoginServlet</url-pattern>
	</servlet-mapping>
</web-app>
```

LoginServlet.java:

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclass LoginServlet extends HttpServlet {

	protectedvoid doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

		String uname = request.getParameter("uname");
		String upwd = request.getParameter("upwd");

		if(uname.equals("durga")&&upwd.equals("ratan"))
		{	writer.println("Login Success...."+uname);
		}
		else
		{	writer.println("Login Fail try with valid credentials");
		}
	}
}
```

Note : At server side to read the data from client side form page we are using logical name of fields, if logical name is wrong we will get ***null*** *value.*

doGet() vs doPost():

Get Request:

1. *The default request type is **get** request.*
2. *In GET request the user entered information is appended to the URL in a query string.*
3. *The query string visible in the URL pattern so GET request is able to provide less request.*
4. *By using GET request we the client is able to send limited amount of data. GET request doesn't have body in the request format hence we are able to send the all parameter values in the form of header part so by using GET request we are able to send limited amount of data(maximum 256 number of characters at a time)*
5. *In general in web applications GET request can be used to get the information from the server.*

*<form method="**get**" action="./LoginServlet">*
http://localhost:9999/LoginApp/LoginServlet?uname=ratan&upwd=durga

Post Request:

1. *POST is not default request. To specify the post request use **<form method="post">**.*
2. *POST request the user entered information is sent as data not appended to URL.*
3. *POST is send the data client to server hence it is able to provide very good security .*
4. *POST request contains body part hence we are able to send the large amount of data client to server by using body part of POST request.*
5. *In general in web applications POST request can be used to send the data to the server.*

*<form method="**post**" action="./LoginServlet">*
http://localhost:9999/LoginApp/LoginServlet

Application: Registration application form

reg.html:

```
<html>
<head>
<center>Registration</center>
</head>
<body>
<form method="post" action="./Registration">
<pre>
                    Name          <input type="text" name="uname"/>
                    Password      <input type="password" name="upwd"/>
          Qualification   <input type="checkbox" name="uqual" value="BSC"/>BSC
                          <input type="checkbox" name="uqual" value="MCA"/>MCA
                    <input type="checkbox" name="uqual" value="M TECH"/>M TECH

          Gender        <input type="radio" name="ugen" value="MALE"/>MALE
                        <input type="radio" name="ugen" value="FEMALE"/>FEMALE

                    Technologies  <select name="utech" size="3" multiple>
                                  <option value="C">C</option>
                                  <option value="C++">C++</option>
                                  <option value="JAVA">JAVA</option>
                                  </select>
     Comments<input type="textarea" name="ucom" rows="5" cols="25"></textarea>
                    <input type="submit"  value="Register"/>
</pre>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
    <display-name>Registration</display-name>
    <servlet>
        <servlet-name>Registration</servlet-name>
        <servlet-class>com.dss.Registration</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>Registration</servlet-name>
        <url-pattern>/Registration</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>Registration.html</welcome-file>
    </welcome-file-list>
</web-app>
```

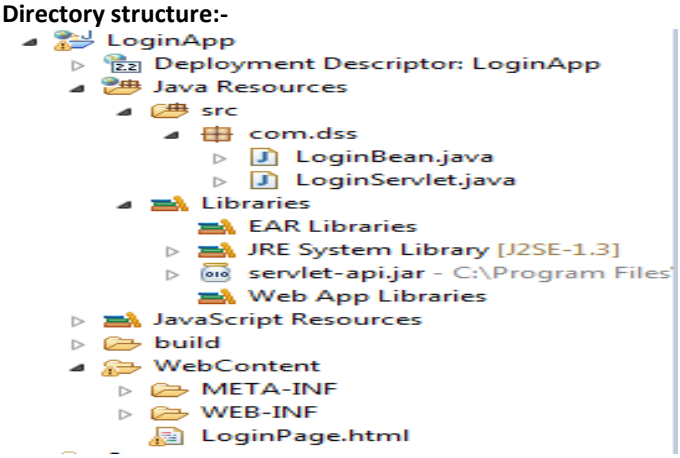
Registration.java:-

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclassRegistrationextends HttpServlet {
        protectedvoid doPost(HttpServletRequest request, HttpServletResponse response)
                                        throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String uname=request.getParameter("uname");
                String upwd=request.getParameter("upwd");
                String[] uqual=request.getParameterValues("uqual");
                String ugen=request.getParameter("ugen");
                String[] utech=request.getParameterValues("utech");
                String ucom=request.getParameter("ucom");
                out.println("<html>");
                out.println("<body bgcolor='lightgreen'>");
                out.println("<center><b><font size='6'>");
                out.println("Name....."+uname);
                out.println("<br><br>");
                out.println("Password......"+upwd);
                out.println("<br><br>");
                out.println("Qualification<br><br>");
                for(int i=0;i<uqual.length;i++)
                {       out.println(uqual[i]);
                        out.println("<br><br>");
                }
                out.println("Gender...."+ugen);
                out.println("<br><br>");
                out.println("Technologies<br><br>");
                for(int i=0;i<utech.length;i++)
                {       out.println(utech[i]);
                        out.println("<br><br>");
                }
                out.println("Comments......"+ucom);
                out.println("<br><br>");
                out.println("Congratulations....."+uname);
                out.println("<br><br>U R Registration Success");
                out.println("</font></b></center></body></html>");
        }
```

}

LoginServelt application used to store the data into the database:-

Directory structure:-



LoginPage.html:-

```
<html>
<head>
<h1><center>login page</center>
</head>
<body>
<form method="get" action="./LoginServlet">
<pre>
user name<input type="text" name="uname"/>
password<input type="password" name="upwd"/>
<input type="submit" value="login"/>
</pre>
</form>
</body>
</html>
```

Web.xml:

```
<web-app>
        <display-name>App1</display-name>
        <servlet>
                <servlet-name>LoginServlet</servlet-name>
                <servlet-class>com.dss.LoginServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>LoginServlet</servlet-name>
                <url-pattern>/LoginServlet</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
                <welcome-file>LoginPage.html</welcome-file>
                </welcome-file-list>
```

```
</web-app>
```

LoginServlet.java:-

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
publicclass LoginServlet extends HttpServlet {
	String status;
protectedvoid doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer=response.getWriter();

		String uname=request.getParameter("uname");
		String upwd=request.getParameter("upwd");
		writer.println("<html>");
		writer.println("<body>");
		writer.println("username is"+uname);
		writer.println("<br>");
		writer.println("user password is"+upwd);
		writer.println("</body>");
		writer.println("</html>");

		LoginBean lb=new LoginBean();
		try {
			lb.checkLogin(uname,upwd);
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
	protectedvoid doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
	}

}
```

LoginBean.java:-

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

publicclass LoginBean {
	static Statement statement;
	static
	{
		try {
			Class.forName("sun.jdbc.odbc.JdbcOdbc");
			Connection
connection=DriverManager.getConnection("jdbc:odbc:ratan","system","manager");
			statement=connection.createStatement();

		} catch (ClassNotFoundException e) {
			e.printStackTrace();
		} catch (SQLException e) {
			e.printStackTrace();
		}
		System.out.println("connection is created successfully");

	}
	publicvoid checkLogin(String uname, String upwd) throws SQLException {
		String query="insert into eee values('"+uname+"','"+upwd+"')";
				statement.executeUpdate(query);
				System.out.println("vales are inserted successfully");
	}

}
```

Example :-

public java.lang.String getParameter(java.lang.String name):-

- ✓ *If you want to read only one form field use getParamerter().*
- ✓ *Returns the value of a request parameter as a String, or null if the parameter does not exist.*

public java.util.Enumeration getParameterNames():-

- ✓ *Used to read all the form data at a time.*
- ✓ *Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.*

public java.lang.String[] getParameterValues(java.lang.String name):

- ✓ *Used to read the all the values at a time but the condition is all the fields should have the common name.*
- ✓ *Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.*

CustomerDetails.html:-

```
<html>
<body>
<form action="CustomerServlet" method="GET">
<p>First Name: <input type="text" name="firstName" /></p>
<p>Last Name:<input type="text" name="lastName" /></p>
<p>Mobile: <input type="text" name="mobile" /></p>
<p>E-Mail: <input type="text" name="email" /></p>
<p> Enter H.No.  <input type="text" name="t1"></p>
<p> Enter Street <input type="text" name="t1"></p>
<p> Enter Area   <input type="text" name="t1"></p>
<p><input type="submit" value="Register" /></p>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<welcome-file-list>
        <welcome-file>CustomerDetails.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>CustomerServlet</servlet-name>
<servlet-class>com.dss.CustomerServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>CustomerServlet</servlet-name>
<url-pattern>/CustomerServlet</url-pattern>
</servlet-mapping>
</web-app>
```

TestDb2.java: for connection pooling

```
package com.dss;

import org.apache.commons.dbcp.BasicDataSource;

public class TestDb2 {
        static BasicDataSource dataSource=null;
        static{
                dataSource = new BasicDataSource();
                dataSource.setUrl("jdbc:oracle:thin:@localhost:1521:xe");
                dataSource.setDriverClassName("oracle.jdbc.driver.OracleDriver");
```

```
                dataSource.setUsername("system");
                dataSource.setPassword("ratan");
                dataSource.setMaxActive(10);
        }
}
```

*Required jars for connection pool :**commons-dbcp.jarcommon-pool.jar** for third party connection pool.*
For oracle Database : ojdbc6.jar or ojdbc14.jar based on oracle version.
For mysql database : mysql-connector.jar
For servlets : servlet-api.jar

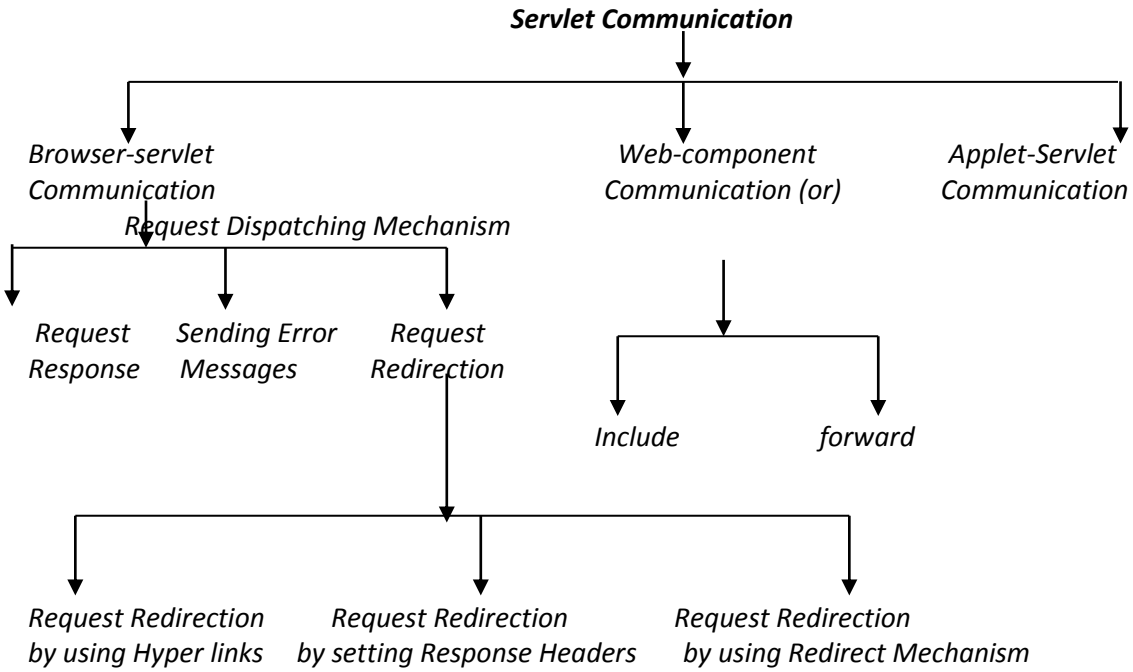
Note: in web application place the all jar files in lib folder instead of build path.

CustomerServlet.java:-

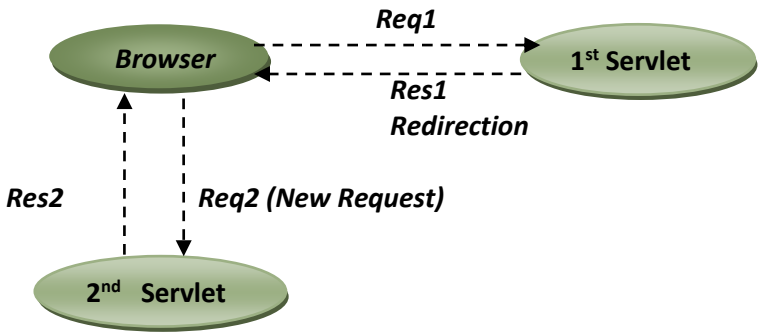
```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CustomerServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;


        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                doPost(request, response);
        }


protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer = response.getWriter();

                String firstname = request.getParameter("firstName");
                String lastname = request.getParameter("lastName");
                String email = request.getParameter("email");
                String mobile = request.getParameter("mobile");
                String[] s = request.getParameterValues("t1");

                StringBuffer sb = new StringBuffer();
```

```
		for(String ss : s)
		{	sb.append(ss+",");
		}
		String addr = new String(sb);

		//printing our data
		writer.println("first name="+firstname+"<br>");
		writer.println("last name="+lastname+"<br>");
		writer.println("email="+email+"<br>");
		writer.println("mobile="+mobile+"<br>");
		writer.println("Address="+addr+"<br>");

		try {
			Connection connection = TestDb2.dataSource.getConnection();
			PreparedStatement preparedStatement =
				connection.prepareStatement("insert into customer values(?,?,?,?,?)");
			preparedStatement.setString(1, firstname);
			preparedStatement.setString(2, lastname);
			preparedStatement.setString(3, email);
			preparedStatement.setString(4, mobile);
			preparedStatement.setString(5, addr);

			int a = preparedStatement.executeUpdate();

			 //processing  the data
			if(a==1)
			{	writer.println("record inserted successfully....");
			}
		} catch (SQLException e) {	e.printStackTrace();	}
	}
}
```

Servlet Communications:-



request redirection by using Hyper-link mechanism:-



welcome.html-

```
<HTML>
<HEAD>
Request Redirection
</HEAD>
<BODY>
<form method="post" action="hutch">
to send the request:-<input type="submit" value="click me">
</BODY>
</HTML>
```

Hutch.java:-

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Hutch extends HttpServlet
{	public void doPost(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
	{	res.setContentType("text/html");
		PrintWriter out=res.getWriter();
		out.println("To Get Services From Hutch Click");
		out.println("<a
	href='http://localhost:9999/vodaphoneapp/welcome.html'>CustomerCare@www.vodaphone.co
	m</a>");
		out.println("</font></b></center></body></html>");
	}
}
```

Web.xml:-

```
<web-app>
	<welcome-file-list>
		<welcome-file>welcome.html</welcome-file>
	</welcome-file-list>
<servlet>
	<servlet-name>h</servlet-name>
	<servlet-class>Hutch</servlet-class>
</servlet>
<servlet-mapping>
	<servlet-name>h</servlet-name>
	<url-pattern>/hutch</url-pattern>
</servlet-mapping>
</web-app>
```

Send redirect by using response-header mechanism:-

```
	res.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
	res.setHeader("location","http://www.facebook.com");
```

Send redirect by using send-redirect mechanism:-

By using send-redirect we are able to redirect the request to within the server and outside of the server.

```
res.sendRedirect("http://www.google.com");
```

LoginServlet.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
```

```
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

                response.setContentType("text/html");
                PrintWriter writer = response.getWriter();
                String uname = request.getParameter("uname");
                String upwd = request.getParameter("upwd");

                if(uname.equals("sravya")&& upwd.equals("infotech"))
                {               /*writer.println("login success");
                writer.println("<a href='http://facebook.com'>click the link to redirect req to fb</a>");
                                */
                        /*response.setStatus(HttpServletResponse.SC_MOVED_PERMANENTLY);
                        response.setHeader("location","https://www.google.co.in");*/

                        response.sendRedirect("http://facebook.com");
                }
                else
                {       response.sendError(808,"login fail try once again");
                }
        }
}
```

Login.html:-

```
<html>
<body>
<form action="./LoginServlet">
user name: <input type="text" name="uname"/><br/>
password : <input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<welcome-file-list>
<welcome-file>login.html</welcome-file>
</welcome-file-list>
<servlet>
<display-name>LoginServlet</display-name>
<servlet-name>LoginServlet</servlet-name>
<servlet-class>com.sravya.LoginServlet</servlet-class>
```

```
</servlet>
<servlet-mapping>
<servlet-name>LoginServlet</servlet-name>
<url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
</web-app>
```

RequestDispatcher:-

The requestDispatcher is used to dispatch the request to another resources(html,servlets,jsp) and it is include the content of another resource.

There are two methods are available in the RequestDispatcher Object

1. *Public void Include(ServletRequest request,ServletResponse response)throws ServeltException,IOException*

   By using above method we are able to transfer the request from one resource to another resources like html,servlets,jsp
2. *Public void Forward (ServletRequest request,ServletResponse response)throws ServletException,IOException*

   By using above method we are able to include the target resource response.

Step 1: get RequestDispatcher object

RequestDispatcher is an object, it will provide very good environment either to include the target resource response into the present resource response or to forward request from present resource to the target resource.

To get RequestDispatcher object we will use the following 2 ways.

1. ServletContext

1. getRequestDispatcher(_) method

2. gteNamedDispatcher(_) method

2. ServletRequest

1. getRequestDispatcher(_) method

getRequestDispatcher(_) vs getNamedDispatcher(_):-

Both the methods are used to get the RequestDispatcher object.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method then we should pass the locator of target resource as parameter.

Note : In case of the servlet, url pattern is treated as locator.

public RequestDispatcher getRequestDispatcher(String path)

To get RequestDispatcher object, if we use getNamedDispatcher(_) method then we have to pass logical name of target resource as parameter.

Note : In case of the servlet, logical name is a name specified along with <servlet-name> tag in web.xml file.

public RequestDispatcher getNamedDispatcher(String path)

What is the difference between getRequestDispatcher(_) method ServletContext and ServletRequest?

Both the methods can be used to get the RequestDispatcher object.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method from ServletContext then we must pass the relative path of target resource.

To get RequestDispatcher object, if we use getRequestDispatcher(_) method from ServletRequest then we have to pass either relative path or absolute path of target resource.

Note: *In general, relative path should prefix with forward slash("/") and absolute path should not prefix with forward slash("/").*

Step 2: Apply either Include mechanism or Forward mechanism to achieve Web-Component Communication:

To represent Include and Forward mechanisms RequestDispatcher has provided the following methods.

public void include(ServletRequest req, ServletResponse res)throws SE, IOE
public void forward(ServletRequest req, ServletResponse res)throws SE, IOE

Login.html:-

```
<html>
<body bgcolor='red'>
<form action="./login" method="post">
Name:<input type="text" name="uname"/><br/>
Password:<input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<servlet>
<servlet-name>sss</servlet-name>
<servlet-class>Servlet1</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>sss</servlet-name>
<url-pattern>/login</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>Welcome</servlet-name>
<servlet-class>Servlet2</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Welcome</servlet-name>
<url-pattern>/success</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>login.html</welcome-file>
</welcome-file-list>
</web-app>
```

Servlet1.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.*;
public class Servlet1 extends HttpServlet {
        public void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
```

```
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

		String upwd=request.getParameter("upwd");
		if(upwd.equals("ratan")){
			RequestDispatcher rd=request.getRequestDispatcher("success");
			rd.forward(request, response);
		}
		else{
			writer.print("oops ! Sorry username or password error! try once again !");
			RequestDispatcher rd=request.getRequestDispatcher("login.html");
			rd.include(request, response);
		}
	}
}
```

Servlet2.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class Servlet2 extends HttpServlet {
	public void doPost(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();

		String n=request.getParameter("uname");
		out.println("Welcome to ratansoft world : "+n);
		out.println("ratan sir welcomes u : "+n);
	}
}
```

<u>ServletContext vs ServletConfig :-</u>

Q: What are the differences between ServletConfig and ServletContext?

1. ServletConfig is an object, it will manage all the configuration details of a particular servlet, where the configuration details include logical name of the servlet, initialization parameters and so on. ServletContext is an object, it will manage all the context details of a particular web application, where the context details include logical name of web application and context parameters and so on.

2. ServletConfig is an object, it will provide the complete view of a particular servlet.

ServletContext is an object, it will provide the complete view of particular web application.

3. ServletConfig object will be prepared by the container immediately after servlet instantiation and just before calling init(_) method in servlet initialization.
ServletContext object will be prepared by the container the moment when we start the server i.e. the time when we deploy the web application.

4. ServletConfig object will be destroyed by the container just before servlet deinstaniation.
ServletContext object will be destroyed by the container when we shutdown the server i.e. the time when we undeploy the web application.

5. Due to the above reasons, the life of ServletConfig object is almost all the life of the respective servlet object.
The life of ServletContext object is almost all the life of the respective web application.

6. If we declare any data in ServletConfig object then that data will be shared upto respective servlet.
If we declare any data in ServletContext object then that data will be shared to all the no. of resources which are available in the present web application.

7. Due to the above reason, ServletConfig object will provide less sharability where as ServletContext object will provide more sharability.

8. In web applications, container will prepare ServletConfig object when it receives the request from client only except in load-on-startup case.
In web applications, container will prepare ServletContext object irrespective of getting request from client.

9. In web applications, ServletConfig object will allow only parameters data but ServletContext object will allow both parameters and attributes data.

10. Due to the above reason, ServletConfig object will allow only Static Inclusion of data where as ServletContext object will allow both Static Inclusion and Dynamic Inclusion of data.
To get the ServletContext object we have to use the following method from ServletConfig.

```
public ServletContext getServletContext();
```

Ex: ServletContext context=config.getServletContext();
Note: In servlet3.0 version, it is possible to get ServletContext object even from ServletRequest.

Ex: ServletContext context=req.getServletContext();
If we want to get the logical name of the web application from ServletContext object first of all we have to declare it in web.xml file.
To declare a logical name in web.xml file we have to use the following xml tag.

```
<web-app>
        ----------
<display-name>logical_name</display-name>
```

</web-app>

To get the logical name of web application from ServletContext object we will use the following method.

Public String getServletContextName()

Ex: String lname=context.getServletContextName();

If we want to provide context parameters on ServletContext object first we have to declare them in web.xml file.

Form.html :

```
<html>
<body>
        <a href="/FirstServlet">click here to get FirstServlet data</a>
        <a href="/secondServlet">click here to get SecondServlet data</a>
</body>
</html>
```

Web.xml:-

```
<web-app>
<display-name>App1</display-name>
        <context-param>
                <param-name>username</param-name>
                <param-value>system</param-value>
        </context-param>

        <context-param>
                <param-name>password</param-name>
                <param-value>manager</param-value>
        </context-param>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>com.sravya.FirstServlet</servlet-class>
        <init-param>
                <param-name>aaa</param-name>
                <param-value>apple</param-value>
        </init-param>
        <init-param>
                <param-name>bbb</param-name>
                <param-value>ratan</param-value>
        </init-param>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>com.sravya.SecondServlet</servlet-class>
```

```
        <init-param>
                <param-name>ccc</param-name>
                <param-value>orange</param-value>
        </init-param>


        <init-param>
                <param-name>ddd</param-name>
                <param-value>anu</param-value>
        </init-param>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer = response.getWriter();
                ServletConfig config =   this.getServletConfig();
                writer.println(config.getInitParameter("aaa"));
                writer.println(config.getInitParameter("bbb"));

                ServletContext context =  config.getServletContext();
                writer.println(context.getInitParameter("username"));
                writer.println(context.getInitParameter("password"));
        }
}
```

SecondServlet.java

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SecondServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer = response.getWriter();

                ServletConfig config =   this.getServletConfig();

                writer.println(config.getInitParameter("ccc"));
                writer.println(config.getInitParameter("ddd"));

                ServletContext context =  config.getServletContext();
                writer.println(context.getInitParameter("username"));
                writer.println(context.getInitParameter("password"));
        }
}
```

Login Application with include & forward:-

Login.html:-

```
<html>
<head>
<body>
<form action="./LoginServlet">
user name: <input type="text" name="uname"/><br/>
password : <input type="password" name="upwd"/><br/>
<input type="submit" value="login"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<display-name>App2</display-name>
<welcome-file-list>
<welcome-file>login.html</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>LoginServlet</servlet-name>
<servlet-class>com.sravya.LoginServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>LoginServlet</servlet-name>
<url-pattern>/LoginServlet</url-pattern>
</servlet-mapping>
```

```
<servlet>
<servlet-name>SuccessServlet</servlet-name>
<servlet-class>com.sravya.SuccessServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SuccessServlet</servlet-name>
<url-pattern>/SuccessServlet</url-pattern>
</servlet-mapping>
</web-app>
```

LoginServlet.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class LoginServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();
		String uname = request.getParameter("uname");
		String upwd = request.getParameter("upwd");
		if(uname.equals("sravya")&&upwd.equals("infotech"))
		{
		//RequestDispatcher dispatcher = request.getRequestDispatcher("SuccessServlet");
			//dispatcher.forward(request, response);
	request.getRequestDispatcher("SuccessServlet").forward(request, response);//project level code
		}
		else
		{	writer.println("!OOPS  login failed can u please try again");
			//RequestDispatcher dispatcher = request.getRequestDispatcher("login.html");
			//dispatcher.include(request, response);
	request.getRequestDispatcher("login.html").incude(request, response);//project level code
		}
	}
}
```

SuccessServlet.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
public class SuccessServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer = response.getWriter();
                writer.println("login success this is success servlet");
                writer.println("<br>");
                writer.println("we are using forward mach");
        }
}
```

3. Applet-Servlet Communication:

In general in web application, we will use a browser as client, we will send request from client browser to a servlet available at server , by executing the respective servlet some response will be send back to the client browser.

Similarly in case of Applet-Servlet Communication, we will use applet as client, from the applet only we will send request to the respective servlet available at server machine, where by executing the respective servlet the required response will be generated and send back to the applet.

In above situation, the communication which we provided between applet and servlet is called as Applet-Servlet Communication.

If we want to achieve Applet-Servlet Communication in web applications we have to use the following steps.

Step 1: Prepare URL object with the respective url.

```
URL u=new
        URL("http://localhost:8080/loginapp/login?uname=abc&upwd=abc123");
```

Step 2: Establish connection between applet and server by using URLConnection object.

```
URLConnection uc=u.openConnection();
```

Step 3: Send request from applet to servlet.

```
uc.setDoInput(true);
```

Note: If we do the above step a request will be send to servlet from applet where container will execute the respective servlet, generate the response and send that response to applet client. But, the response is available on URLConnection object.

Step 4: Get InputStream from URLConnection.

```
InputSream is=uc.getInputStream();
```

Step 5: Read the response from InputStream.

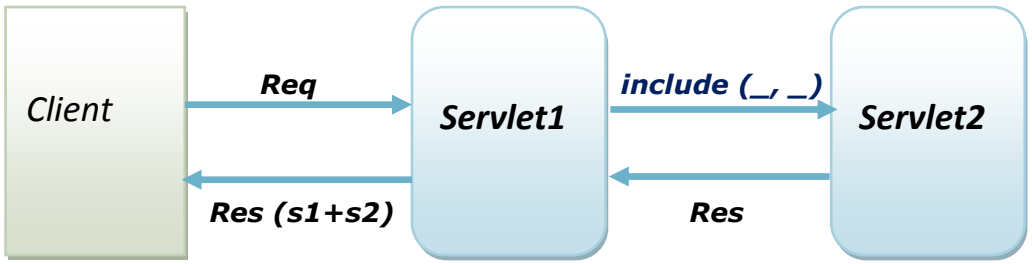
```
BufferedReader br=new BufferedReader(new InputStreamReader(is));
String res=br.readLine();
```

LoginApplet.java:-

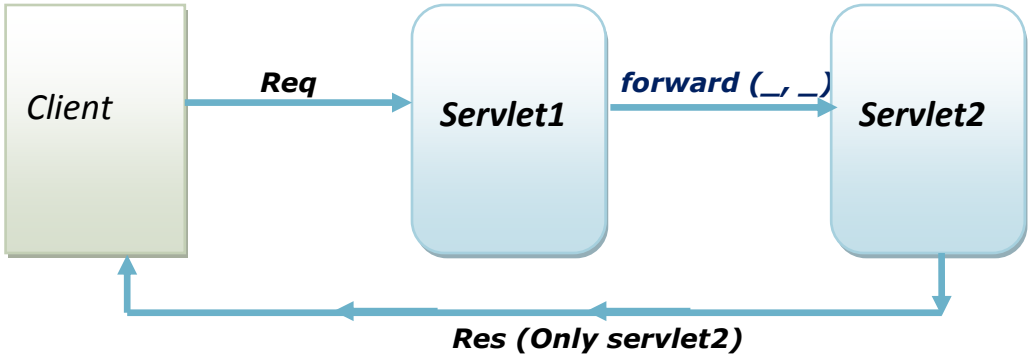
```
import java.applet.Applet;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Label;
```

```
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
public class LoginApplet extends Applet implements ActionListener {
	Label l1,l2;
	TextField tf1,tf2;
	Button b;
	String res="";
	public void init(){
		this.setBackground(Color.pink);
		this.setLayout(new FlowLayout());
		l1=new Label("User Name");
		l2=new Label("Password");
		tf1=new TextField(20);
		tf2=new TextField(20);
		tf2.setEchoChar('*');
		b=new Button("Login");
		b.addActionListener(this);
		this.add(l1);	this.add(tf1);		this.add(l2);	this.add(tf2);	this.add(b);
	}
	public void actionPerformed(ActionEvent ae) {
		try{
URL u=new
URL("http://localhost:2011/loginapp1/login?uname="+tf1.getText()+"&upwd="+tf2.getText());
			URLConnection uc=u.openConnection();
			uc.setDoInput(true);
			InputStream is=uc.getInputStream();
			BufferedReader br=new BufferedReader(new InputStreamReader(is));
			res=br.readLine();
			repaint();
		}catch (Exception e) {
			e.printStackTrace();
		}
	}
	public void paint(Graphics g){
		Font f=new Font("arial", Font.BOLD, 30);
		g.setFont(f);
		g.drawString("Status :"+res, 50, 250);
	}
}
oginApplet.html:-
<applet code="LoginApplet" width="500" height="500"></applet>
```

Include:



Forward:



Filters:-

- ✓ *To write the pre-processing logics & post processing logics we are using filters.*
- ✓ *The main objective of the filter is to provide request preprocessing & response post processing.*
- ✓ *Filters are mainly used to perform filtering tasks,*
  - *Conversion*
  - *Encryption, decription*
  - *Security*
  - *Input validations*
  - *Data compression …etc*

Filter Api :-

To develop the filter example we have to use below interfaces and these interfaces are present in javax.servlet package

a. *Filter <interface>*
b. *FilterConfig <interface>*
c. *FilterChain <interface>*

Filter creation :-

Our normal java class will become Filter class when we implements Filter interface.The Filter contains 3-methods hence in implementation class must override three methods.

Filter implementation:-

```
public interface Filter
{   public abstract void init(FilterConfig filterconfig)   throws ServletException;
        public abstract void doFilter(ServletRequest servletrequest, ServletResponse
servletresponse, FilterChain filterchain)     throws IOException, ServletException;
   public abstract void destroy();
}
```

Example :

```
class MyFilter implements Filter
{       //logics here   [must override three methods]
}
```

Init() *:- this method invoked only once it is used to initialize the Filter*

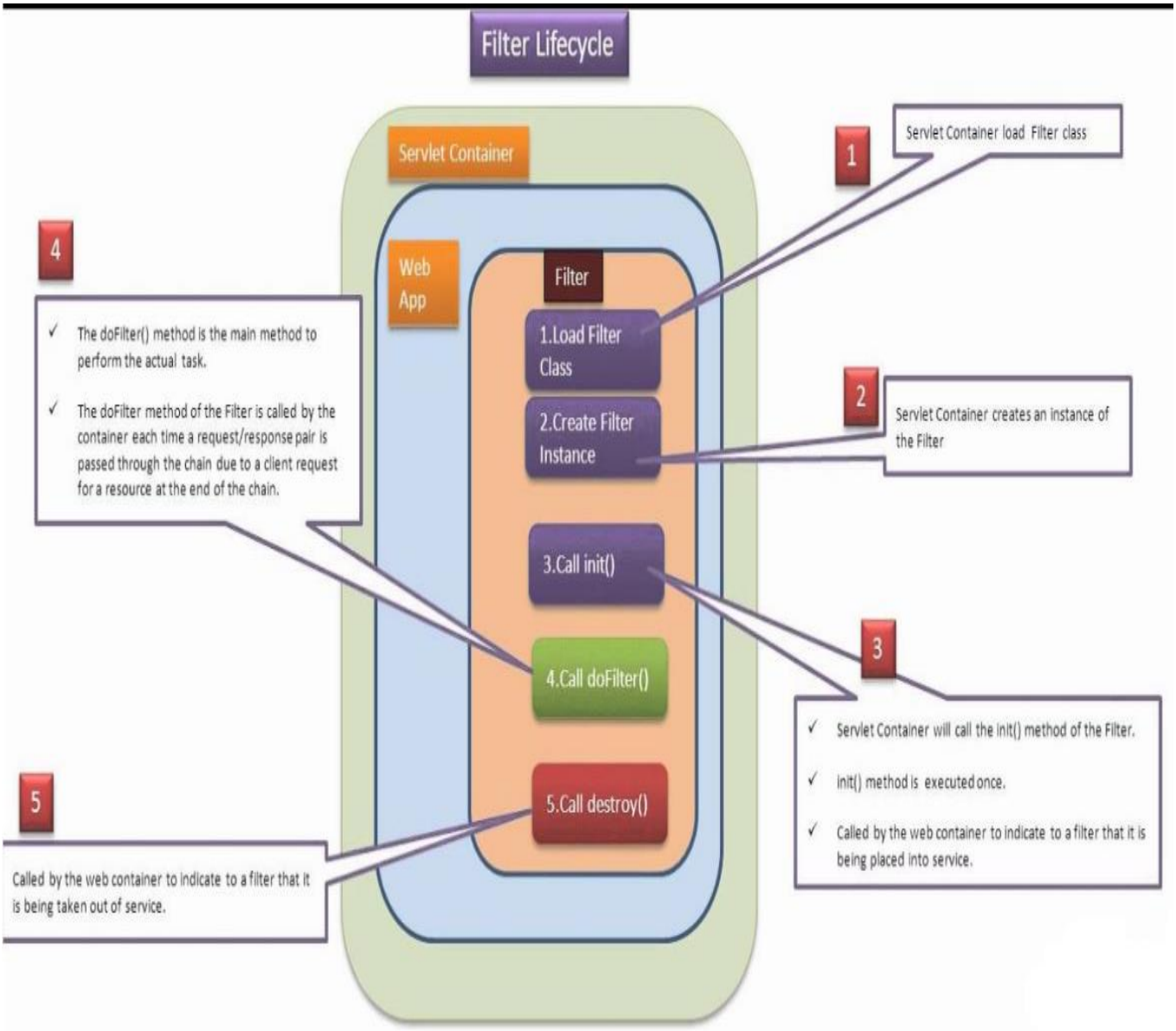
doFilter() :-

it is invoked every time when user send the request to resource. And it is used to write the logics of the Filter by using doFilter() method.

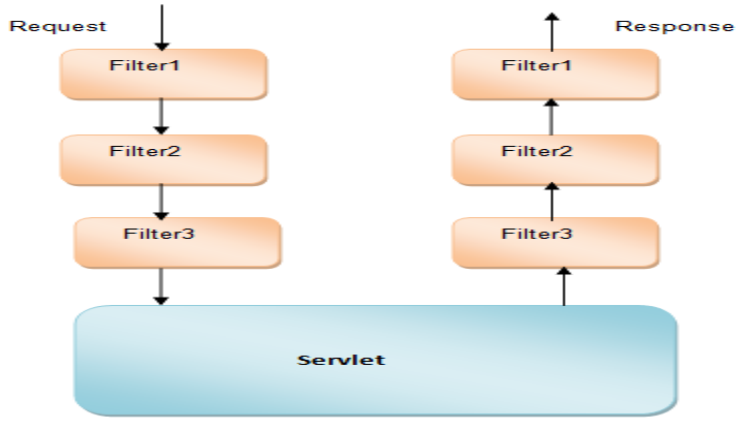
destroy() :- *called when the filter is has been taken out of the service.*

FilterConfig : it contains configuration details of particular Filter
ServletRequest : contains requested details.
ServletResponse : actual response to disply to client browser.
FilterChain : used to forward the request to next resource.

Filter life cycle



Request & reponce Flow:-



Where init(_) method can be used to perform Filter Initialization.

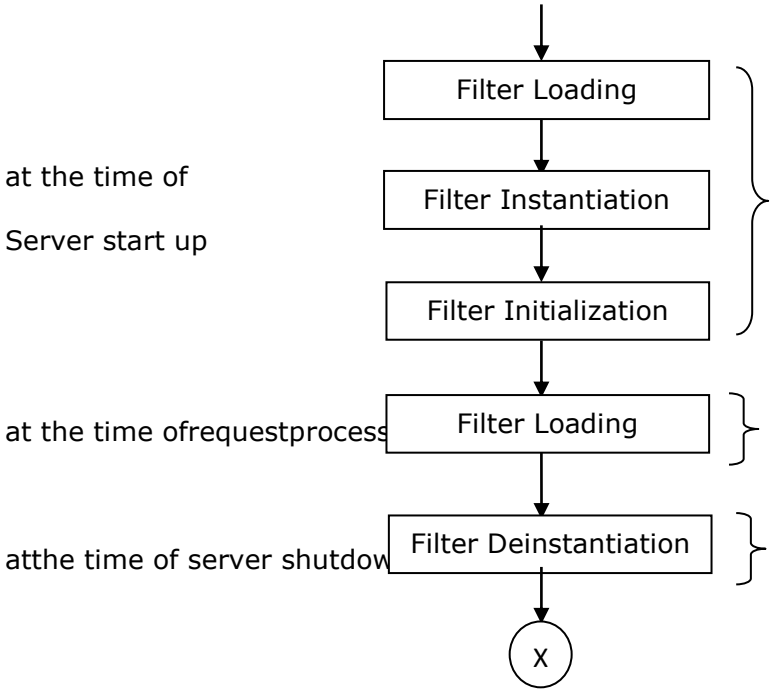
Where doFilter(_,_,_) method is same as service(_,_) method in servlets it is able to accommodate actual Filter logic.

Where destroy() method can be used to perform Filter Deinstantiation.

While executing a particular Filter in web applications, if we satisfy all the Filter constraints then we need to bypass the request from present Filter to the next Filter web resource, for this we need to use the following method from FilterChain.

public void doFilter(ServletRequest req, ServletResponse res)throws SE, IOE

While executing a particular web application, when container identify a particular Filter to execute then container will execute that Filter by following the following Filter life cycle.



In web applications, by default all the Filters are auto-loaded, auto-instantiated, auto-initialized at the time of server start up. So that Filters should not require load-on-startup configuration in web.xml file.

Step 2: Filter class Configuration:

To configure a Filter in web.xml file we have to use the following xml tags.

```
<web-app>
<filter>
<filter-name>logical_name</filter-name>
<filter-class>Fully Qualified name of Filter</filter-class>
</filter>
<filter-mapping>
<filter-name>logical_name</filter-name>
<url-pattern>pattern_name</url-pattern>
                    or
<servlet-name>logical name of servlet</servlet-name>
</filter-mapping>
        --------
</web-app>
```

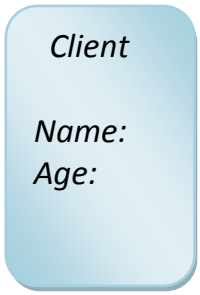
If we want to provide mapping between a Filter and Servlet then we have to provide the same url-pattern for both Filter and Servlet or provide the respective servlet logical name along with <servlet-name> tag in place of <url-pattern> tag under <filter-mapping>.

In web applications, it is possible to use a single Filter for multiple number of web resources.

To achieve this we have to use " **/*** " (Directory match) as url-pattern to the respective Filter.

In web applications, it is possible to provide multiple number of Filters for a single web resource, in this case container will execute all the Filters as per the order in which we provided <filter-mapping> tags in web.xml file.

1. *Get the requested data*
2. *Create the session object*
3. *place the data in session obj*
4. *Forward the req to ext form*



Form.html:-

```
<html>
<body>
<form action="./SuccessServlet">
User name : <input type="text" name="uname"/><br>
user age   : <input type="text" name="uage"/><br>
user address : <input type="text" name="uaddr"/><br>
<input type="submit" value="registration"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<display-name>App3</display-name>
<welcome-file-list>
<welcome-file>form.html</welcome-file>
</welcome-file-list>

<filter>
<filter-name>Filter1</filter-name>
<filter-class>com.sravya.Filter1</filter-class>
</filter>
<filter-mapping>
<filter-name>Filter1</filter-name>
<url-pattern>/SuccessServlet</url-pattern>
</filter-mapping>

<filter>
<filter-name>Filter2</filter-name>
<filter-class>com.sravya.Filter2</filter-class>
</filter>
<filter-mapping>
<filter-name>Filter2</filter-name>
<url-pattern>/SuccessServlet</url-pattern>
</filter-mapping>

<servlet>
        <servlet-name>SuccessServlet</servlet-name>
<servlet-class>com.sravya.SuccessServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SuccessServlet</servlet-name>
<url-pattern>/SuccessServlet</url-pattern>
</servlet-mapping>
</web-app>
```

Filter1.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class Filter1 implements Filter {
	public void destroy() {
		// TODO Auto-generated method stub
	}
	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();
		int age = Integer.parseInt(request.getParameter("uage"));

		if(age>20)
		{
		// pass the request along the filter chain
		chain.doFilter(request, response);
		}
		else
		{	writer.println("u r not eligible for mrg u age is below 20 years");
			request.getRequestDispatcher("form.html").include(request, response);
		}
	}
	public void init(FilterConfig fConfig) throws ServletException {
	}

}
```

Filter2.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
public class Filter2 implements Filter {
	public void destroy() {
		// TODO Auto-generated method stub
```

```
	}
	public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();
		String uaddr = request.getParameter("uaddr");
		if(uaddr.equals("hyderabad"))
		{	chain.doFilter(request, response);
		}
		else
		{	writer.println("this application only for hyd person");
			request.getRequestDispatcher("form.html").include(request, response);
		}
	}
	public void init(FilterConfig fConfig) throws ServletException {
		// TODO Auto-generated method stub
	}
}
```

SuccessServlet.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SuccessServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();
		String uname = request.getParameter("uname");
		String uage = request.getParameter("uage");
		String uaddr = request.getParameter("uaddr");

		writer.println("************ U r registration success************");
		writer.println("<br>");
		writer.println("user name="+uname);
		writer.println("<br>");
		writer.println("<br>");
		writer.println("user age="+uage);
		writer.println("<br>");
		writer.println("user naddress="+uaddr);
		writer.println("<br>");
		writer.println("user registration id=13456");
		writer.println("<br>");
		writer.println("we will find one girl for u soon.....keep smiling");
	}
```

}

Session Tracking Mechanisms:

As part of the web application development it is essential to manage the clients previous request data at the time of processing later request.

To achieve the above requirement if we use request object then container will create request object when it receives request from client and container will destroy request object when it dispatch response to client.

Due to this reason request object is not sufficient to manage clients previous request data at the time of processing later request.

To achieve the above requirement we able to use ServletContext object, but ServletContext object will share its data to all the components which are used in the present applications and to all the users of the present web application.

Due to this reason ServletContext object is unable to provide clear cut separation between multiple users.

In web applications, to manage clients previous request data at the time of processing later request and to provide separation between multiple users we need a set of mechanisms explicitly at server side called as **Session Tracking Mechanisms.**

Session:

Session is a time duration, it will start from the starting point of client conversation with server and will terminate at the ending point of client conversation with the server.

The data which we transferred from client to server through multiple number of requests during a particular session then that data is called **State of the Session.**

In general in web applications, container will prepare a request object similarly to represent a particular user we have to prepare a separate session.

If we allow multiple number of users on a single web application then we have to prepare multiple number of session objects.

In this context, to keep track of all the session objects at server machine we need a set of explicit mechanisms called as **Session Tracking Mechanisms.**

In web applications, there are 4 types of Session Tracking Mechanisms.

1) ***HttpSession Session Tracking Mechanism***
2) ***Coockies Session Tracking Mechanism***
3) ***URL-Rewriting Session Tracking Mechanism***
4) ***Hidden Form Fields Session Tracking Mechanism***

From the above Session Tracking Mechanisms Servlet API has provided the first 3 Session Tracking Mechanisms as official mechanisms, Hidden Form Fields Session Tracking Mechanism is purely developers creation.

1. HttpSession Session Tracking Mechanism:

In **HttpSession Session Tracking Mechanism**, we will create a separate HttpSession object for each and every user, at each and every request we will pick up the request parameters from request object and we will store them in the respective HttpSession object for the sake of future reusability.

After keeping the request parameters data in HttpSession object we have to generate the next form at client browser by forwarding the request to particular static page or by generating dynamic form.

In HttpSession Session Tracking Mechanism, to create HttpSession object we will use either of the following methods.
req.getSession();
req.getSession(false);

Q: What is the difference between getSession() method and getSession(false) method?
Ans: Both the methods can be used to return HttpSession object.

To get HttpSession object if we getSession() method then container will check whether any HttpSession object existed for the respective user or not, if any HttpSession object is existed then container will return the existed HttpSession object reference.

If no HttpSession object is existed for the respective user then container will create a new HttpSession object and return its reference.

public HttpSession getSession()

Ex: HttpSession hs=req.getSession();

To get HttpSession object if we getSession(false) method then container will check whether any HttpSession object existed w.r.t. user or not, if any HttpSession object is existed then container will return that HttpSession object reference.

If no HttpSession object is existed then container will return null value without creating new HttpSession object.

public HttpSession getSession(boolean b)

Ex: HttpSession hs=req.getSession(false);

Note: getSession(true) method functionality is almost all same as getSession() method.

Q: If we allow multiple number of users to access present web application then automatically container will create multiple number of HttpSession objects. In this case how container will identify user specific HttpSession object in order to put user specific attributes and to get attributes?
Ans: In HttpSession Session Tracking Mechanism, when we create HttpSession object automatically container will create an unique identification number in the form of hexadecimal number called as **Session Id.**Container will prepare session id in the form of Cookie with the name **JSESSIONID.**

In general the basic nature of Cookie is to transfer from server to client automatically along with response and it will be transferred from client to server automatically along with request.

Due to the above nature of Cookies session id Cookie will be transferred from server to client and from client to server automatically.

In the above context, if we use getSession() method or getSession(false) method first container will pick up session id value from request and it will identify the user specific HttpSession object on the basis of session id value.

To destroy HttpSession object explicitly we will use the following method from HttpSession.
public void invalidate()

If we want to destroy HttpSession object after a particular ideal time duration then we have to use the following method.

public void setMaxInactiveInterval(int time)

In web applications, HttpSession object will allow only attributes data, it will not allow parameters data.

To set an attribute on to the HttpSession object we have to use the following method.

public void setAttribute(String name, Object value)

To get a particular attribute value from HttpSession object we have to use the following method.

public Object getAttribute(String name)

To get all attribute names from HttpSession object we have to use the following method.
public Enumeration getAttributeNames()
To remove an attribute from HttpSession object we have to use the following method.
public void removeAttribute(String name)

Drawbacks:

1. In HttpSession Session Tracking Mechanism, we will create a separate HttpSession object for each and every user, where if we increase number of users then automatically number of HttpSession object will be created at server machine, it will reduce the overall performance of the web application.
2. In case of HttpSession Session Tracking Mechanism, we are able to identify user specific HttpSession object among multiple number of HttpSession objects by carrying Session Id value from client to server and from server to client.

In the above context, if the client browser disable Cookies then HttpSession Session Tracking Mechanism will not execute its functionality.

Form1.html:

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="./FirstServlet">
<pre>
                    Name        <input type="text" name="uname"/>
                    Age                 <input type="text" name="uage"/>
                    <input type="submit" value="Next"/>
</pre>
</form></font></b></body></html>
```

Form2.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br><hr>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="./SecondServlet">
<pre>               Qualification       <input type="text" name="uqual"/>
                    Designation                 <input type="text" name="udes"/>
                    <input type="submit" value="Next"/>
</pre>
</form></font></b></body></html>
```

Form3.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="./DisplayServlet">
<pre>                    Email          <input type="text" name="email"/>
                        Mobile                  <input type="text" name="mobile"/>
                        <input type="submit" value="display"/>
</pre>
</form></font></b></body></html>
```

Web.xml:-

```
<web-app >
<display-name>HttpSession</display-name>
<welcome-file-list>
<welcome-file>form1.html</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>com.sravya.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>com.sravya.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>DisplayServlet</servlet-name>
<servlet-class>com.sravya.DisplayServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DisplayServlet</servlet-name>
<url-pattern>/DisplayServlet</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
package com.sravya;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class FirstServlet extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		String uname = request.getParameter("uname");
		String uage = request.getParameter("uage");

		HttpSession session = request.getSession();//sid-cookie

		session.setAttribute("uname",uname);
		session.setAttribute("uage",uage);

		RequestDispatcher dispatcher = request.getRequestDispatcher("form2.html");
		dispatcher.forward(request, response);
	}
}
```

SecondServlet.java:-

```
package com.sravya;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class SecondServlet extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		String uqual = request.getParameter("uqual");
		String udes =  request.getParameter("udes");
		HttpSession session = request.getSession();//sid already avaialble

		session.setAttribute("uqual", uqual);
		session.setAttribute("udes", udes);

		RequestDispatcher dispatcher = request.getRequestDispatcher("form3.html");
		dispatcher.forward(request, response);
	}
}
```

DisplayServlet.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
public class DisplayServlet extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

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

		HttpSession session = request.getSession();//located

		writer.println("*******Complete details*******");
		writer.println("<br>");
		writer.println("user name: "+session.getAttribute("uname"));
		writer.println("<br>");
		writer.println("user age: "+session.getAttribute("uage"));
		writer.println("<br>");
		writer.println("user qualification: "+session.getAttribute("uqual"));
		writer.println("<br>");
		writer.println("user designation: "+session.getAttribute("udes"));
		writer.println("<br>");
		writer.println("user mobile: "+mobile);
		writer.println("<br>");
		writer.println("user email: "+email);
		writer.println("<br>");
	}
}
```

URL-Rewriting Session Tracking Mechanism:

In case of HttpSession Session Tracking Mechanism, when we create HttpSession object automatically Session Id will be created in the form of the Cookie, where Session Id Cookie will be transferred from server to client and from client to server along with response and request automatically.

In HttpSession Session Tracking Mechanism, we are able to identify user specific HttpSession object on the basis of Session Id only.

In this context, if we disable Cookies at client browser then HttpSession Session Tracking Mechanism will not execute its functionality.

In case of Cookies Session Tracking Mechanism, the complete client conversation will be stored at the respective client machine only in the form of Cookies, here the Cookies data will be opened to every user of that machine. So that Cookies Session Tracking Mechanism will not provide security for the application data.

To overcome the above problem, we have to use URL-Rewriting Session Tracking Mechanism.

In case of URL-Rewriting Session Tracking Mechanism, we will not maintain the clients conversation at the respective client machine, we will maintain the clients conversationin the form of HttpSession object at server machine. So that URL-Rewriting Session Tracking Mechanism is able to provide very good security for the application data.

URL-Rewriting Session Tracking Mechanism is almost all same as HttpSession Session Tracking Mechanism, in URL-Rewriting Session Tracking Mechanism we will not depending on a Cookie to maintain Session Id value, we will manage Session Id value as an appender to URL in the next generated form.

In this context, if we send a request from the next generated form automatically the appended Session Id value will be transferred to server along with the request.
In this case, eventhough if we disable Cookies at client browser, but still we are able to get Session Id value at server machine and we are able to manage clients previous request data at the time of processing the later request.

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session Id value in the next generated form. So that this mechanism is called as **URL-Rewriting Session Tracking Mechanism.**

In URL-Rewriting Session Tracking Mechanism, it is mandatory to append Session Id value to the URL by getting Session Id value explicitly.

To perform this work HttpServletResponse has provided a separate method like,

public String encodeURL(String url)

Ex: out.println("<form method='get'
action='"+res.encodeURL("./second")+"'>");

Drawback:

In URL-Rewriting Session Tracking Mechanism, every time we need to rewrite the URL with Session Id value in the generated form, for this we must execute encodeURL() method. So that URL-Rewriting Session Tracking Mechanism should require dynamically generated forms, it will not execute its functionality with static forms.

Form.html:-

```
<html>
<body>
<form action="./FirstServlet">
User name : <input type="text" name="uname"/><br>
user age   : <input type="text" name="uage"/><br>
<input type="submit" value="next"/>
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
<display-name>url</display-name>
<welcome-file-list>
<welcome-file>form.html</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>com.sravya.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>com.sravya.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>DisplayServlet</servlet-name>
<servlet-class>com.sravya.DisplayServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DisplayServlet</servlet-name>
<url-pattern>/DisplayServlet</url-pattern>
</servlet-mapping>

</web-app>
```

FirstServlet.java:-

```
package com.sravya;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class FirstServlet extends HttpServlet {
	@SuppressWarnings("deprecation")
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		String uname = request.getParameter("uname");
		String uage = request.getParameter("uage");

		HttpSession session = request.getSession();

		session.setAttribute("uname", uname);
		session.setAttribute("uage", uage);

		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

		writer.println("<html>");
		writer.println("<body>");
	writer.println("<form method='get' action="+response.encodeUrl("./SecondServlet")+">");
		writer.println("<br>");
		writer.println("user qualificatins :<input type='text' name='uqual'/>");
		writer.println("<br>");
		writer.println("user designation : <input type='text' name='udes'/>");
		writer.println("<br>");
		writer.println("<input type='submit' value='next'/>");
		writer.println("</form>");
		writer.println("</body>");
		writer.println("</html>");
	}
}
```

SecondServlet.java:-

```
package com.sravya;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class SecondServlet extends HttpServlet {

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

		String uqual = request.getParameter("uqual");
		String udes = request.getParameter("udes");

		HttpSession session = request.getSession();

		session.setAttribute("uqual", uqual);
		session.setAttribute("udes", udes);

		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

		writer.println("<html>");
		writer.println("<body>");
		writer.println("<form method='get'
action="+response.encodeUrl("./DisplayServlet")+">");
		writer.println("<br>");
		writer.println("user email :<input type='text' name='email'/>");
		writer.println("<br>");
		writer.println("user mobile : <input type='text' name='mobile'/>");
		writer.println("<br>");
		writer.println("<input type='submit' value='display'/>");
		writer.println("</form>");
		writer.println("</body>");
		writer.println("</html>");
	}
}
```

DisplayServlet.java:-

```
package com.sravya;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class DisplayServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

		HttpSession session = request.getSession();//locate

		writer.println("user name ="+session.getAttribute("uname"));
		writer.println("<br>");
		writer.println("user age ="+session.getAttribute("uage"));
		writer.println("<br>");
		writer.println("user qualifications ="+session.getAttribute("uqual"));
		writer.println("<br>");
		writer.println("user designations ="+session.getAttribute("udes"));
		writer.println("<br>");
		writer.println("user email id ="+request.getParameter("email"));
		writer.println("<br>");
		writer.println("user mobile ="+request.getParameter("mobile"));
	}
}
```

Cookies Session tracking machanisum:-

- ✓ *Cookie is a small object, it can be used to represent a single name value pair and which will be maintained permanently at client machine.*
- ✓ *In Cookies Session Tracking Mechanism, at each and every client we will pick up all the request parameters, prepare a separate Cookie for each and every request parameter, add all the Cookies to response object.*

In the above context, when container dispatch response to client automatically all the added Cookies will be transferred to client and maintain at client machine permanently.

In the above context, when we send further request from the same client automatically all the Cookies will be transferred to server along with request.

By repeating the above process at each and every request we are able to manage client's previous data at the time of processing later request.

Limitaitons:-

- ✓ *If we disable the Cookies at client browser then Cookies Session Tracking Mechanism will not execute its functionality.*
- ✓ *In case of Cookies Session Tracking Mechanism, all the clients data will be maintain at the respective client machine only, which is open to every user of that machine. So that Cookies Session Tracking Mechanism will not provide security for the application data.*

To create Cookie object with a particular name-value pair we have to use the following Cookie class constructor.

```
public Cookie Cookie(String name, String value)
```

To add a Cookie to the response object we have to use the following method from HttpServletResponse.

```
public void addCookie(Cookie c)
```

To get all the Cookies from response object we need to use the following method.

```
public Cookie[] getCookies()
```

To get name and value of a Cookie we have to use the following methods,

```
public String getName()
public String getValue()
```

In web applications, it is possible to provide comments to the Cookies. So that to set the comment to Cookie and get the comment from Cookie we need to use the following methods.

```
public void setComment(String comment)
public String getComment()
```

In web applications, it is possible to provide version numbers to the Cookies. So that to set a version number to Cookie and get a version number from Cookie we need to use the following methods.

```
public void setVersion(int version_no)
public int getVersion()
```

In web applications, it is possible to specify life time to the Cookies. So that to set max age to Cookie and get max age from Cookie we need to use the following methods.

```
public void setMaxAge(int age)
public int getMaxAge()
```

In web applications, it is possible to provide domain names to the Cookies. So that to set domain name to Cookie and get domain name from Cookie we need to use the following methods.

public void setDomain(String domain)
public String getDomain()

In web applications, it is possible to provide a particular path to the Cookies to store. So that to set a path to Cookie and get a path from Cookie we need to use the following methods.

public void setPath(String path)
public String getPath()

Form1.html:

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="./FirstServlet">
<pre>                    Name          <input type="text" name="uname"/>
                    Age                    <input type="text" name="uage"/>
                    <input type="submit" value="Next"/>
</pre>
</form></font></b></body></html>
```

Form2.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br><hr>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="./SecondServlet">
<pre>                    Qualification          <input type="text" name="uqual"/>
                    Designation                    <input type="text" name="udes"/>
                    <input type="submit" value="Next"/>
</pre>
</form></font></b></body></html>
```

Form3.html

```
<html>
<head><center><b><font color="red" size="7">
Registration Form
</font></b></center></head>
<br><br>
<body bgcolor="lightblue"><b>
<font size="7">
<form method="post" action="./DisplayServlet">
```

```
<pre>                    Email           <input type="text" name="email"/>
                        Mobile              <input type="text" name="mobile"/>
                        <input type="submit" value="display"/>
</pre>
</form></font></b></body></html>
```

Web.xml:-

```
<web-app >
<display-name>Cookies</display-name>
<welcome-file-list>
<welcome-file>form1.html</welcome-file>
</welcome-file-list>

<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>com.sravya.FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/FirstServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>com.sravya.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/SecondServlet</url-pattern>
</servlet-mapping>

<servlet>
<servlet-name>DisplayServlet</servlet-name>
<servlet-class>com.sravya.DisplayServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DisplayServlet</servlet-name>
<url-pattern>/DisplayServlet</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
package com.sravya;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class FirstServlet extends HttpServlet {

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

		String uname = request.getParameter("uname");
		String uage = request.getParameter("uage");

		Cookie cookie1 = new Cookie("uname", uname);
		Cookie cookie2 = new Cookie("uage", uage);

		response.addCookie(cookie1);
		response.addCookie(cookie2);

		request.getRequestDispatcher("form2.html").forward(request, response);
	}
}
```

SecondServlet:-

```
package com.sravya;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SecondServlet extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		String uqual = request.getParameter("uqual");
		String udes = request.getParameter("udes");

		Cookie cookie3 = new Cookie("uqual",uqual);
		Cookie  cookie4 = new Cookie("udes",udes);

		response.addCookie(cookie3);
		response.addCookie(cookie4);
		request.getRequestDispatcher("form3.html").forward(request, response);
	}
}
```

DisplayServlet.java:-

```
package com.sravya;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class DisplayServlet extends HttpServlet {
	protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();

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

		Cookie[] c = request.getCookies();
		writer.println("user name="+ c[0].getValue());
		writer.println("<br>");
		writer.println("user name="+ c[1].getValue());
		writer.println("<br>");
		writer.println("user name="+ c[2].getValue());
		writer.println("<br>");
		writer.println("user name="+ c[3].getValue());
		writer.println("<br>");
		writer.println("user name="+ mobile);
		writer.println("<br>");
		writer.println("user name="+ email);
		writer.println("<br>");
	}
}
```

Hidden Form Field Session Tracking Mechanism:-

- ✓ *Hidden Form Field Session Tracking Mechanism is not official Session Tracking Mechanism from Servlet API, it was purely developers creation.*
- ✓ *In Hidden Form Field Session Tracking Mechanism, at each and every request we will pick up all the request parameters, generate dynamic form, in dynamic form generation we have to maintain the present request parameters data in the form of hidden fields.*

In the above context, if we dispatch the response to client then we are able to get a dynamic form with visible fields and with invisible fields.

If we send a request from dynamic form then automatically all the visible fields data and invisible fields data will be send to server as request parameters.

By repeating above process at each and every request we are able to manage the clients previous request data at the time of processing the later request.

studentform.html:-

```
<html>
<head>
<body bgcolor="lightgreen">
        <form method="get" action="./first">
        <center><b><br><br>
        Student Name:<input type="text" name="sname"/><br><br>
        Student Id:<input type="text" name="sid"/><br><br>
        Student Address:<input type="text" name="saddr"/><br><br>
        <input type="submit" value="Submit">
        </b></center>
        </form>
</body>
</html>
```

web.xml:-

```
<web-app>
<display-name>hiddenformfieldsapp</display-name>
<welcome-file-list>
<welcome-file>studentform.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>FirstServlet</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>FirstServlet</servlet-name>
<url-pattern>/first</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>SecondServlet</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
```

```
<servlet-name>SecondServlet</servlet-name>
<url-pattern>/second</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>ThirdServlet</servlet-name>
<servlet-class>ThirdServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>ThirdServlet</servlet-name>
<url-pattern>/third</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class FirstServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                PrintWriter out=response.getWriter();

                String sname=request.getParameter("sname");
                String sid=request.getParameter("sid");
                String saddr=request.getParameter("saddr");

                out.println("<html><body bgcolor='lightyellow'>");
                out.println("<center><b><br><br>");
                out.println("Welcome to Student Application");
                out.println("<br><br>");
                out.println("<form method='get' action='/hiddenformfieldsapp/second'>");
                out.println("<input type='hidden' name=sname value='"+sname+"'>");
                out.println("<input type='hidden' name=sid value='"+sid+"'>");
                out.println("<input type='hidden' name=saddr value='"+saddr+"'>");
                out.println("<br><br>");
                out.println("Student Age:");
                out.println("<input type='text' name='sage'>");
                out.println("<br><br>");
                out.println("<input type='submit' value='Submit'>");
                out.println("</form></b></center></body></html>");
        }
}
```

SecondServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SecondServlet extends HttpServlet {
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                PrintWriter out=response.getWriter();
                String sname=request.getParameter("sname");
                String sid=request.getParameter("sid");
                String saddr=request.getParameter("saddr");
                String sage=request.getParameter("sage");
                out.println("<html><body bgcolor='lightyellow'>");
                out.println("<center><b><br><br>");
                out.println("Student Details Are...");
                out.println("<br><br>");
                out.println("Student Name....."+sname+"<br><br>");
                out.println("Student Id....."+sid+"<br><br>");
                out.println("Student Address....."+saddr+"<br><br>");
                out.println("<a href='/hiddenformfieldsapp/third?sage="+sage+"'>
                                        SHOW STUDENT AGE</a>");
        }
}
```

ThirdServlet.java:-

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class ThirdServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                PrintWriter out=response.getWriter();
                String sage=request.getParameter("sage");
                out.println("<html><body bgcolor='lightpink'>");
                out.println("<center><b><br><br>");
                out.println("Student Age is....."+sage);
                out.println("</b></center></body></html>");
        }
}
```

Servlet Listeners & Events :-

ServletContextListener & ServletContextEvent :-

- ✓ *ServletContext event is notified when the web application deployed into the server.*
- ✓ *If you want to perform any operations at the time of deployment of the web application such as creating database connection & table creation …etc*
- ✓ *We have to implement ServletContextListener interface we have to provide the implementation of fallowing methods.*

The ServletContextInterface contains fallowing methods,

public void contextInitialized(ServletContextEvent e)

itis invoked when application is deployed on the server.

public void contextDestroyed(ServletContextEvent e):

it is invoked when application is undeployed from the server.

Form.html:-

```
<html>
<body>
        <form method="get" action="./MyServlet">
        enter table name to Get Data : <input type="text" name="tname"/>
        <input type="submit" value="submit">
        </form>
</body>
</html>
```

Web.xml:

```
<web-app>

<welcome-file-list>
<welcome-file>form.html</welcome-file>
</welcome-file-list>

<listener>
<listener-class>com.dss.MyListener</listener-class>
</listener>

<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>com.dss.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/MyServlet</url-pattern>
</servlet-mapping>

</web-app>
```

MyListener.java:

```
package com.dss;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

public class MyListener implements ServletContextListener {
    public void contextDestroyed(ServletContextEvent event)  {
        System.out.println("contextDestroyed method Connection closed");
        ServletContext context =  event.getServletContext();
        Connection connection =  (Connection) context.getAttribute("conn");
        try {      connection.close();
        }
        catch (SQLException e) {
        e.printStackTrace();  }
     }

   public void contextInitialized(ServletContextEvent event)  {
        System.out.println("contextInitialized method Connection creation");
      try {
                  Class.forName("oracle.jdbc.driver.OracleDriver");
                  Connection connection =
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","ratan");
                  ServletContext  context =  event.getServletContext();
                  context.setAttribute("conn", connection);
        }
         catch (ClassNotFoundException | SQLException e)
        {         // TODO Auto-generated catch block
                  e.printStackTrace();
        }
   }
}
```

MyServlet.java:-
package com.dss;

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyServlet extends HttpServlet {
	protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter writer = response.getWriter();
		String tname  = request.getParameter("tname");

		ServletContext context =  getServletContext();
		Connection connection =  (Connection) context.getAttribute("conn");
		try {
	PreparedStatement preparedStatement = connection.prepareStatement("select * from
"+tname,ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

			ResultSet set =  preparedStatement.executeQuery();
			while(set.next())
			{writer.println(set.getInt(1)+"---"+set.getString(2)+"---"+set.getDouble(3));
			}
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
}
```

HttpSessionEvent vs HttpSessionListener:-

The httpSessionEvent is notified when the session object is changed. The corresponding listener is HttpSessionListener.

We can perform some operatrions like counting number of login logged in users & maintains the login details.

There are two methods declared in the HttpSessionListener interface which must be implemented by the servlet programmer to perform some action.

public void sessionCreated(HttpSessionEvent e)
is invoked when session object is created.
public void sessionDestroyed(ServletContextEvent e)
is invoked when session is invalidated.

Form.html:-

```
<html>
<body>
<form method='get' action="./First">
Name:<input type="text" name="username"><br>
Password:<input type="password" name="userpass"><br>
<input type="submit" value="login"/>
</form>
</html>
</body>
```

Web.xml:-

```
<web-app>
<welcome-file-list>
<welcome-file>form.html</welcome-file>
</welcome-file-list>
<listener>
<listener-class>com.dss.CountUserListener</listener-class>
</listener>
<servlet>
    <servlet-name>First</servlet-name>
    <servlet-class>com.dss.First</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>First</servlet-name>
    <url-pattern>/First</url-pattern>
</servlet-mapping>
<servlet>
    <servlet-name>Logout</servlet-name>
    <servlet-class>com.dss.Logout</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>Logout</servlet-name>
    <url-pattern>/Logout</url-pattern>
</servlet-mapping>
</web-app>
```

CountUSerListener:-

```
package com.dss;

import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpSessionEvent;
import javax.servlet.http.HttpSessionListener;

public class CountUserListener implements HttpSessionListener{
   ServletContext ctx=null;
   static int total=0,current=0;

   public void sessionCreated(HttpSessionEvent e) {
          System.out.println("Session object created with id:"+e.getSession().getId());
   total++;
   current++;
     ctx=e.getSession().getServletContext();
   ctx.setAttribute("totalusers", total);
   ctx.setAttribute("currentusers", current);
      }

  public void sessionDestroyed(HttpSessionEvent e) {
          System.out.println("Session object Destroyed with id:"+e.getSession().getId());
          current--;
     ctx.setAttribute("currentusers",current);
   }
}
```

First.java:-

```
package com.dss;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

public class First extends HttpServlet {
public void doGet(HttpServletRequest request,HttpServletResponse response) throws ServletException,
IOException {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      String n=request.getParameter("username");
      out.print("Welcome "+n);

      HttpSession session=request.getSession();

//retrieving data from ServletContext object
      ServletContext ctx=getServletContext();
      int t=(Integer)ctx.getAttribute("totalusers");
      int c=(Integer)ctx.getAttribute("currentusers");
      out.print("<br>total users= "+t);
      out.print("<br>current users= "+c);

      out.print("<br><a href='Logout'>logout</a>");

      out.close();
   }
 }
```

Logout.java:-

```
package com.dss;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

 public class Logout extends HttpServlet {
public void doGet(HttpServletRequest request,   HttpServletResponse response)
                                                        throws ServletException, IOException {
        response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      HttpSession session=request.getSession(false);
      session.invalidate();//invalidating session

      out.print("You are successfully logged out");
 out.close();
   }
}
```

Annotations in servlets:-

Annotaitons are used to provide the metadata of the application.
Annotations are introduced in servlet 3.0 version
Annotaitons are replacement of web.xml file but not completely.

General annotations in Servlet

@WebListener	*to configure the listeners*
@WebServlet	*to configure the servlets*
@WebFilter	*to configure the Filters*
@WebInitParam	*to configure the init param data*

With web.xml:-

```
<web-app>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>com.dss.MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/MyServlet</url-pattern>
</servlet-mapping>
</web-app>
```

With annotaitons :-

```
@WebServlet("/MyServlet")
public class MyServlet extends HttpServlet
{       //code here
}
```

Servlet with multiple url patterns:-

```
@WebServlet(urlPatterns = {"/first", "/second"})
public class UploadServlet extends HttpServlet {
   // implement servlet doPost() and doGet()...
}
```

Servlet with load-on-startup:-

```
@WebServlet(urlPatterns="/initializeResources", loadOnStartup=1)
```

Example :-@WebServlet

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebInitParam;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


@WebServlet(urlPatterns = { "/MyServlet" },
                initParams = { @WebInitParam(name = "user1" , value = "ratan"),
                                @WebInitParam(name = "user2" , value = "anu")
                                })

public class MyServlet extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter writer = response.getWriter();

                ServletConfig config = this.getServletConfig();
                writer.println("init param-1="+config.getInitParameter("user1"));
                writer.println("<br>");
                writer.println("init param-2="+config.getInitParameter("user2"));
        }
}
```

Example :- @WebListener is used to configure the fallowing Listeners.

Context Listener	*(javax.servlet.ServletContextListener)*
Context Attribute Listener	*(javax.servlet.ServletContextAttributeListener)*
Servlet Request Listener	*(javax.servlet.ServletRequestListener)*
Servlet Request Attribute Listener	*(javax.servlet.ServletRequestAttributeListener)*
Http Session Listener	*(javax.servlet.http.HttpSessionListener)*
Http Session Attribute Listener	*(javax.servlet.http.HttpSessionAttributeListener)*

```
@WebListener
public class MyListener implements HttpSessionListener
{       //code here
}
```

Example : @WebFilter

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;
import javax.servlet.annotation.WebInitParam;

@WebFilter(urlPatterns="/Filter1",
                initParams={
                        @WebInitParam(name="user1", value="ratan"),
                        @WebInitParam(name="user2", value="anu")
                                })

public class Filter1 implements Filter {
        FilterConfig config;
        public void init(FilterConfig config) throws ServletException {
                        this.config=config;
        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws
IOException, ServletException {
                PrintWriter writer = response.getWriter();
                writer.println("init param-1="+config.getInitParameter("user1"));
                writer.println("<br>");
                writer.println("init param-2="+config.getInitParameter("user2"));
        }
        public void destroy() {
        }
}
```

Limitations of annotaitons :-

1. *It is not possible to configure welcome file list.*
2. *It is not possible to configure <context-param> data.*

Example:-
Web.xml :-

```
<web-app>
<context-param>
<param-name>email</param-name>
<param-value>ratna5256@gmail.com</param-value>
</context-param>
<context-param>
<param-name>phone</param-name>
<param-value>9000160099</param-value>
</context-param>
</web-app>
```

ExampleServlet.java:-

```
package com.dss;

import javax.servlet.ServletConfig;
import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import java.io.PrintWriter;

public class ExampleServlet extends HttpServlet {
   private String email, phone;

   @Override
   public void init(ServletConfig config) throws ServletException {
      ServletContext ctx = config.getServletContext();
      email = ctx.getInitParameter("email");
      phone = ctx.getInitParameter("phone");
   }

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

           response.setContentType("text/html");
          PrintWriter writer = resp.getWriter();

          writer.println("Servlet Context Parameter example configuration");
          writer.println("E-mail:" + email );
          writer.println("Phone: " + phone );
          }
}
```

Response type :-(MIME type)Multipurpose Internet Mail Extension

The servlet is able to send the response back to client in different format like,

a. *Text/html*
b. *Image/jpg*
c. *Pdf*
d. *Excel ……etc*

Response type image format:-

- ✓ *To add the image resposnce into response object use ServletOutputStream object.*
- ✓ *To read the the image use FileInputStream class.*
- ✓ *To make the performance faster use BufferdinputStream & BufferedOutputStream.*

Index.html

click for photo

```
package com.dss;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.FileInputStream;
import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class MyServlet1  extends HttpServlet {
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                response.setContentType("image/jpg");
           ServletOutputStream out = response.getOutputStream();

          FileInputStream fileInputStream = new FileInputStream("F:\\ratan\\ratan.jpg");
           BufferedInputStream inputStream = new BufferedInputStream(fileInputStream);

           BufferedOutputStream outputStream = new BufferedOutputStream(out);
            int c;
           while((c=inputStream.read())!=-1)
           {       outputStream.write(c);
           }
                out.close();
                fileInputStream.close();
                inputStream.close();
                outputStream.close();
        }
}
```

Deployment process:

Generally there are four ways to deploy the application into server.

1. *Create the war file & deploy the war file (import & export options in IDE).*
2. *Deploy the war file in tomcat webapps folder.*
3. *Deploy the war file directly from server console.*
4. *Copy the tomcat folder structure from IDE, deploy into tomcat web apps folder.*

Create the war file & deploy the war file (import & export options in IDE).

JSP (Java Server Pages)

Introduction
Servlet vs Jsp
JSP Model1 Architecture
JSP Model2 Architecture
Jsp life cycle
Jsp toservlet translation
Jsp tags
Jsp implicit 9-objects
Directives
- *Page*
- *Include*
- *Taglib*

Scripting Elements
- *Scriptlets*
- *Expression*
- *Declarations*
- *Comments*

Jsp sopes
- *Page*
- *Request*
- *Application*
- *Session*

Jsp Actions
- *Standred Actions*
- *Custom actions*

Exception handling in jsp

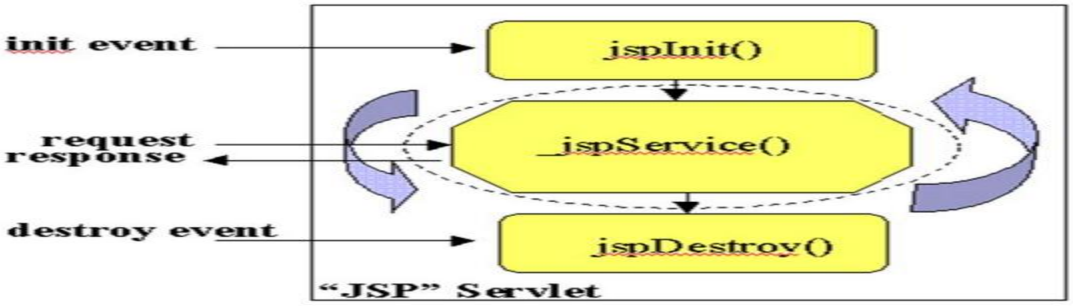
JSTL:-
- *Core tags*
- *SQL tags*
- *Xml tags*
- *Function tags*
- *I18n tags*

Jsp vs Servlets:-

- ✓ *Jsp is extension of servlets so we can use all features of servlets in addition we can use implicit objects, custom tags, predefined tags.*
- ✓ *If we down modifications on servlets for every modification need to recompiled and redeployed but in jsp refresh button is enough to reflect the modifications.*
- ✓ *Servlets runs faster compere to jsp because the jsp are internally converted into servlets that converted servlets are executed then we will get the response.*
- ✓ *To write the servlets we required more java knowledge but to write the jsp code less java knowledge is sufficient.*
- ✓ *In MVC jsp acts as a view part & servlets acts as a controller part.*
- ✓ *Servlets are best for more processing logics but jsp are best to develop dynamic web pages for more presentation logics rather than processing logics.*
- ✓ *The present version of servlets is **3.1**& present version of jsp is **2.2***
- ✓ *In servlets we are mixing both presentation logics and business logics but in jsp we can separate our business logics with presentation logics.*
- ✓ *We must place the servlets in private area of directory structure so to access the private area elements web.xml is mandatory but it is possible to place the jsp pages both public & private area hence to access the jsp **web.xml** is optional.*
- ✓ *The servlets predefined support **servlet-api.jar**& jap predefined support **jsp-api.jar.***
- ✓ *The life cycle methods of servlets*
  - *Inti() Service() Destroy()*
  - 

The life cycle methods of jsp

_jspInt() _jspService() _jspDestroy()



Jsp pages are contains fallowing tags:-

- ✓ *html tags*
- ✓ *jsp tags*
- ✓ *custom tags*
- ✓ *action tags*
- ✓ *jstl tags*
- ✓ *frame work tags structs , jsf, spring*
- ✓ *java code (not recammanded)*

Application using Servlets:-

Login.html:-

```
<html>
<body>
<form action="LoginServlet" method="POST">
User Name: <input type="text" name="username" /><br>
Password: <input type="text" name="password" /><br>
<input type="submit" value="Sign In" />
</form>
</body>
</html>
```

Web.xml:-

```
<web-app>
        <welcome-file-list>
        <welcome-file>login.html</welcome-file>
        </welcome-file-list>
</web-app>
```

In this example we are using Annotations hence the web.xml configurations are minimized.

LoginServlet.java:-

```
package com.dss;

import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
        private static final long serialVersionUID = 1L;
```

```
   public LoginServlet() {  }

        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
                doPost(request, response);
        }

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

                response.setContentType("text/html");
                PrintWriter out = response.getWriter();

                String username = request.getParameter("username");
                String password = request.getParameter("password");
                System.out.println("User Name:"+username);
                System.out.println("Password:"+password);

                        if(username.equals("durga") && password.equals("ratan")){
                        //Rendering success response on browser.
                        out.println("<html><body>");
        out.println("<h1><font color=\"green\"/>Successfully logged in at:"+ new Date()+"</h1>");
                        out.println("</body></html>");
                        }
                        else{
                        //Rendering failure response to browser.
                        out.println("<html><body>");
                        out.println("<h1><font color='red'/>Invalid Username or Password.</h1>");
                        out.println("</body></html>");
                        }
        }
}
```

Conclusion:-

In the above example, Servlet class is used for:

1) Handling User Inputs.
2) Performing (or Invoking) Business Logic.
3) Presentation layer (Rendering response on browser).

***Disadvantages**:-*

1. *Writing html code inside the servlets is not recommended and html code written by page authors who may not know java.*
2. *Inside the servlet class the HTML tags are enclosed with out.Println () method it decsable IDE to give predefined support.*
3. *For every change servlet is recompiled.*
4. *Servlets are directly can't access we have to use web.xml file configuration.*

Developing same application by using JSP:-

It is always to recommended to separate both business logics & presentation logics, hence JSP were introduced.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String username = request.getParameter("username");
        String password = request.getParameter("password");
        System.out.println("User Name:"+username);
        System.out.println("Password:"+password);
        //Perform (or Invoke) Business Logic
        if(username.equals("durga") && password.equals("ratan"))
        {       //Invoke Success response page.
                RequestDispatcher rd = request.getRequestDispatcher("/Success.jsp");
                rd.forward(request, response);
        }
        else
        {       //Invoke Failure response page.
                RequestDispatcher rd = request.getRequestDispatcher("/Failure.jsp");
                rd.forward(request, response);
        }
}
```

Success.jsp:-

```
<%@ page import="java.util.Date" %>
<html>
<body>
<h1><font color="green"/>Successfully logged in at:<%= new Date()%></h1>
</body>
</html>
```

Failure.jsp:-

```
<html>
<body>
<h1><font color="red"/>Invalid Username or Password</h1>
</body>
</html>
```

Conclusion: *Now, Servlet class is used for:*

1. *Handle User Inputs*
2. *Perform (or Invoke) Business logic* } *called as presentation logics*
3. ***Control View navigation***

Advantages with JSPs:

1. *JSPs are tag-based approach, hence, convenient for page authors.*
2. *We can use IDEs such as Dream viewer for designing JSP pages.*
3. *JSPs are automatically compiled by the web container.*
4. *JSPs can be accessed directly as a simple without web.xml.*

Presentation logics vs presentation layer:-

presentation logic= read input data + business logics +view navigation
java code java code java code
Hence presentation logic is java code
Presentation layer is html code.

Web container = servlet container + jsp engine
Servlet container name is ***Catalina***
Jsp engine name ***Jasper***

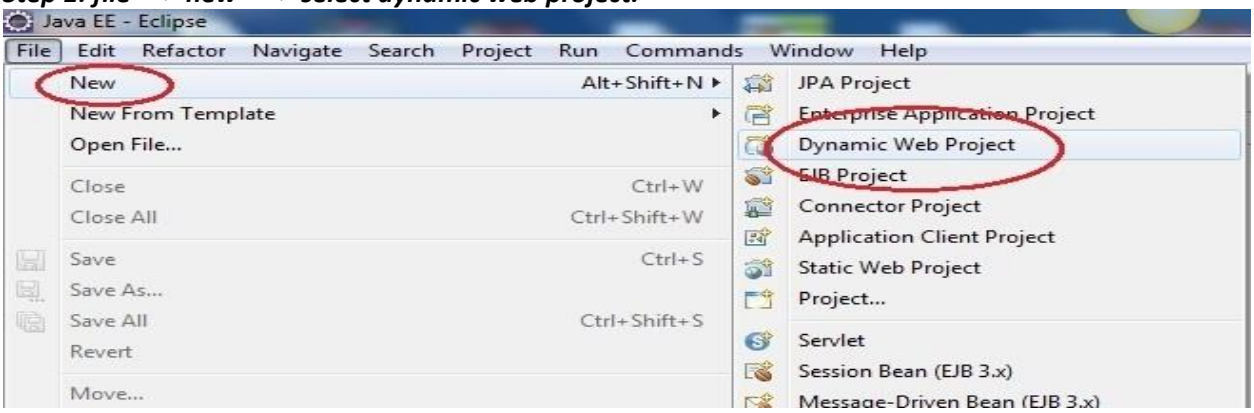
Web servers contains only ***web container***
Application server contains both ***web container & ejb container***

The JSP API contains two packages :-

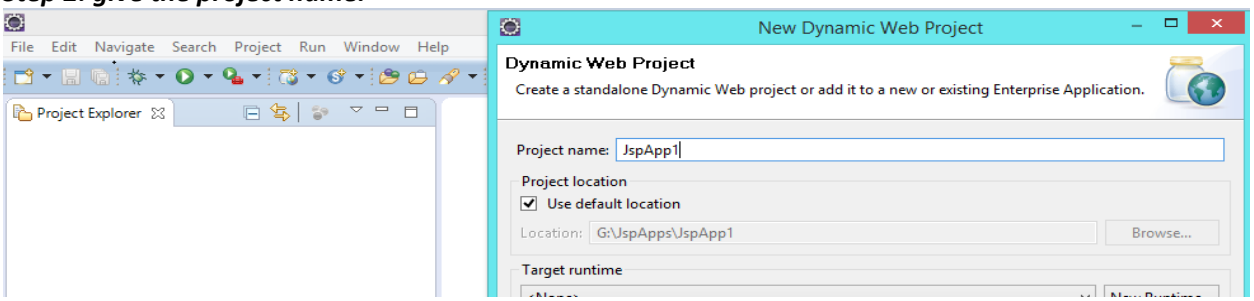
1. *Javax.servlet.jsp*
2. *Javax.servlet.jsp.tagext*

Project creation steps:-

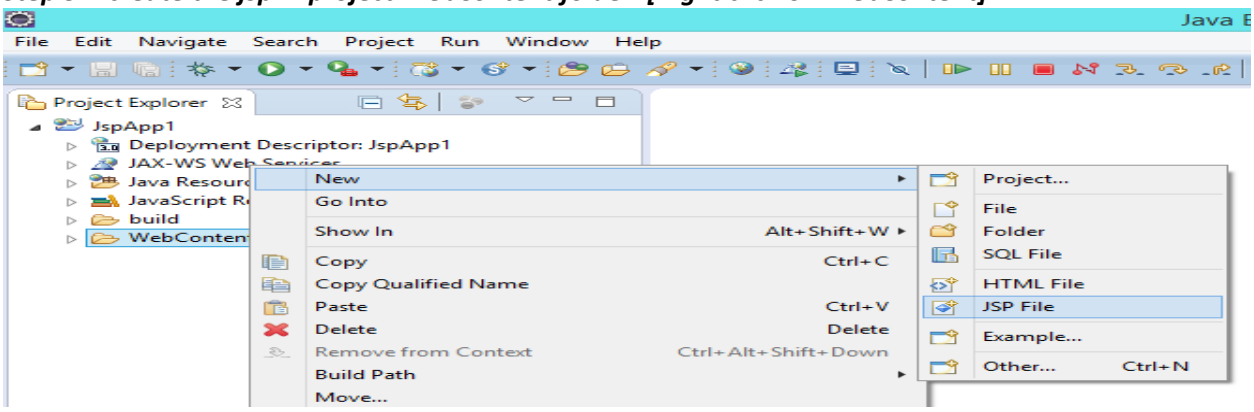
Step 1: file ---> new ----> select dynamic web project.



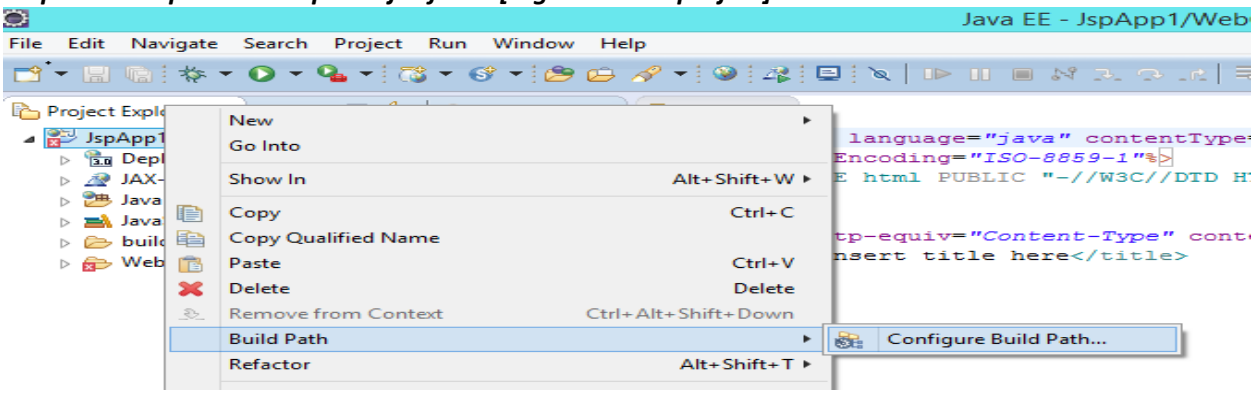
Step 2: give the project name.



Step 3: - create the jsp in project WebContent folder. [Right click on WebContent]



Step 4:- build path the required jar files. [Right click on project]



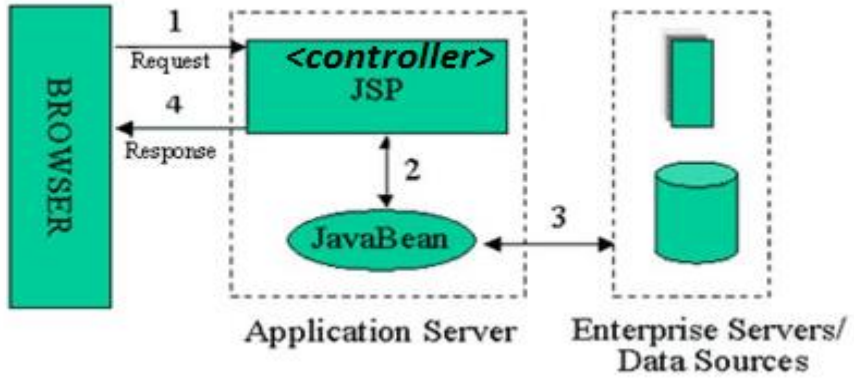
Two models of designing JSPs:-

1. *Jsp Model-1 Architecture*
2. *Jsp Model-2 Architecture*

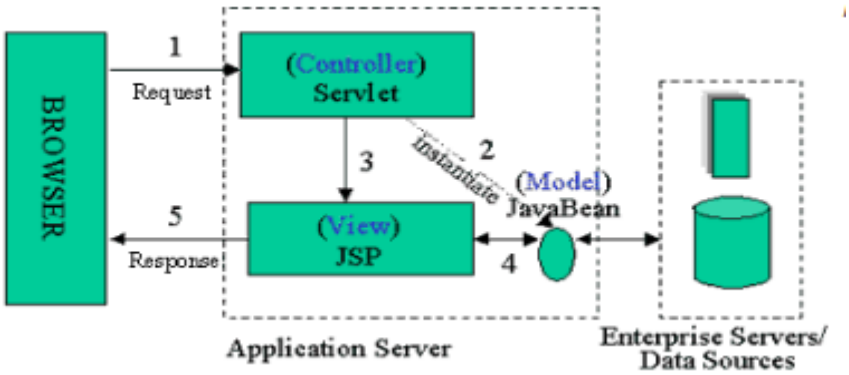
Jsp Model-1 Architecture :-

1. *Presentation layer writing in jsp (no servlets).*
2. *Inputs are handled jsp & business logics are written in jsp*
3. *Presentation logics are writer in jsp.*

In this model jsp alone responsible to write presentation logics & presentation layer & model component.



Jsp Model-2 Architecture :-(MVC model view Controller)



- ✓ *It is also called as model view controller architecture.*

Controller is responsible for presentation logics,

Handling user inputs
Invoke business logics
View navigation

M ---> java bean classes
V ----> JSP pages
C ----> servlets

In model-2 (MVC)architecture the application controller may or may not be centralized since we may have single or more than one controller servlet classes.

in MVC2 the application contains only one controller it is centralized.

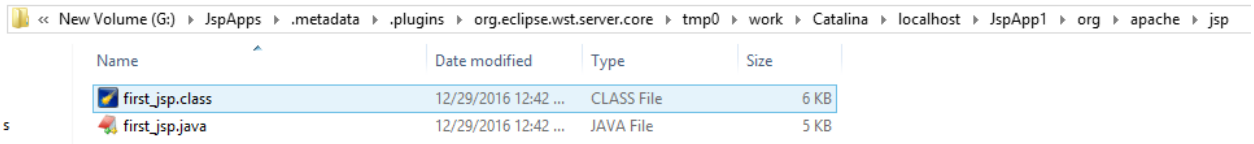
Jsp Application :-

first.jsp:-

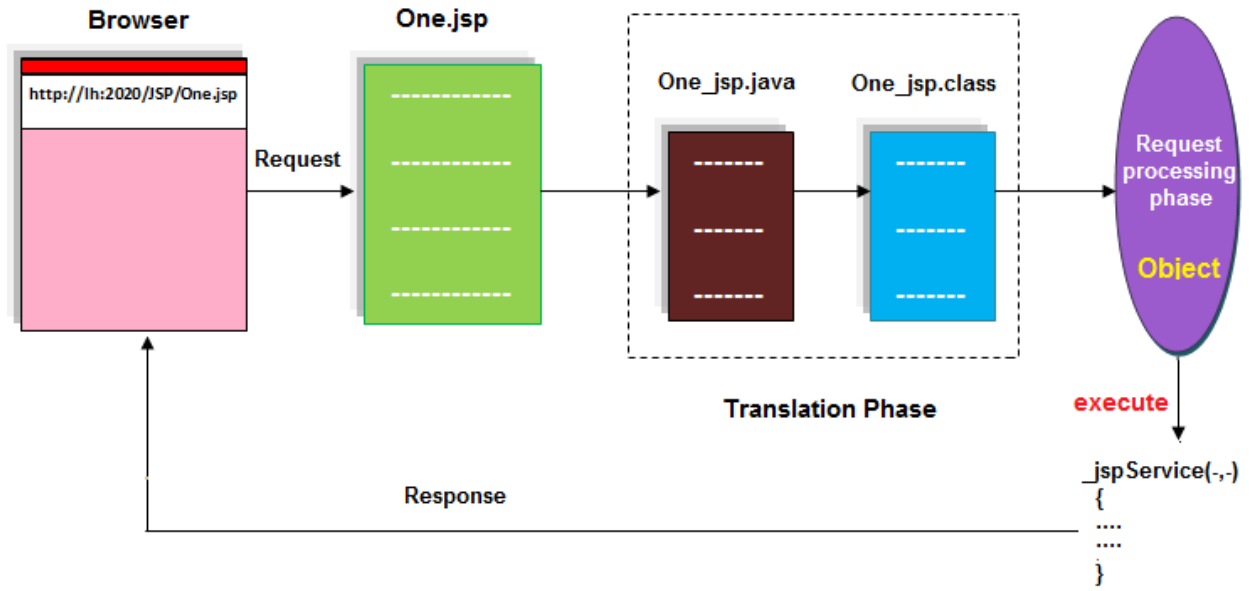
```
<%@ page language="java" contentType="text/html"%>
<html>
<body bgcolor="lightgreen">
<center><b><font size="7" color="red">
   First Jsp Application By Mr. Ratan
</font></b></center>
</body>
</html>
```

When you send the request to first.jsp it is translated to first_jsp.java the translated servlet location is,

G:\JspApps\.metadata\.plugins\org.eclipse.wst.server.core\tmp0\work\Catalina\localhost\JspApp1\org\apache\jsp



For the First Request:-



Mainly the jsp life cycle process :-

1. Translation phase

2. Compilation phase

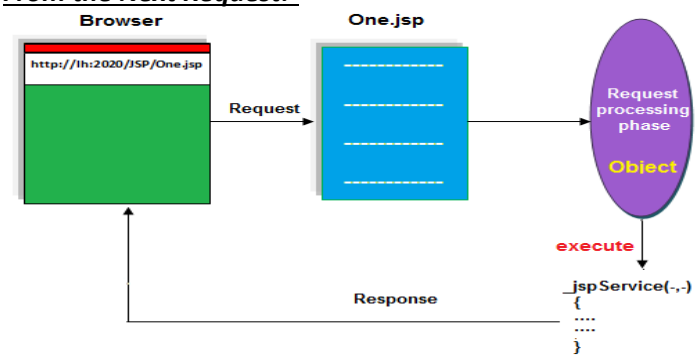
3. Loading & Instantiating

4. _jspInit() method (Initialization)

5. _jspService() method (Request Handling)

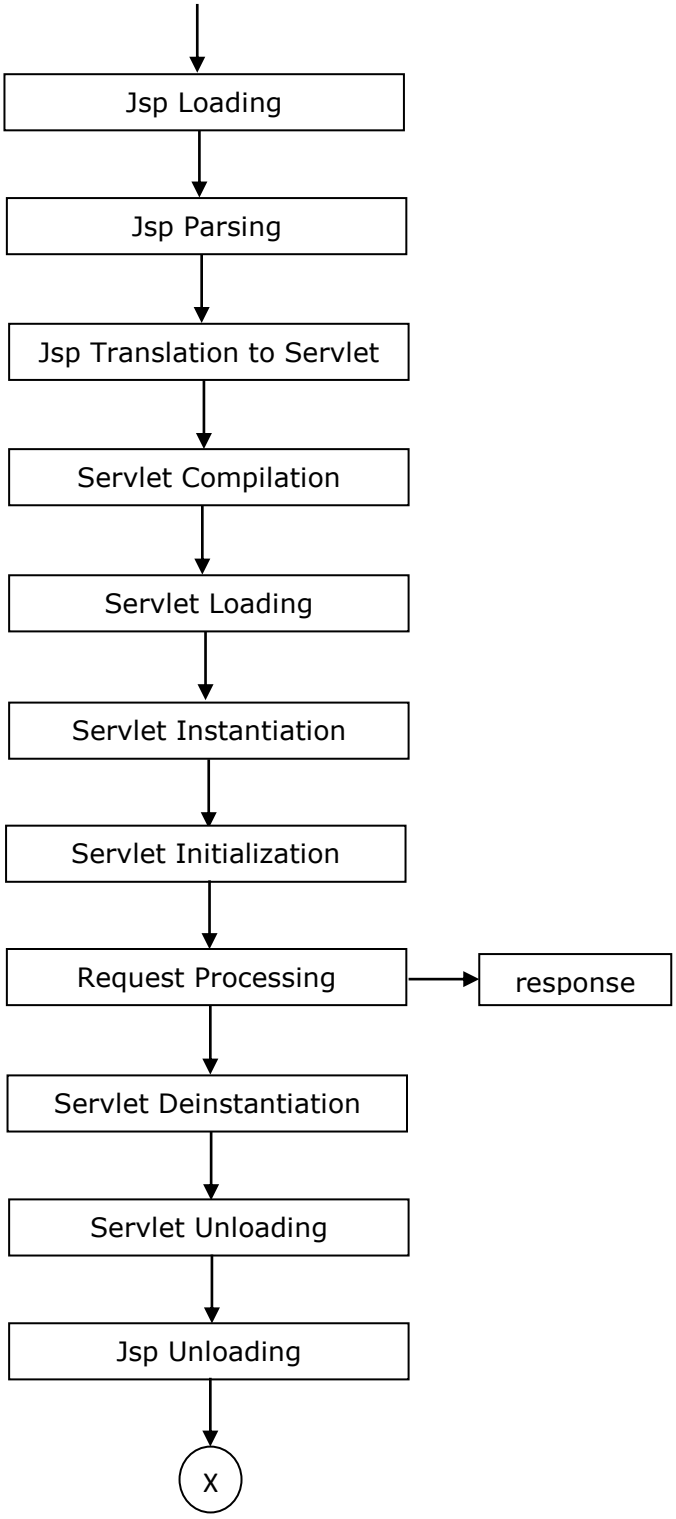
6. _jspDestroy() method(Removing)

From the Next Request:-



jsp Life Cycle:-

request



- ✓ **Jsp Loading:-**
  Here container will load Jsp file to the memory from web application directory structure.

- ✓ **Jsp Parsing:-***Here container will check all the tags in Jsp page are in well-formed format* or not.

- ✓ **Jsp Translation to Servlet:-**
  After the Jsp parsing container will translate the loaded Jsp page into a particular servlet.
  While executing a Jsp page Tomcat container will provide the translated servlet in the following location at Tomcat Server.
  C:\Tomcat7.0\work\catalina\localhost\org\apache\Jsp\first_Jsp.java
  If the Jsp file name is first.jsp then Tomcat Server will provide a servlet with name first_jsp. Java . The default super class for translated servlet is HttpJspBase.

- ✓ **Servlet Compilation:-**
  After getting the translated servlet container will compile servlet java file and generates the respective .class file.

- ✓ **Servlet Loading:-***Here container will load the translated servlet class byte code to the memory.*

- ✓ **Servlet Instantiation:-** *Here container will create object for the loaded servlet.*

- ✓ **Servlet Initialization:-***Here container will access _JspInit() method to initialize the servlet.*

- ✓ **Creating request and response objects:-**
  After the servlet initialization container will create a thread to access _JspService(_,_) method, for this container has to create HttpServletRequest and HttpServletResponse.

- ✓ **Generating Dynamic response:-**
  After getting request and response objects container will access _JspService(_,_) method, by executing its content container will generate some response on response object.

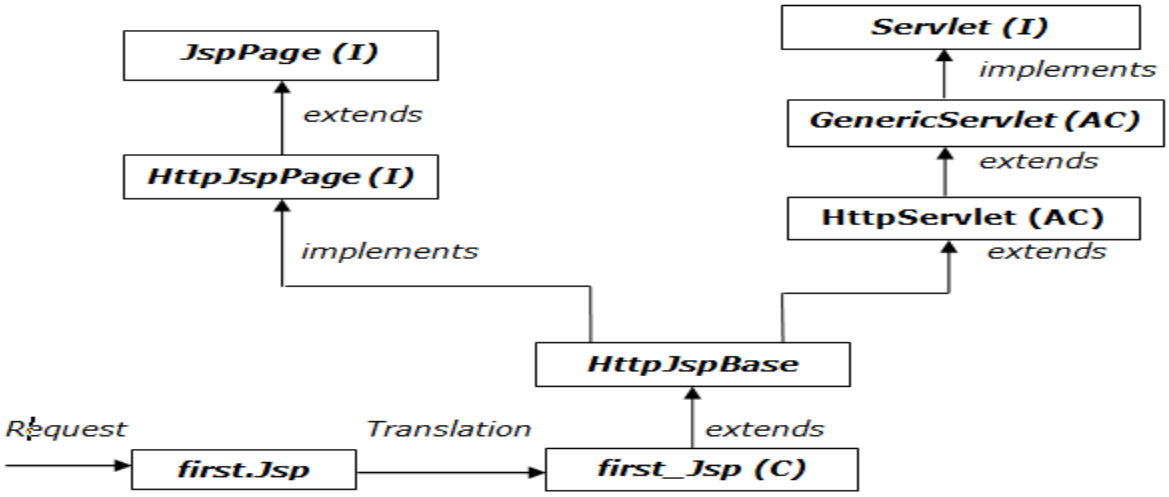
- ✓ **Dispatching Dynamic response to Client:-**
  When container generated thread reached to the ending point of _JspService(_,_) method then that thread will be in Dead state, with this container will dispatch dynamic response to client through the Response Format prepared by the protocol.

- ✓ **Destroying request and response objects:-**
  When the dynamic response reached to client protocol will terminate its virtual socket connection, with this container will destroy request and response objects.

- ✓ **Servlet Deinstantiation:-**
  After destroying request and response objects container will be in waiting state depends on the container, then container identifies no further request for the same resource then container will destroy servlet object, for this container will execute _JspDestroy() method.

- ✓ **Servlet Unloading and Jsp Unloading:-**
  After the servlet deinstantiation container will eliminate the translated servlet byte code and Jsp code from memory.

Internal implementations of all jsp life cycle methods:-

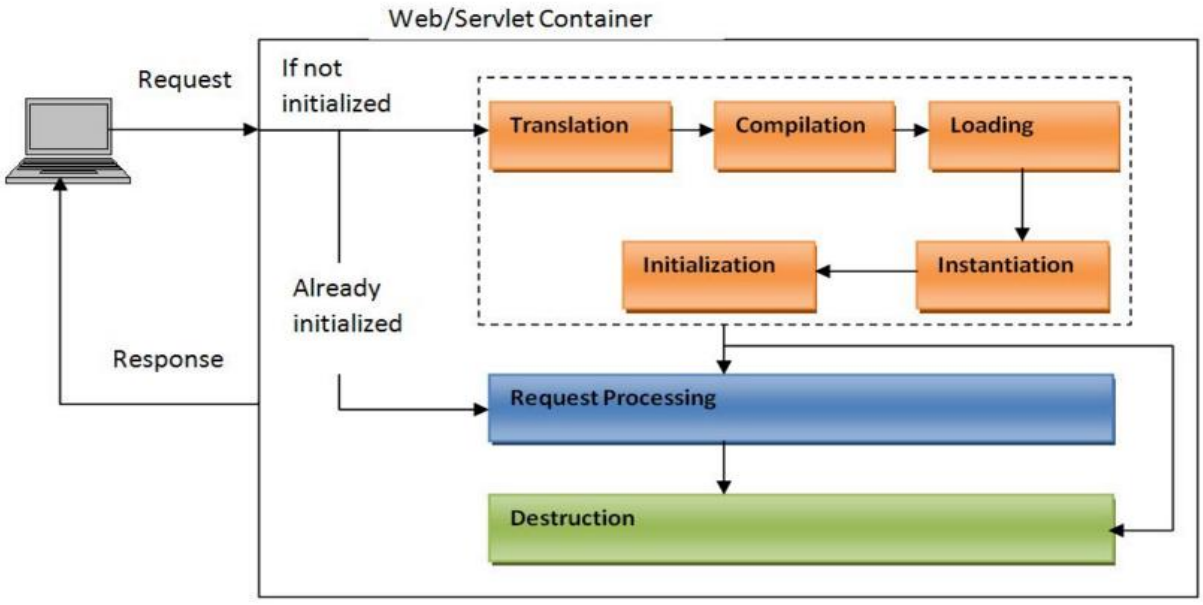


Where JspPage interface has declared the following methods.
public void _JspInit()
public void _JspDestroy()

Where HttpJspPage interface has provided the following method.
public void _JspService(HttpServletRequest request, HttpServletResponse response)

For the above 3 abstract methods HttpJspBase class has provided the default implementation but _JspService(_,_) method would be overridden in first_jsp class with the content what we provided in first.jsp file.

JSP Flow of execution:-



Translated Servlet:-

```
package org.apache.jsp;

import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;

public final class first_jsp extends org.apache.jasper.runtime.HttpJspBase
   implements org.apache.jasper.runtime.JspSourceDependent,
              org.apache.jasper.runtime.JspSourceImports {

 private static final javax.servlet.jsp.JspFactory _jspxFactory =
       javax.servlet.jsp.JspFactory.getDefaultFactory();
 private static java.util.Map<java.lang.String,java.lang.Long> _jspx_dependants;
 private static final java.util.Set<java.lang.String> _jspx_imports_packages;
 private static final java.util.Set<java.lang.String> _jspx_imports_classes;

 static {
  _jspx_imports_packages = new java.util.HashSet<>();
  _jspx_imports_packages.add("javax.servlet");
  _jspx_imports_packages.add("javax.servlet.http");
  _jspx_imports_packages.add("javax.servlet.jsp");
  _jspx_imports_classes = null;
 }

 private volatile javax.el.ExpressionFactory _el_expressionfactory;
 private volatile org.apache.tomcat.InstanceManager _jsp_instancemanager;

 public java.util.Map<java.lang.String,java.lang.Long> getDependants() {
  return _jspx_dependants;
 }
 public java.util.Set<java.lang.String> getPackageImports() {
  return _jspx_imports_packages;
 }
 public java.util.Set<java.lang.String> getClassImports() {
  return _jspx_imports_classes;
 }
 public javax.el.ExpressionFactory _jsp_getExpressionFactory() {
  if (_el_expressionfactory == null) {
   synchronized (this) {
    if (_el_expressionfactory == null) {
     _el_expressionfactory =
_jspxFactory.getJspApplicationContext(getServletConfig().getServletContext()).getExpressionFactory();
    }
   }
  }
  return _el_expressionfactory;
 }
```

```
  public org.apache.tomcat.InstanceManager _jsp_getInstanceManager() {
    if (_jsp_instancemanager == null) {
      synchronized (this) {
        if (_jsp_instancemanager == null) {
          _jsp_instancemanager =
org.apache.jasper.runtime.InstanceManagerFactory.getInstanceManager(getServletConfig());
        }
      }
    }
    return _jsp_instancemanager;
  }
//life cycle methods
  public void _jspInit() {  }
  public void _jspDestroy() { }
  public void _jspService(final javax.servlet.http.HttpServletRequest request, final
javax.servlet.http.HttpServletResponse response)
        throws java.io.IOException, javax.servlet.ServletException {

final java.lang.String _jspx_method = request.getMethod();
if (!"GET".equals(_jspx_method) && !"POST".equals(_jspx_method) && !"HEAD".equals(_jspx_method)
&& !javax.servlet.DispatcherType.ERROR.equals(request.getDispatcherType())) {
response.sendError(HttpServletResponse.SC_METHOD_NOT_ALLOWED, "JSPs only permit GET POST or
HEAD");
return;
}

    final javax.servlet.jsp.PageContext pageContext;
    javax.servlet.http.HttpSession session = null;
    final javax.servlet.ServletContext application;
    final javax.servlet.ServletConfig config;
    javax.servlet.jsp.JspWriter out = null;
    final java.lang.Object page = this;
    javax.servlet.jsp.JspWriter _jspx_out = null;
    javax.servlet.jsp.PageContext _jspx_page_context = null;

    try {
      response.setContentType("text/html");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                                null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write("\r\n");
      out.write("<html>\r\n");
```

```
    out.write("<body bgcolor=\"lightgreen\">\r\n");
    out.write(" <center><b><font size=\"7\" color=\"red\">\r\n");
    out.write("    First Jsp Application By Mr. Ratan\r\n");
    out.write("   </font></b></center>\t\r\n");
    out.write("</body>\r\n");
    out.write("</html>");
   } catch (java.lang.Throwable t) {
    if (!(t instanceof javax.servlet.jsp.SkipPageException)){
     out = _jspx_out;
     if (out != null && out.getBufferSize() != 0)
      try {
       if (response.isCommitted()) {
        out.flush();
       } else {
        out.clearBuffer();
       }
      } catch (java.io.IOException e) {}
     if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
     else throw new ServletException(t);
    }
   } finally {
    _jspxFactory.releasePageContext(_jspx_page_context);
   }
  }
}
```

Conclusion:- *The _jspInit() and _jspDestroy() methods are invoked once, but _jspService() method is executed each time a JSP page is requested.*
If there is any error during translation, then the container raises an exception with error code 500 (internal server error).

9-implicit objects of JSP:-

Implicit objects are automatically created by web container (Jsp Engine+servlet container) with predefined names.

Implicit object	*Type*	*Remark*
Request	Javax.servlet.http.HttpServletRequest	
Response	Javax.servlet.http.HttpServletResponse	
Config	Javax.servlet.ServletConfig	
Application	Javax.servlet.ServletContext	
Session	Javax.servlet.http.HttpSession	*It is created only when <%@ page session="true" %>, which is default.*
Out	Javax.servlet.jsp.JspWriter	*Specially designed for JSP.*
Page	Java.lang.Object	*Represents current servlet class reference i.e., current jsp page.*
pageContext	Javax.servlet.jsp.PageContext	*Specially designed for JSPs.*
exception	Java.lang.Throwable	*It is created only when <%@page isErrorPage="true" %>*

Example :- It is possible to configure the jsp file by using web.xml .

Note: *If we configure a jsp in web.xml then we can send the request to the jsp either by using jsp filename or by using its url-pattern.*

If we want to configure a JSP in web.xml file the xml elements are same as Servlet-configuration, We need to replace <servlet-class> element with <jsp-file>

Servlets are private files present in classes folder hence access these file by using web.xml
Jsp are public file hence we can access the public file elements directly by sending request from browser.

First.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
 this is the jsp file configured in web.xml
</body>
</html>
```

Web.xml:-

```
<web-app>
<servlet>
<servlet-name>FirstJsp</servlet-name>
<jsp-file>/first.jsp</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>FirstJsp</servlet-name>
<url-pattern>/first</url-pattern>
</servlet-mapping>
</web-app>
```

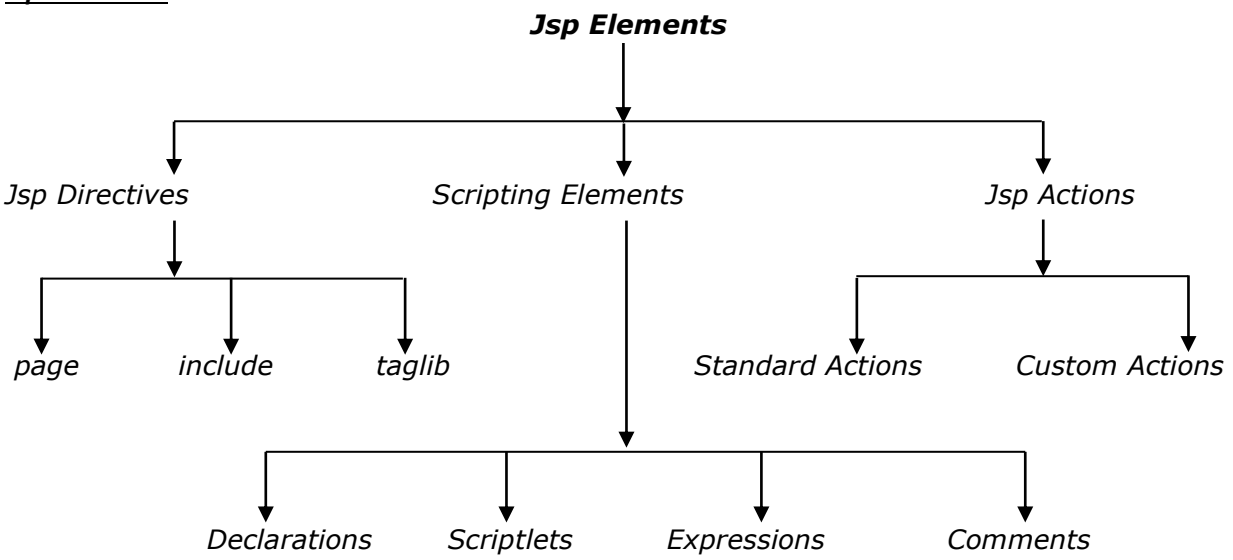
url pattern to access the jsp file:-

http://localhost:8888/JspApp1/first.jsp // direct accessing jsp file

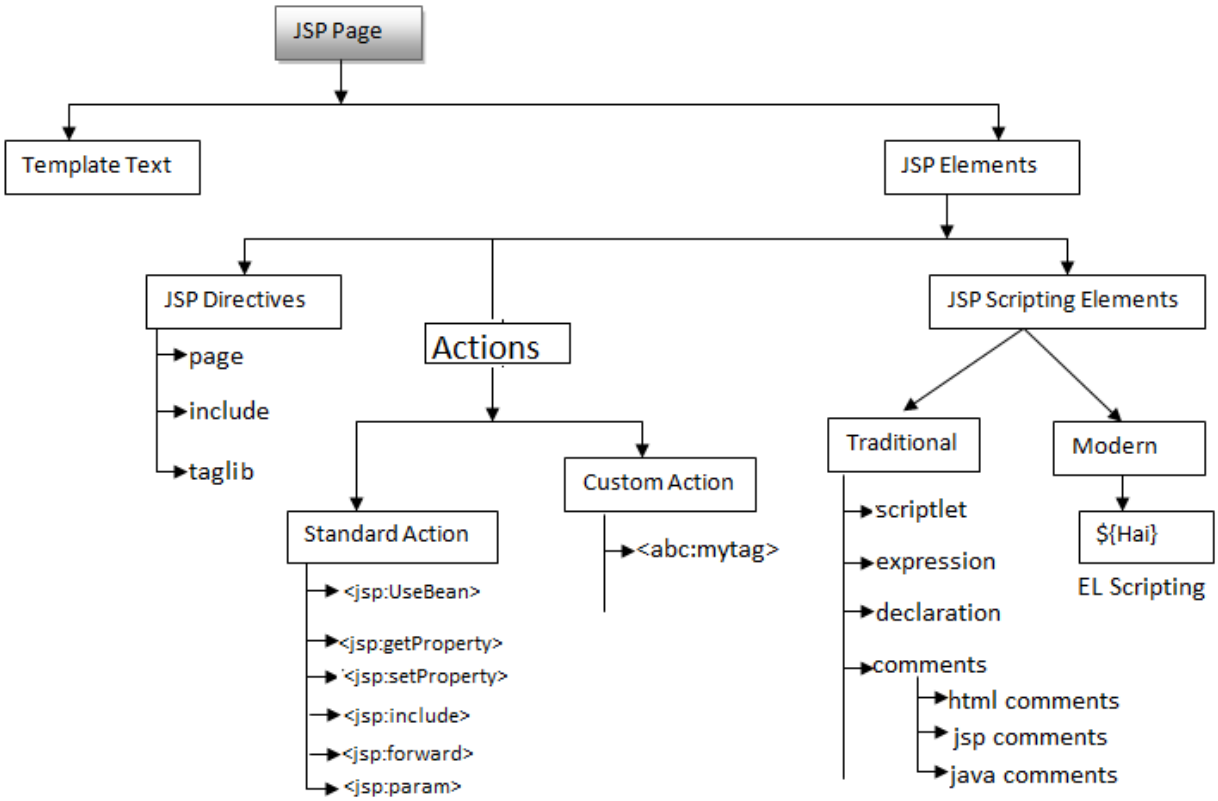
or

http://localhost:8888/JspApp1/first //accessing by using web.xml url pattern

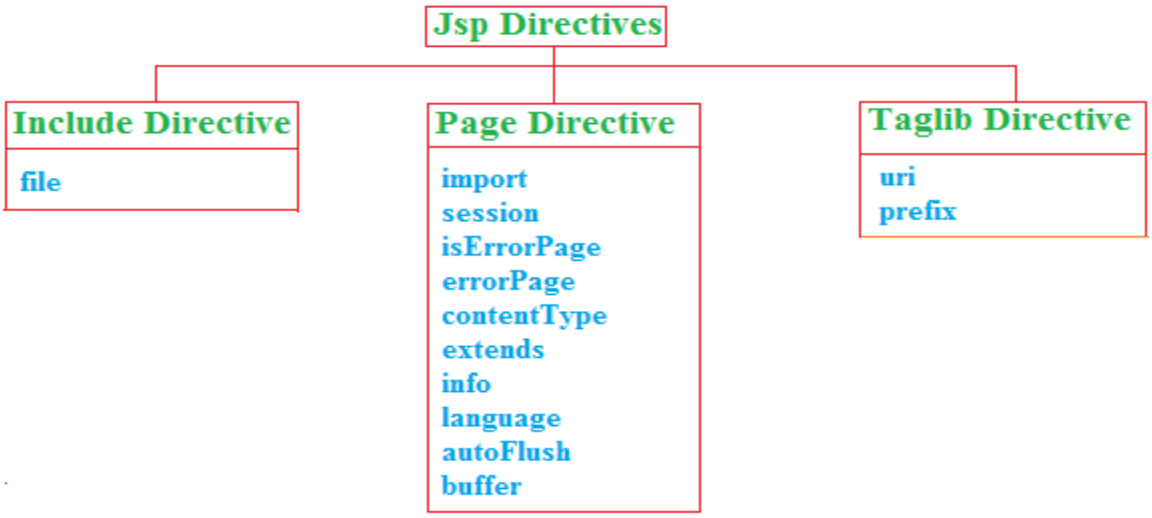
Jsp Elements:



Complete diagram of JSP elements:-



Jsp Directives:-



The jsp directives are messages that tell the web container how to translate a jsp page into the corresponding servlet.

There are three types of directives in jsp.

Directive	***Description***
<%@ page attribute="value" %>	*Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.*
<%@ include attribute="value" %>	*Includes a file during the translation phase.*
<%@ taglib attribute="value" %>	*Declares a tag library, containing custom actions, used in the page*

Page Directive:-

Page Directive can be used to define the present Jsp page characteristics like to define import statements, specify particular super class to the translated servlet, to specify metadata about present Jsp pages and so on.

<%@page [attribute-list]%>

Where attribute-list in Jsp page directive may include the following list.

Language	*contentType*	*import*	*extends*
Info	*buffer*	*autoFlush*	*errorPage*
isErrorPage	*session*	*isThreadSafe*	*isELIgnored*

- ✓ ***language:-***

  This attribute can be used to specify language.
  The default value of this attribute is java.

  <%@page language="java"%>

- ✓ ***contentType:***

  This attribute will take a particular MIME type in order to give an intimation to the client about to specify the type of response which Jsp page has generated.

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

  The default value of this attribute is text/html.

- ✓ ***import:***

  This attribute can be used to import a particular packages into the present Jsp pages.

  <%@page import="java.io.*"%>

  If we want to import multiple number of packages into the present Jsp pages then we have to use either of the following 2 approaches.

  Specify multiple number of packages with comma(,) .

  <%@page import="java.io.,java.util.*,java.sql.*"%>*

  Provide multiple number of import attributes for the list of packages.

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

The default values of this attribute are java.lang, javax.servlet, javax.servlet.http, javax.servlet.jsp.

Note: Among all the Jsp page attributes only import attribute is repeatable attribute, no other attribute is repeatable.

- ✓ ***extends:***

  This attribute will take a particular class name, it will be available to the translated servlet as super class.

  <%@page extends="com.dss.MyClass"%>

  Where MyClass should be an implementation class to HttpJspPage interface and should be a subclass to HttpServlet.

  The default value of this attribute is HttpJspBase class.

- ✓ ***info:***

  This attribute can be used to specify some metadata about the present Jsp page.

  <%@page info="First Jsp Application"%>

  If we want to get the specified metadata programmatically then we have to use the following method from Servlet interface.

  public String getServletInfo()

  The default value of this attribute is Jasper JSP2.2 Engine.

- ✓ ***buffer:***

  This attribute can be used to specify the particular size to the buffer available in JspWriter object.

  Note: Jsp technology is having its own writer object to track the generated dynamic response, JspWriter will provide very good performance when compared with PrintWriter in servlets.

  <%@page buffer="52kb"%>

  The default value of this attribute is 8kb.

- ***autoFlush:***

It is a boolean attribute, it can be used to give an intimation to the container about to flush or not to flush dynamic response to client automatically when JspWriter buffer filled with the response completely.

If autoFlush attribute value is true then container will flush the complete response to the client from the buffer when it reaches its maximum capacity.

If autoFlush attribute value is false then container will raise an exception when the buffer is filled with the response.

org.apache.jasper.JasperException:An exception occured processing JSP page/first.jsp at line:9 root cause: java.io.IOException:Error:Jsp Buffer Overflow.

Note: if we provide 0kb as value for buffer attribute and false as value for autoFlush attribute then container will raise an exception like org.apache.jasper.JasperException:/first.jsp(1,2) jsp.error.page.badCombo

The default value of this attribute is true.

*Application: -**form.jsp***

```
<%@ page language="java" contentType="text/html" buffer="1kb" autoFlush="true"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<%      for(int i=0; i<1000; i++) {
                                out.println("RAMA");      }
    %>
</body>
</html>
```

Observation :-

<%@ page language="java" contentType="text/html" buffer="1kb" autoFlush="false"%>

Caused by: java.io.IOException: Error: JSP Buffer overflow

- ***session:***

It is a Boolean attribute, it is give intimation to the container about to allow or not to allow session implicit object into the present Jsp page.The default value of this attribute is true.

Example: <%@page session="true"%>

<%@ page session="true" %>

The generated servlet class contains:

HttpSession session = null;

session = pageContext.getSession();

<%@ page session="**false**" %>

Above two lines won't be generated in servlet class.

- ✓ ***errorPage:***

  This attribute can be used to specify an error page to execute when we have an exception in the present Jsp page.

  <%@page errorPage="error.jsp"%>

- ✓ ***isErrorPage:***

It is a boolean attribute, it can be used to give an intimation to the container about to allow or not to allow exception implicit object into the present Jsp page.

If we provide value as true to this attribute then container will allow exception implicit object into the present Jsp page.

If we provide value as false to this attribute then container will not allow exception implicit object into the present Jsp page.

The default value of this attribute is false.

Example: <%@page isErrorPage="true"%>

first.jsp:

```
<%@ page language="java" contentType="text/html" import="java.util.*" errorPage="error.jsp"%>
<html>
<body>
<%!Date d=null;%>
<%
        out.println(d.toString());
%>
</body>
</html>
```

error.jsp:

```
<%@ page language="java" contentType="text/html" isErrorPage="true"%>
<html>
<body>
        <%=exception%>
</body>
</html>
```

- ***isThreadSafe:-***

It is a boolean attribute, it can be used to give an intimation to the container about to allow or not to allow multiple number of requests at a time into the present Jsp page.

If we provide true as value to this attribute then container will allow multiple number of requests at a time.

If we provide false as value to this attribute then automatically container will allow only one request at a time and it will implement SingleThreadModel interface in the translated servlet.

The default value of this attribute is true.

*Indicates whether the code in JSP already provides thread-safe (or) the generated servlet class to provide thread-safe by implementing **SingleThreadModel** interface.*

*<%@ page isThreadSafe="**true**" %>*

In this case, the code inside JSP already provides thread-safe, hence, the generated servlet class not required to implement SingleThreadModel interface. The JSP page can process any number of requests simultaneously.

*<%@page isThreadSafe="**false**" %>*

The code inside JSP is not thread-safe, hence, the generated servlet class must implements SingleThreadModel interface.

- ***isELIgnored:-***

It is a boolean attribute, it can be used to give an intimation to the container about to allow or not to allow Expression Language syntaxes in the present Jsp page.

Note: Expression Language is a Scripting language, it can be used to eliminate java code completely from the Jsp pages.

If isELIgnored attribute value is true then container will eliminate Expression Language syntaxes from the present Jsp page.

If we provide false as value to this attribute then container will allow Expression Language syntaxes into the present Jsp pages.

The default value of this attribute is false.

<%@page isELIgnored="true"%>

The default value in JSP1.2 version is: true.

The default value from JSP2.0 version is: false

Include Directive:

The common code across all JSP pages recommended to write once in a separate JSP page, and include this JSP across all other JSPs using <%@ include %> directive.

*The main advantage with this approach is to promote **Reusability** and **Maintainability**.*

<%@include file="--"%>

Where file attribute can be used to specify the name and location of the target resource.

logo.jsp:

```
<html>
<body><center>
<table width="100%" height="20%" bgcolor="red">
<tr><td colspan="2"><center><b><font size="7" color="white">
Ratan Software Solutions
</font></b></center></td></tr>
</table></center></body>
</html>
```

footer.jsp:

```
<html>
<body><center>
<table width="100%" height="15%" bgcolor="blue">
<tr><td colspan="2"><center><b><font size="6" color="white">
   copyrights2010-2020@Ratansoftwaresolutions
</font></b></center></td></tr>
</table></center></body>
</html>
```

body.jsp:

```
<html>
<body bgcolor="lightyellow">
<center><b><font size="7">
<p><br>
   Ratan Software Solutions is one of the Training Institute.
<br><br></p>
</font></b></center></body>
</html>
```

mainpage.jsp:

```
<%@include file="logo.jsp"%>
<%@include file="body.jsp"%>
<%@include file="footer.jsp"%>
```

Note : in including mechanism all the dependent file content Is included in main file during translation page hence it is static inclusion.

Mainpage_jsp.java:- (Translation File)

```
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class main_jsp extends org.apache.jasper.runtime.HttpJspBase
   implements org.apache.jasper.runtime.JspSourceDependent {
  public void _jspInit() {  }
  public void _jspDestroy() {  }
  public void _jspService(final javax.servlet.http.HttpServletRequest request, final
javax.servlet.http.HttpServletResponse response)     throws java.io.IOException,
javax.servlet.ServletException {
     out.write("<html>\r\n");
     out.write("    <body><center>\r\n");
     out.write("    <table width=\"100%\" height=\"20%\" bgcolor=\"red\">\r\n");
     out.write("    <tr><td colspan=\"2\"><center><b><font size=\"7\" color=\"white\">\r\n");
     out.write("     Ratan Software Solutions\r\n");
     out.write("     </font></b></center></td></tr>\r\n");
     out.write("     </table></center></body>\r\n");
     out.write("    <body bgcolor=\"lightyellow\">\r\n");
     out.write("    <center><b><font size=\"7\">\r\n");
     out.write("    <p><br>\r\n");
     out.write("    Ratan Software Solutions is one of the Training Institute.\r\n");
     out.write("    <br><br></p>\r\n");
     out.write("    </font></b></center></body>\r\n");
     out.write("    <table width=\"100%\" height=\"15%\" bgcolor=\"blue\">\r\n");
     out.write("    <tr><td colspan=\"2\"><center><b><font size=\"6\" color=\"white\">\r\n");
     out.write("    copyrights2010-2020@Ratansoftwaresolutions\r\n");
     out.write("    </font></b></center></td></tr>\r\n");
     out.write("    </table></center></body>\r\n");
     out.write("</html>\r\n");
      out.write("</body>\r\n");
     out.write("</html>");
   }
}
```

Taglib directive:-

We can use 'taglib' directive to make custom tags available to the JSP page.

Syntax:-

<%@ taglib uri="URI" prefix="prefix" %>

Example: -

*<%@ taglib uri="**http://struts.apache.org/tags-html**" prefix="**html**"%>*

JSP Scripting Elements:

Jsp scripting elements are enables you to write the java code inside the jsp file.
There are 3 types of Scripting Elements.

Expressions **: <%= expression%>** that are evaluated and inserted into output,
Scriptlets **: <% code %>** that are inserted into the servlets service method,
Declarations**: <%! code %>** that are inserted into the body of the servlet class, outside of any methods.

1) ***Declarations:***

It can be used to declare the variable declarations, method definitions, classes declaration…etc
The code written inside the jsp declaration tag is placed outside the service() method of servlet.

Syntax : **<%! *Field or method Declarations*; %>**

Example :

```
<% int i = 0;%>
<%! int add(int a,int b){
 return a+b; }
 %>
```

Application:-

```
<%@ page language="java" contentType="text/html" import="java.util.*"%>
<html>
<body>
<%! Date d;%>
<%      d=new Date();
        out.println(d);
 %>
<%=d%>
</body>
</html>
```

2) ***Scriptlets***:-*This Scripting Element can be used to provide a block of java code. This is code is placed inside _jspService() method.*

Syntax: **<% Block of java code %>**

Example :

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1" import="java.util.*" %>
    <html>
    <body>
        <!-- basic application -->
        <%!     Date d=null;     %>
        <%      d= new Date();  %>
         today date is=<%=d%>
    </body>
    </html>
```

If we provide any block of java code by using scriptlets then that code will be available to translated servlet inside _jspService(_,_) method.

3) Expressions:

This is scripting element can be used to evaluate the single java expression and display that expression resultant value onto the client browser.

Syntax: <%=Java Expression%>

If we provide any java expression in expression scripting element then that expression will be available to translated servlet inside _JspService(_,_) method as a parameter to out.write() method.
Ex: out.write(Java Expression);

4) Jsp Comments:-

In Jsp pages, we are able to use the following 3 types of comments.

- *XML-Based Comments*
- *Jsp-Based Comments*
- *Java-Based Comments inside Scripting Elements*

1. XML-Based Comments:

```
<!--
      St-1
      St -2 Description
     -->
```

2. *Jsp-Based Comments:*

```
<%--
        st-1
st-2Description
 --%>
```

3. *Java-Based Comments inside Scripting Elements:*

a. Single Line Comment:

```
//--------Description---------
```

b. Multiline Comment:

```
/*  -----
*/
```

c. Documentation Comment:-

```
/*
*
    *
*/
```

Request & response & out objects:-

Login Application :

Login.html

```
<html>
<body>
<form action=./Login.jsp method="post">
User name:<input type="text" name="username" /><br>
Password:<input type="password" name="password" /><br>
<input type="submit" value="login">
</form>
</body>
</html>
```

Login.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>

<%!
String uname;
String upwd;
%>

<%
uname = request.getParameter("username");
upwd = request.getParameter("password");
if(uname.equals("ratan") && upwd.equals("ratan"))
{       out.println("login success");
}
else
{       out.println("login failure");
}
%>
<br/>
user name : <%=uname %>
<br/>
user Password : <%=upwd %>
</body>
</html>
```

Note : check the converted servlet ***Login_jsp.java*** *in the fallowing location.*

1. *Declaration tag elements present in servlet class outside of the methods.*
2. *Script lets tag data present in _jspService() method.*
3. *Expression tag data present as a parameter of out.print(eid) method.*

Application Request & response:-

Login.html:

```
<html>
<body>
<form action=./Login.jsp method="post">
User name:<input type="text" name="username" /><br>
Password:<input type="password" name="password" /><br>
<input type="submit" value="login">
</form>
</body>
</html>
```

Login.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>

<%!
String uname;
String upwd;
%>

<%
uname = request.getParameter("username");
upwd = request.getParameter("password");
if(uname.equals("ratan") && upwd.equals("ratan"))
{                   response.sendRedirect("http://www.facebook.com");
}
else
{                   response.sendError(777,"login fail try with valid user name & password");
}
%>
<br/>
user name : <%=uname %>
<br/>
user Password : <%=upwd %>
</body>
</html>
```

Scopes in jsp :-

There are four scopes in jsp
Page
Request
Session
Application

Page scope:-

'page' scope means, the JSP object can be accessed only from within the same page where it was created.

The default scope for JSP objects created using <jsp:useBean> tag is page. JSP implicit objects out, exception, response, pageContext, config and page have 'page' scope.

Request scope:-

A JSP object created using the 'request' scope can be accessed from any pages that serves that request. More than one page can serve a single request. The JSP object will be bound to the request object.

Implicit object request has the 'request' scope.

Session scope :-

'session' scope means, the JSP object is accessible from pages that belong to the same session from where it was created. The JSP object that is created using the session scope is bound to the session object.

Implicit object session has the 'session' scope.

Application scope:-

A JSP object created using the 'application' scope can be accessed from any pages across the application. The JSP object is bound to the application object.

Implicit object application has the 'application' scope.

The following table summarizes more details about JSP scopes: -

Scope	**Maintained by**	**Methods**
Page	*PageContext* *<<abstract class>>*	*Public void setAttribute(String name, Object value)* *Public Object getAttribute(String name)* *Public void removeAttribute(String name)* *Public Enumeration getAttributeNamesInScope(int scope)*
Request	*ServletRequest* *<<interface>>*	*Public void setAttribute(String name, Object value)* *Public Object getAttribute(String name)* *Public void removeAttribute(String name)* *Public Enumeration getAttributeNames()*
Session	*HttpSession* *<<interface>>*	*Public void setAttribute(String name, Object value)* *Public Object getAtribute(String name)* *Public void removeAttribute(String name)* *Public Enumeration getAttributeNames()*
Application	*ServletContext* *<<interface>>*	*Public void setAttribute(String name, Object value)* *Public Object getAttribute(String name)* *Public void removeAttribute(String name)* *Public Enumeration getAttributeNames()*

PageContext :-

- ✓ *pageContext is a implicit object in jsp of type PageContext object.*
- ✓ *The PageContext object is used to set or get or remove the the attricutes form fallowing scopes,*
  - *Page*
  - *Request*
  - *Session*
  - *Application*

form.html:-

```
<html>
<body>
<form action="welcome.jsp" method="post">
Enter user name = <input type="text" name="uname"/>
<input type="submit" value="submit"/>
</form>
</body>
</html>
```

Welcome.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
        <%
        String uname = request.getParameter("uname");
        pageContext.setAttribute("uname",uname, PageContext.SESSION_SCOPE);

        out.println("<a href='second.jsp'>click here to get msg</a>");
        %>
        </body>
</html>
```

Second.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
<%String uname = (String)pageContext.getAttribute("uname",PageContext.SESSION_SCOPE);
        out.println("Welcome="+uname);
        %>
</body>
</html>
```

Config & application implicit objects:-

Form.html

```
<html><body>
		<a href="First">click here to get First jsp</a><br>
		<a href="Second">click here to get Second jsp</a><br>
</body></html>
```

Web.xml:-

```
<web-app>
<welcome-file-list>
		<welcome-file>form.html</welcome-file>
</welcome-file-list>

<context-param>
		<param-name>username</param-name>
		<param-value>ratan</param-value>
</context-param>
<context-param>
		<param-name>password</param-name>
		<param-value>Anu</param-value>
</context-param>

<servlet>
		<servlet-name>aaa</servlet-name>
		<jsp-file>/First.jsp</jsp-file>
		<init-param>
		<param-name>Animal</param-name>
		<param-value>Tiger</param-value>
		</init-param>
</servlet>
<servlet-mapping>
		<servlet-name>aaa</servlet-name>
		<url-pattern>/First</url-pattern>
</servlet-mapping>

<servlet>
		<servlet-name>bbb</servlet-name>
		<jsp-file>/Second.jsp</jsp-file>
		<init-param>
		<param-name>Fruit</param-name>
		<param-value>Mango</param-value>
		</init-param>
</servlet>
<servlet-mapping>
		<servlet-name>bbb</servlet-name>
		<url-pattern>/Second</url-pattern>
</servlet-mapping>

</web-app>
```

First.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
        <%!String animal;
        String uname;
        String upwd;
        %>

        <%animal = config.getInitParameter("Animal");
        uname = application.getInitParameter("username");
        upwd = application.getInitParameter("password");
        %>

        Init param-1 <%=animal%><br>
        context param-1 <%=uname%><br>
        context param-2  <%=upwd%><br>
</body>
</html>
```

Second.java

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
        <%!     String animal;
        String uname;
        String upwd;
        %>

        <%      animal = config.getInitParameter("Fruit");
        uname = application.getInitParameter("username");
        upwd = application.getInitParameter("password");
        %>

        Init param-1  <%=animal%><br>
        context param-1  <%=uname%><br>
        context param-2 <%=upwd%><br>
</body>
</html>
```

Session implicit object:-

- ✓ *A session is an object associated with a visitor. We can set the data & get data of particular user.*
- ✓ *The data is visible until the user is invalidate the session.once the user invalidate the session we will get null values.*

session.invalidate();

When you declare session=false in jsp it is not possible to use session object inside the Jsp.

```
<%@ page language="java" contentType="text/html" session="false"%>
```

Form.html:-

```
<html>
<body>
<form method="post" action="savename.jsp">
what's your name? <input type="text"  name="uname" size=20>
<p><input type=submit value="submit">
</form>
</body>
</html>
```

SaveName.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
        <%   String name = request.getParameter("uname");
                session.setAttribute( "theName", name );
        %>
<a href="NextPage.jsp">Continue</A>
</body>
</html>
```

NextPage.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<head>
Hello,<%= session.getAttribute("theName")%>
</body>
</html>
```

Example :-

Form1.html

```
<html>
<body bgcolor="pink">
<center>    <form method="get" action="./First.jsp">
            Name : <input type="text" name="uname"/><br><br>
            Age : <input type="text" name="uage"/><br><br>
            <input type="submit" value="Next"/>
            </form>
</center>
</body>
</html>
```

First.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
		<%!String name;
		int age;
		%>
		<%   name  = request.getParameter("uname");
		age = Integer.parseInt(request.getParameter("uage"));

		session.setAttribute("name", name);
		session.setAttribute("age", age);
		%>
		<jsp:forward page="form2.html"/>
</body>
</html>
```

Form2.html:-

```
<html>
<body bgcolor="pink">
<center>
		<form method="get" action="./second.jsp">
		Qualification : <input type="text" name="uqual"/><br><br>
		Designation : <input type="text" name="udesig"/><br><br>
		<input type="submit" value="Next"/>
		</form>
</center>
</body>
</html>
```

Second.jsp:-

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
		<%!String qual;
		String desig;
		%>

		<%	qual  = request.getParameter("uqual");
		desig = request.getParameter("udesig");
		%>

		User name : <%=session.getAttribute("name")%><br>
		User Age : <%=session.getAttribute("age")%><br>
		User Qualifications : <%=qual%><br>
		User Designation : <%=desig%><br>
</body>
</html>
```

<u>Exception handling in Jsp:-</u>

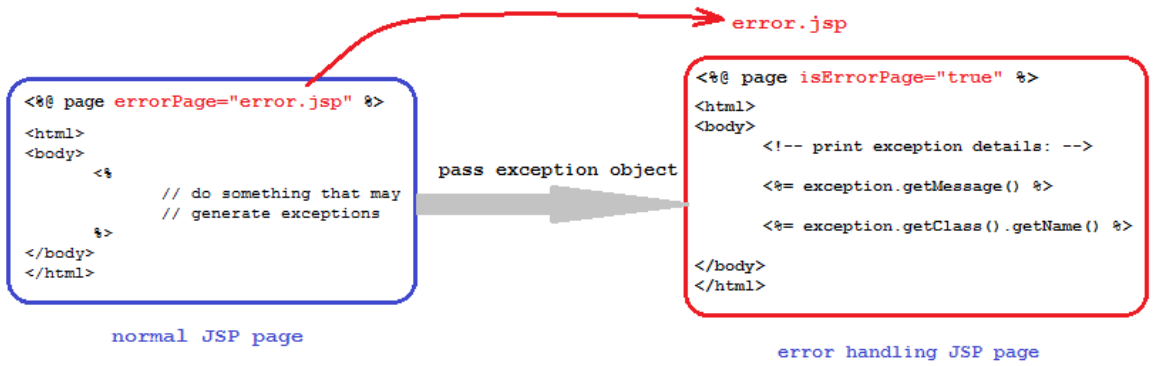
We can handle the exceptions in 3-ways

1. *Programmatic approach using simple try-catch*
2. *Programmatic approach. Using isErrorPage errorPage*
3. *Declarative approach. Using <error-page> tag in Deployment Descriptor.*

Programmatic approach. Using try-catch:-

```
<html>
<body>
<%
        try{
            out.println(10/0);
        }
        catch (ArithmeticException e){
          out.println("An exception occurred: " + e.getMessage());
        }
 %>
</body>
</html>
```

Programmatic approach. Using isErrorPage errorPage



This approach specify one error for all kind of exceptions, so we can't specify the error page different exceptions like,

SQLException ---> dbError.jsp

IOException ---> ioerror.jsp

When the file name of error page is changed we have to update all referenced jsp files.

Exception handling programmatic Approach :-

Process.jsp:-

```
<%@ page language="java"  errorPage="error.jsp" contentType="text/html"%>
<html>
<body>
        <%out.print("ratan".charAt(20));  %>
</body>
</html>
```

Web.xml:-

```
<web-app>
    <error-page>
        <exception-type>java.lang.StringIndexOutOfBoundsException</exception-type>
        <location>/error.jsp</location>
    </error-page>
</web-app>
```

error.jsp:-

```
<%@ page contentType="text/html"%>
<html>
<body>
<%out.print("hi Mr.RATAN StringIndexOutOfBounds Exception raised");
%>
</body>
</html>
```

- ✓ *The above error pages can be used across all JSPs in a given web application. Declarative approach is always recommended.*
- ✓ *In error page no need to specify the isErrorPage attribute.*

Q)*What is the difference between PrintWriter and JspWriter?*

Ans: PrintWriter is a writer in servlet applications, it can be used to carry the response. PrintWriter is not BufferedWriter so that its performance is very less in servlets applications.

JspWriter is a writer in Jsp technology to carry the response. JspWriter is a BufferedWriter so that its performance will be more when compared with PrintWriter.

Q: What is PageContext? and What is the purpose of PageContext in Jsp Applications?

Ans: *PageContext is an implicit object in Jsp technology, it can be used to get all the Jsp implicit objects even in non-jsp environment.*

To get all the Jsp implicit objects from PageContext we have to use the following method.

public Xxx getXxx()

Where Xxx may be out, request, response and so on.

Ex: JspWriter out=pageContext.getOut();
HttpServletRequest request=pageContext.getRequest();
ServletConfig config=pageConetxt.getServletConfig();

Note: While preparing Tag Handler classes in custom tags container will provide pageContext object as part of its life cycle, from this we are able to get all the implicit objects.

In Jsp applications, by using pageContext object we are able to perform some operations with the attributes in Jsp scopes page, request, session and application like adding an attribute, removing an attribute, getting an attribute and finding an attribute.

To set an attribute onto a particular scope pageContext has provided the following method.
public void setAttribute(String name, Object value, int scope)

Where scopes may be
public static final int PAGE_SCOPE=1;
public static final int REQUEST_SCOPE=2;
public static final int SESSION_SCOPE=3;
public static final int APPLICATION_SCOPE=4;

Example:

<% pageContext.setAttribute(“a”, “aaa”, pageConext.REQUEST_SCOPE); %>

To get an attribute from page scope we have to use the following method.
public Object getAttribute(String name)

Ex: String a=(String)pageContext.getAttribute(“a”);

If we want to get an attribute value from the specified scope we have to use the following method.
public Object getAttribute(String name, int scope)

Ex: String uname=(String)pageContext.getAttribute(“uname”, pageContext.SESSION_SCOPE);

To remove an attribute from a particular we have to use the following method.
public void removeAttribute(String name, int scope)

Ex: pageContext.removeAttribute(“a”, pageContext.SESSION_SCOPE);

To find an attribute value from page scope, request scope, session scope and application scope we have to use the following method.
Public Object findAttribute(String name)

Ex: String name=pageContext.findAttribute(“uname”);

Jsp Actions:

In Jsp technology, by using scripting elements we are able to provide java code inside the Jsp pages, but the main theme of Jsp technology is not to allow java code inside Jsp pages.

To eliminate java code from Jsp pages we have to eliminate scripting elements, to eliminate scripting elements from Jsp pages we have to provide an alternative i.e. Jsp Actions.

In case of Jsp Actions, we will define a scripting tag in Jsp page and we will provide a block of java code w.r.t. scripting tag.

When container encounters the scripting tag then container will execute respective java code, by this an action will be performed called as Jsp Action.

In Jsp technology, there are 2 types of actions.

1. Standard Actions
2. Custom Actions

1. Standard Actions:

Standard Actions are Jsp Actions, which could be defined by the Jsp technology to perform a particular action.

Jsp technology has provided all the standard actions in the form of a set of predefined tags called Action Tags.

1. <jsp:useBean---->
2. <jsp:setProperty---->
3. <jsp:getProperty---->
4. <jsp:include---->
5. <jsp:forward---->
6. <jsp:param---->
7. <jsp:plugin---->
8. <jsp:fallback---->
9. <jsp:params---->
10. <jsp:declaration---->
11. <jsp:scriptlet---->
12. <jsp:expression---->

Java Beans:

Java Bean is a reusable component.

Java Bean is a normal java class which may declare properties, setter and getter methods in order to represent a particular user form at server side.

If we want to prepare Java Bean components then we have to use the following rules and regulations.

1. Java Bean is a normal java class, it is suggestible to implement Serializable interface.
2. Always Java Bean classes should be public, non-abstract and non-final.
3. In Java Bean classes, we have to declare all theproperties w.r.t. the properties define in the respective user form.
4. In Java Bean classes, all the properties should be private.
5. In Java Bean classes, all the behaviours should be public.
6. If we want to declare any constructor in Java Bean class then that constructor should be public and zero argument.

<jsp:useBean>:

- ✓ *The main purpose of <jsp:useBean> tag is to interact with bean object from a particular Jsp page.*
- ✓ *This tag used to locate or instantiate bean class. If the bean object is already created it doesn't create a bean based on scope. But if the bean is not created it instantiate the bean.*

<jsp:useBean id="--" class="--" type="--" scope="--"/>
Id = instance name
Class = fully qualified name of the bean class
Scope =page request session application (scope of the bean object)

Note*: In <jsp:useBean> tag, always it is suggestible to provide either application or session scope to the scope attribute value.*

When container encounters the above tag then container will pick up class attribute value i.e. fully qualified name of Bean class then container will recognize Bean class .class file and perform Bean class loading and instantiation.

After creating Bean object container will assign Bean object reference to the variable specified as value to id attribute.

After getting Bean object reference container will store Bean object in a scope specified as value to scope attribute.

<jsp:setProperty>:

The main purpose of <jsp:setProperty> tag is to execute a particular setter method in order to set a value to a particular Bean property.

Syntax: <jsp:setProperty name="--" property="--" value="--"/>

Where name attribute will take a variable which is same as id attribute value in <jsp:useBean> tag.
Where property attribute will take a property name in order to access the respective setter method.
Where value attribute will take a value to pass as a parameter to the respective setter method.

3. <jsp:getProperty>:

The main purpose of <jsp:getProperty> tag is to execute a getter method in order to get a value from Bean object.

Syntax: <jsp:getProperty name="--" property="--"/>

Where name attribute will take a variable which is same as id attribute value in <jsp:useBean> tag.
Where property attribute will take a particular property to execute the respective getter method.

Note: In case of <jsp:useBean> tag, in general we will provide a separate <jsp:setProperty> tag to set a particular value to the respective property in Bean object.

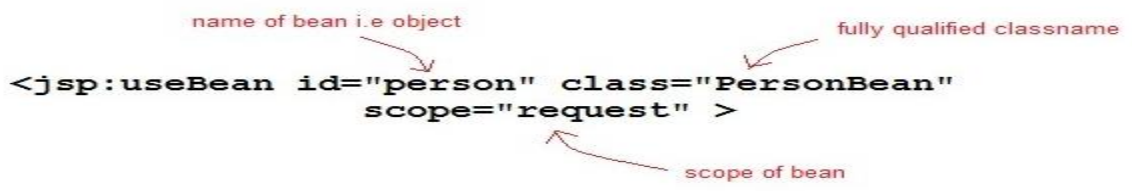
In case of <jsp:useBean> tag, it is possible to copy all the request parameter values directly onto the respective Bean object.

To achieve this we have to provide "*" as value to property attribute in <jsp:setProperty> tag.

Ex: <jsp:setProperty name="e" property="*"/>

If we want to achieve the above requirement then we have to maintain same names in the request parameters i.e. form properties and Bean properties.

Note: The above "*" notation is not possible with <jsp:getProperty> tag.



Form.html

```
<html>
<h5> *******Employee Details********</h5>
<body>
<form method="post" action="main.jsp">
        Employee id :<input type="text" name="eid"/><br>
        Employee name : <input type="text" name="ename"/><br>
        Employee salary :<input type="text" name="esal"/></br>
       <input type="submit" value="submit"/>
</form>
</body>
</html>
```

Main.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
        <%!     int eid;
                String ename;
                double esal;
        %>
        <%      eid = Integer.parseInt(request.getParameter("eid"));
                ename = request.getParameter("ename");
                esal = Double.parseDouble(request.getParameter("esal"));
        %>
        <jsp:useBean id="eb" class="com.sravya.EmpBean" scope="page"/>
        <jsp:setProperty property="ename" name="eb" value="<%=ename%>"/>
        <jsp:setProperty property="eid" name="eb" value="<%=eid%>"/>
        <jsp:setProperty property="esal" name="eb" value="<%=esal%>"/>
        <h5>**********Employee Details**********</h5>
        Employee name : <jsp:getProperty property="ename" name="eb"/><br>
        Employee id  : <jsp:getProperty property="eid" name="eb"/>              <br>
        Employee salary:  <jsp:getProperty property="esal" name="eb"/><br>
</body>
</html>
```

EmpBean.java

```
package com.sravya;
import java.io.Serializable;
publicclass EmpBean implements Serializable{
	int eid;
	String ename;
	double esal;
	publicint getEid() {
		return eid;
	}
	publicvoid setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	publicvoid setEname(String ename) {
		this.ename = ename;
	}
	publicdouble getEsal() {
		return esal;
	}
	publicvoid setEsal(double esal) {
		this.esal = esal;
	}
}
```

4. <jsp:include>:

Q: What are the differences between include directive and <jsp:include> action tag?

Ans: 1. In Jsp applications, **include directive** can be used to include the content of the target resource into the present Jsp page.

In Jsp pages, **<jsp:include>** action tag can be used to include the target resource response into the present Jsp page response.

2. In general include directive can be used to include static resources where the frequent updations are not available.

<jsp:include> action tag can be used to include dynamic resources where the frequent updations are available.

3. In general directives will be resolved at the time of translation and actions will be resolved at the time of request processing. Due to this reason include directive will be resolved at the time of translation but <jsp:include> action tag will be resolved at the time of request processing.

If we are trying to include a target Jsp page into present Jsp page by using include directive then container will prepare only one translated servlet.

To include a target Jsp page into the present Jsp page if we use <jsp:include> action tag then container will prepare 2 separate translated servlets.

In Jsp applications, include directives will provide static inclusion, but <jsp:include> action tag will provide dynamic inclusion.

In Jsp technology, <jsp:include> action tag was designed on the basis of Include Request Dispatching Mechanism.

Syntax:<jsp:include page="--" flush="--"/>

Where page attribute will take the name and location of the target resource to include its response.

Where flush is a boolean attribute, it can be used to give an intimation to the container about to autoFlush or not to autoFlush dynamic response to client when JspWriter buffer filled with the response at the time of including the target resource response into the present Jsp page.

Working with Database :-

Form.html:-

```
<html>
	<body bgcolor="lightgreen">
	<form action="add.jsp">
		<pre>
			<u>Product Details</u>
			Product Id : <input type="text" name="pid"/>
			Product Name : <input type="text" name="pname"/>
			Product Cost : <input type="text" name="pcost"/>
			<input type="submit" value="ADD"/>
		</pre>
	</form>
	</body>
</html>
```

Add.jsp:-

```
<%@page import="java.sql.*"%>
<%!
	String pid;
	String pname;
	int pcost;
	static Connection con;
	static Statement st;
	ResultSet rs;
	ResultSetMetaData rsmd;
	static{
		try{
			Class.forName("oracle.jdbc.driver.OracleDriver");
	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "ratan");
			st=con.createStatement();
		}
		catch(Exception e){
			e.printStackTrace();
		}
	}
%>
<%		pid=request.getParameter("pid");
		pname=request.getParameter("pname");
		pcost=Integer.parseInt(request.getParameter("pcost"));
		st.executeUpdate("insert into product values('"+pid+"','"+pname+"',"+pcost+")");
		out.println("values are inserted successfully");
%>
<jsp:include page="form.html"/>
```

Include Directive:-

1) *It is used to include the target jsp page into present jsp page.*
2) *Include directive include the target page data into present jsp file at translation time(jsp-servlet).*
3) *If any modification done in jsp file will not be visible until jsp file compiles again.*
4) *Include directive is static import.*
5) *It is better for static pages, if we are using for dynamic pages for every change it requires translation.*
6) *By using include directive we are unable to pass parameters to target page.*

Syntax:- <%@ include file="file_name" %>

Example:- <%@ include file="hello.jsp" %>

Include Action:-

1) *It is used to include the target jsp into present jsp page.*
2) *The content of the target jsp included at runtime instead of translation time.*
3) *If you done the modifications these are effected when we send the request to jsp.*
4) *Include action is dynamic import.*
5) *Instead of including original data it is calling include method at runtime.*
6) *By using include action we are able to pass parameters to the included page.*

```
<jsp:include page="file_name" />
        <jsp:param name="parameter_name" value="parameter_value" />
</jsp:include>
```

Syntax:- <jsp:include page="file_name" />

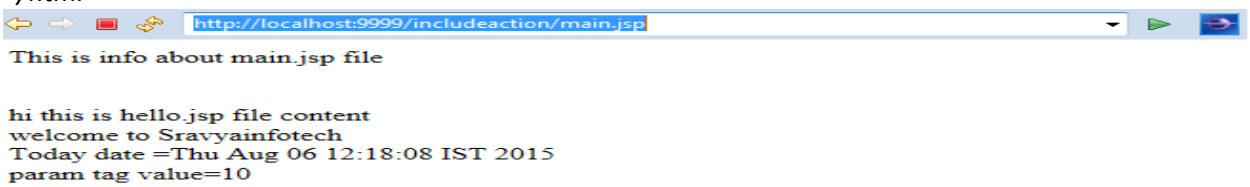
Example:- <jsp:include page="hello.jsp"/>

main.jsp:-

```
<%@ page language="java" contentType="text/html" %>
<html>
<body>
 This is info about main.jsp file<br><br><br>
<jsp:include page="hello.jsp">
        <jsp:param name="a" value="10"/>
</jsp:include>
</body>
</html>
```

hello.jsp:-

```
<%@ page language="java" contentType="text/html" %>
<%@page  import="java.util.*"%>
<html>
<head>
<body>
        hi this is hello.jsp file content<br>
        welcome to Sravyainfotech<br>
        <%="Today date ="+new Date()%><br>
        <%="param tag value="+request.getParameter("a")%>
</body>
</html>
```



5. <jsp:forward>:

Q: What are the differences between<jsp:include> action tag and <jsp:forward> action tag?

Ans: 1. **<jsp:include>** action tag can be used to include the target resource response into the present Jsp page.

<jsp:forward> tag can be used to forward request from present Jsp page to the target resource.

2. <jsp:include> tag was designed on the basis of Include Request Dispatching Mechanism.

<jsp:forward> tag was designed on the basis of Forward Request Dispatching Mechanism.

3. When Jsp container encounter <jsp:include> tag then container will forward request to the target resource, by executing the target resource some response will be generated in the response object, at the end of the target resource container will bypass request and response objects back to first resource, at the end of first resource execution container will dispatch overall response to client. Therefore, in case of <jsp:include> tag client is able to receive all the resources response which are participated in the present request processing.

When container encounters<jsp:forward> tag then container will bypass request and response objects to the target resource by refreshing response object i.e. by eliminating previous response available in response object, at the end of target resource container will dispatch the generated dynamic response directly to the client without moving back to first resource. Therefore, in case of <jsp:forward> tag client is able to receive only target resource response.

Syntax:<jsp:forward page="--"/>

Where page attribute specifies the name and location of the target resource.

<jsp:include>&<jsp:forward> :-

login.html

```
<html>
<body>
<form action="main.jsp">
        <h5> *******Login Details*******</h5>
        User name : <input type="text" name="uname"/><br>
        Password : <input type="password" name=upwd/><br>
        <input type="submit" value="login"/>
</form>
</body>
</html>
```

Main.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
	<%!	String uname;
		String upwd;
	%>
		<%
		String uname = request.getParameter("uname");
		String upwd = request.getParameter("upwd");
		if(uname.equals("sravya"))
		{
		%>
		<jsp:forward page="success.jsp"/>
		<%
		}
		else
		{out.println("****U R Login is fail try with another username  & password*******");
		}%>
			<jsp:include page="login.html"/>
</body>
</html>
```

success.jsp

```
<%@ page language="java" contentType="text/html"%>
<html>
<body>
	Hi this is Success jsp<br>
	U r login is Success
</body>
</html>
```

6. <jsp:param>:

This action tag can be used to provide a name value pair to the request object at the time of by passing request object from present Jsp page to target page either in include mechanism or in forward mechanism or in both.

This tag must be utilized as a child tag to <jsp:include> tag and <jsp:forward> tags.

Syntax:<jsp:param name=”--” value=”--”/>

----------Application4----------

forwardapp:

registrationform.html:

```
<html>
	<body bgcolor="lightgreen">
	<form action="registration.jsp">
		<pre>
			<u>Registration Form</u>
			Name : <input type="text" name="pname"/>
			Age : <input type="text" name="uage"/>
			Address : <input type="text" name="uaddr"/>
			<input type="submit" value="Registration"/>
		</pre>
	</form>
	</body>
</html>
```

existed.jsp:

```
<center><h1>User Existed</h1></center>
```

success.jsp:

```
<center><h1>Registration Success</h1></center>
```

failure.jsp:

```
<center><h1>Registration Failure</h1></center>
```

registration.jsp:

```
<%@page import="java.sql.*"%>
<%!
	String uname;
	int uage;
	String uaddr;
	static Connection con;
	static Statement st;
	ResultSet rs;
	static{
		try{
			Class.forName("oracle.jdbc.driver.OracleDriver");

	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"ratan");
			st=con.createStatement();
		}
		catch(Exception e){
			e.printStackTrace();
		}
	}
%>
```

```
<%
	try{
		uname=request.getParameter("uname");
		uage=Integer.parseInt(request.getParameter("uage"));
		uaddr=request.getParameter("uaddr");
		rs=st.executeQuery("select * from reg_users where uname='"+uname+"'");
		boolean b=rs.next();
		if(b==true)
		{
%>
	<jsp:forward page="existed.jsp"/>
<%
		}
		else
		{
			int rowCount=st.executeUpdate("insert into reg_users values
('"+uname+"',"+uage+",'"+uaddr+"')");
			if(rowCount == 1)
			{
%>
	<jsp:forward page="success.jsp"/>
<%
			}
			else
			{
%>
	<jsp:forward page="failure.jsp"/>
<%
			}
		}
	}
	catch(Exception e){
%>
	<jsp:forward page="failure.jsp"/>
<%
		e.printStackTrace();
	}
%>
```

7. <jsp:plugin>:

This tag can be used to include an applet into the present Jsp page.

Syntax:<jsp:plugin code=”--” width=”--” height=”--” type=”--”/>

Where code attribute will take fully qualified name of the applet.

Where width and height attributes can be used to specify the size of applet.

Where type attribute can be used to specify which one we are going to include whether it is applet or bean.

Ex:<jsp:plugin code=”Logo” width=”1000” height=”150” type=”applet”/>

8. <jsp:params>:

In case of the applet applications, we are able to provide some parameters to the applet in order to provide input data.

Similarly if we want to provide input parameters to the applet from <jsp:plugin> tag we have to use <jsp:param> tag.
<jsp:param> tag must be utilized as a child tag to <jsp:params> tag.
<jsp:params> tag must be utilized as a child tag to <jsp:plugin> tag.

Syntax:

```
<jsp:plugin>
<jsp:params>
<jsp:param name=”--” value=”--”/>
<jsp:param name=”--” value=”--”/>
          ----------
</jsp:params>
    ----------
</jsp:plugin>
```

If we provide any input parameter to the applet then that parameter value we are able to get by using the following method from Applet class.

public String getParameter(String name)

Ex: String msg=getParameter(“message”);

pluginapp:
LogoApplet.java:

```
import java.awt.*;
import java.applet.*;
public class LogoApplet extends Applet
{
        String msg;
        public void paint(Graphics g)
        {
        msg=getParameter("message");
        Font f=new Font("arial",Font.BOLD,40);
        g.setFont(f);
        this.setBackground(Color.blue);
        this.setForeground(Color.white);
        g.drawString(msg,150,70);
        }
}
```

logo.jsp:

```
<jsp:plugin code="LogoApplet" width="1000" height="150" type="applet">
        <jsp:params>
        <jsp:param name="message" value="ratan software solutions"/>
        </jsp:params>
</jsp:plugin>
```

9. <jsp:fallback>:

The main purpose of <jsp:fallback> tag is to display an alternative message when client browser is not supporting <OBJECT---> tag and <EMBED---> tag.

Syntax:<jsp:fallback>-------Description---------</jsp:fallback>

In Jsp applications, we have to utilize <jsp:fallback> tag as a child tag to <jsp:plugin> tag.

Ex:<jsp:plugin code="LogoApplet" width="1000" height="150" type="applet">
<jsp:fallback>Applet Not Allowed</jsp:fallback>
</jsp:plugin>

10. <jsp:declaration>:

This tag is almost all same as the declaration scripting element, it can be used to provide all the Java declarations in the present Jsp page.

Syntax:<jsp:declaration>

-------- } Java Declarations

</jsp:declaration>

11. <jsp:scriptlet>:

This tag is almost all same as thescripting element scriptlets, it can be used to provide a block of Java code in Jsp pages.

Syntax:<jsp:scriptlet>

-------- } Block of Java code

</jsp:scriptlet>

12. <jsp:expression>:

This tag is almost all same as the scripting element expression, it can be used to provide a Java expression in the present Jsp page.

Syntax:<jsp:expression> Java Expression </jsp:expression>

```
<%@page import="java.util.*"%>
<jsp:declaration>
Date d=null;
String date=null;
</jsp:declaration>
<jsp:scriptlet>
d=new Date();
date=d.toString();
</jsp:scriptlet>
<html>
<body bgcolor="lightyellow">
<center><b><font size="6" color="red"><br><br>
Today Date : <jsp:expression>date</jsp:expression>
</font></b></center></body>
</html>
```

2. Custom Actions:

In Jsp technology, by using Jsp directives we are able to define present Jsp page characteristics, we are unable to use Jsp directives to perform actions in the Jsp pages.

To perform actions if we use scripting elements then we have to provide Java code inside the Jsp pages.

The main theme of Jsp technology is not to allow Java code inside Jsp pages, to eliminate Java code from Jsp pages we have to eliminate scripting elements.

If we want to eliminate scripting elements from Jsp pages we have to use actions.

There are 2 types of actions in Jsp technology.

1. Standard Actions
2. Custom Actions

Standard Actions are predefined actions provided by Jsp technology, these standard actions are limited in number and having bounded functionalities so that standard actions are not sufficient to satisfy the complete client requirement.

In this context, there may be a requirement to provide Java code inside the Jsp pages so that to eliminate Java code completely from Jsp pages we have to use Custom Actions.

Custom Actions are Jsp Actions which could be prepared by the developers as per their application requirements.

In Jsp technology, standard actions will be represented in the form of a set of predefined tags are called as **Action Tags.**

Similarly all the custom actions will be represented in the form of a set of user defined tags are called as **Custom Tags.**

To prepare custom tags in Jsp pages we have to use the following syntax.

Syntax:<prefix_Nmae:tag_Name>

        -------- } Body

</prefix_Name>

If we want to design custom tags in our Jsp applications then we have to use the following 3 elements.

1. Jsp page with taglib directive
2. TLD(Tag Library Descriptor) file
3. TagHandler class

Where TagHandler class is a normal Java class it is able to provide the basic functionality for the custom tags.

Where TLD file is a file, it will provide the mapping between custom tag names and respective TagHandler classes.

Where taglib directive in the Jsp page can be used to make available the tld files into the present Jsp page on the basis of custom tags prefix names.

Internal Flow:

When container encounters a custom tag container will pick up the custom tag name and the respective prefix name then recognize a particular taglib directive on the basis of the prefix attribute value.

After recognizing taglib directive container will pick up uri attribute value i.e. the name and location of tld file then container will recognize the respective tld file.

After getting tld file container will identify the name and location of TagHandler class on the basis of custom tag name.

When container recognize the respective TagHandler class .class file then container will perform TagHandler class loading, instantiation and execute all the life cycle methods.

1. Taglib Directive:

In custom tags design, the main purpose of taglib directive is to make available the required tld file into the present Jsp page and to define prefix names to the custom tags.

Syntax:<%@taglib uri="--" prefix="--"%>

Where prefix attribute can be used to specify a prefix name to the custom tag, it will have page scope i.e. the specified prefix name is valid up to the present Jsp page.

Where uri attribute will take the name and location of the respective tld file.

2. TLD File:

The main purpose of TLD file is to provide the mapping between custom tag names and the respective TagHandler classes and it is able to manage the description of the custom tags attributes.

To provide the mapping between custom tag names and the respective TagHandler class we have to use the following tags.

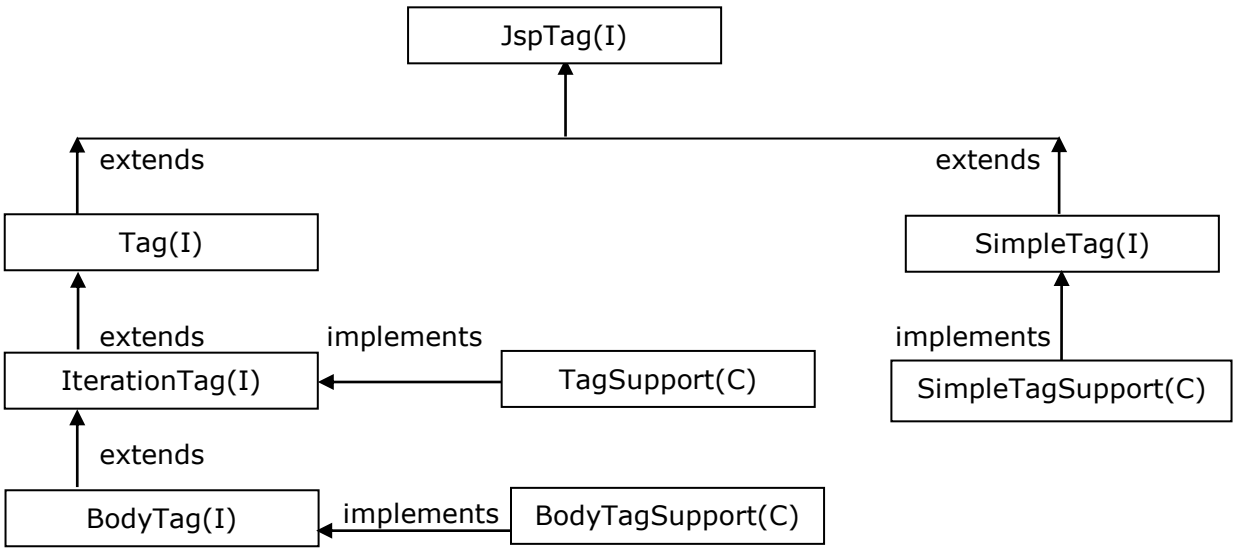
```
<taglib>
<jsp-version>jsp version</jsp-version>
<tlib-version>tld file version</tlib-version>
<short-name>tld file short name</short-name>
<description>description about tld file</description>
<tag>
<name>custom tag name</name>
<tag-class>fully qualified name of TagHandler class</tag-class>
<body-content>jsp or empty</body-content>
<short-name>custom tag short name</short-name>
<description>description about custom tags</description>
</tag>
        ---------
</taglib>
```

3. TagHandler class:

In custom tags preparation, the main purpose of TagHandler class is to define the basic functionality for the custom tags.

To design TagHandler classes in custom tags preparation Jsp API has provided some predefined library in the form of javax.servlet.jsp.tagext package (tagext-→tag extension).

javax.servlet.jsp.tagext package has provided the following library to design TagHandler classes.



The above Tag library was divided into 2 types.

1. Classic Tag library
2. Simple Tag library

As per the tag library provided by Jsp technology there are 2 types of custom tags.

1. Classic tags
2. Simple Tags

1. Classic Tags:

Classic Tags are the custom tags will be designed on the basis of Classic tag library provided by Jsp API.

As per the classic tag library provided by Jsp API there are 3 types of classic tags.

1. Simple Classic Tags
2. Iterator Tags
3. Body Tags

1. Simple Classic Tags:

Simple Classic Tags are the classic tags, which should not have body and attributes list.

To design simple classic tags the respective TagHandler class must implement Tag interface either directly or indirectly.

Predefined support:-

```
package javax.servlet.jsp.tagext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.PageContext;
public interface Tag extends JspTag
{       public abstract void setPageContext(PageContext pagecontext);
        public abstract void setParent(Tag tag);
        public abstract Tag getParent();
        public abstract int doStartTag()throws JspException;
        public abstract int doEndTag()throws JspException;
        public abstract void release();
        public static final int SKIP_BODY = 0;
        public static final int EVAL_BODY_INCLUDE = 1;
        public static final int SKIP_PAGE = 5;
        public static final int EVAL_PAGE = 6;
}
```

Userdefined class:-

```
public class MyHandler extends Tag
{       //write the code here
}
```

Where the purpose of setPageContext(_) method is to inject pageContext implicit object into the present TagHandler class.

Where the purpose of setParent(_) method is to inject parent tags TagHandler class object into the present TagHandler class.

Where the purpose of getParent() method is to return the parent tags TagHandler class object from the TagHandler class.

Where the purpose of doStartTag() method is to perform a particular action when container encounters the start tag of the custom tag.

After the custom tags start tag evaluating the custom tag body is completely depending on the return value provided by doStartTag () method.

There are 2 possible return values from doStartTag() method.

1. EVAL_BODY_INCLUDE
2. SKIP_BODY

If doStartTag() method returns EVAL_BODY_INCLUDE constant then container will evaluate the custom tag body.

If doStartTag() method returns SKIP_BODY constant then container will skip the custom tag body and encounter end tag.

Where the purpose of doEndTag() method is to perform an action when container encounters end tag of the custom tag.

Evaluating the remaining the Jsp page after the custom tag or not is completely depending on the return value provided by doEndTag() method.

There are 2 possible return values from doEndTag() method.

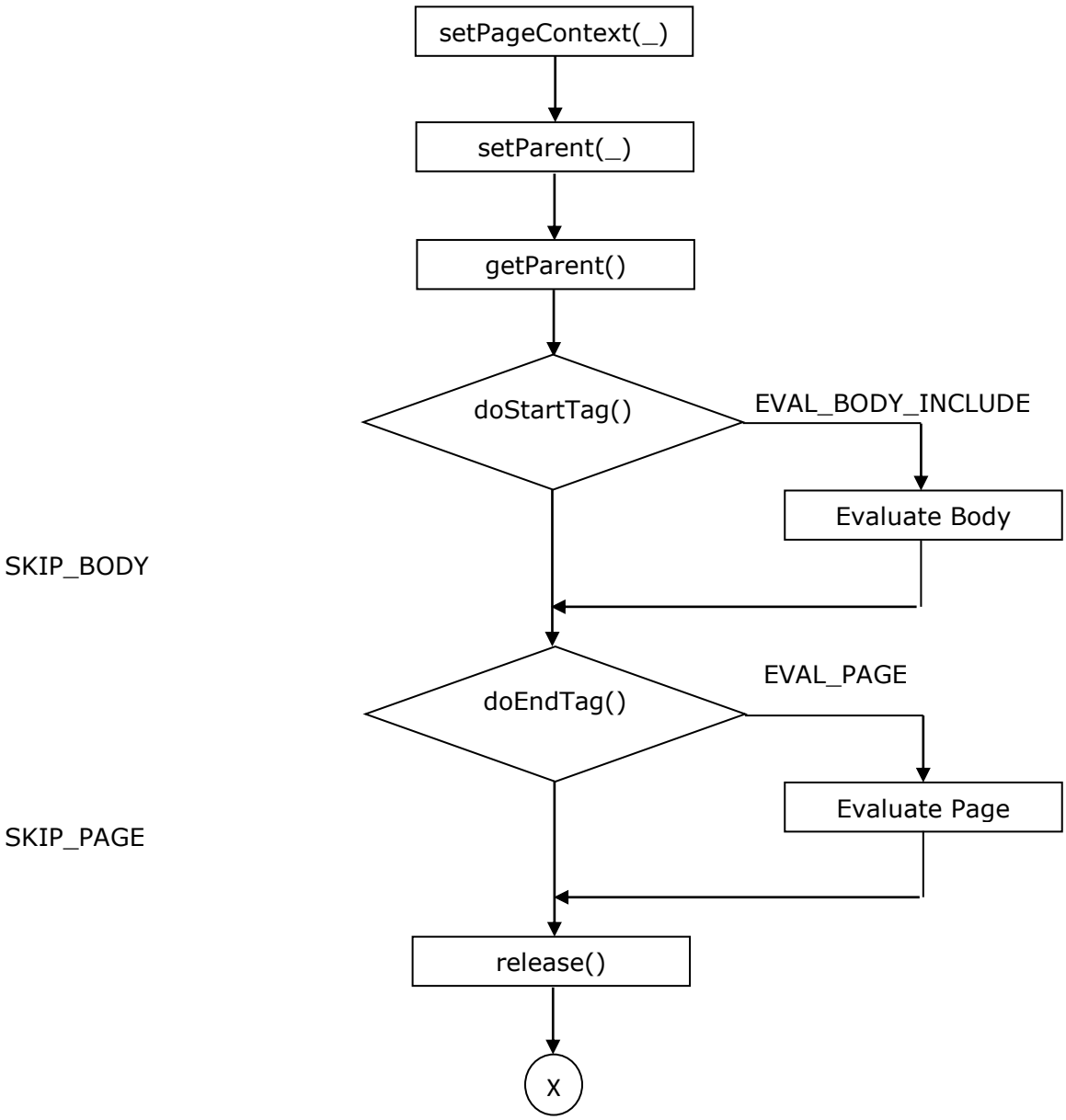
1. EVAL_PAGE
2. SKIP_PAGE

If doEndTag() method returns EVAL_PAGE constant then container will evaluate the remaining Jsp page.

If doEndTag() method returns SKIP_PAGE constant then container will not evaluate the remaining Jsp page.

Where release() method can be used to perform TagHandler class deinstantiation.

Life Cycle of Tag interface:



Note: In Tag interface life cycle, container will execute getParent() method when the present custom tag is child tag to a particular parent tag otherwise container will not execute getParent() method.

Steps to design custom tag application:-

Step1:- create he TagHandler class to provide the action

Step 2:- create the Tag Library Discriptor file(tld) file to configure custom tags.

Step 3:- write the jsp file to declare the custom tag configured in tld file.

Application:-

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello>
        hi ratan how r u
</mytags:hello>
<br>
this is remaining page of jsp
```

hello.tld:-

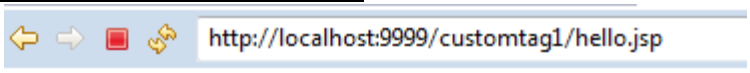
```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
                <tag>
                        <name>hello</name>
                        <tag-class>com.tcs.Customtag</tag-class>
                        <body-content>jsp</body-content>
                </tag>
</taglib>
```

Customtag.java:-

```
package com.tcs;
import java.io.IOException;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.Tag;
publicclass Customtag implements Tag{
        PageContext context;
        publicint doEndTag() throws JspException {
                System.out.println("doEndTag method");
                returnSKIP_PAGE;
        }
        publicint doStartTag() throws JspException {
                JspWriter writer =  context.getOut();
                try {
                        writer.println("this is custom tag application by Ratan");
                        writer.print("<br>");
                } catch (IOException e) {
                        e.printStackTrace();
                }
                returnEVAL_BODY_INCLUDE;
        }
```

```
	public Tag getParent() {
		System.out.println("getParent method");
		returnnull;
	}
	publicvoid release() {	}
	publicvoid setPageContext(PageContext arg0) {
		this.context=arg0;
		System.out.println("page context method");
	}
	publicvoid setParent(Tag arg0) {
		System.out.println("set parent method");
	}
}
```

The above example output:-



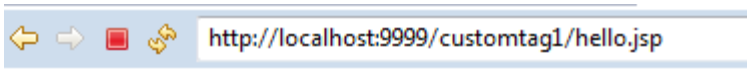
this is custom tag application by Ratan
hi ratan how r u

Server console check the life cycle methods:-

```
INFO: Server startup in 796 ms
page context method
set parent method
doEndTag method
```

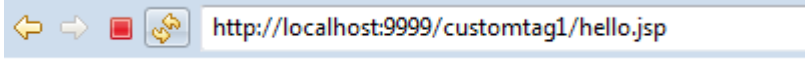
Observations :-

```
returnEVAL_BODY_INCLUDE;returnSKIP_PAGE;
```



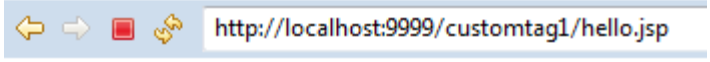
this is custom tag application by Ratan
hi ratan how r u

```
returnSKIP_BODY;returnSKIP_PAGE;
```



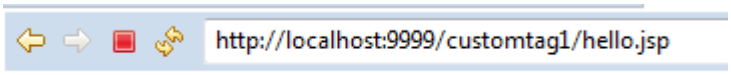
this is custom tag application by Ratan

```
returnEVAL_BODY_INCLUDE;	returnEVAL_PAGE;
```



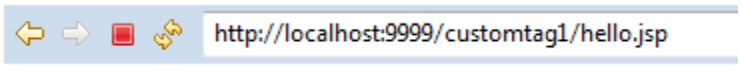
this is custom tag application by Ratan
hi ratan how r u
this is remaining page of jsp

```
returnEVAL_BODY_INCLUDE;	returnSKIP_PAGE;
```



this is custom tag application by Ratan
hi ratan how r u

```
return 5;                                    return 1;
```



this is custom tag application by Ratan
hi ratan how r u

Note: *To compile above code we need to set the classpath environment variable to the location of jsp-api.jar file.*

Observations:

Case 1: *In the above application, if we provide <body-content> type is empty in the tld file and if we provide body to the custom tag then container will raise an Exception like*

org.apache.jasper.JasperException: /hello.jsp(2,0) According to TLD, tag mytags:hello must be empty, but is not

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello>
        hi ratan how r u
</mytags:hello>
```

hello.tld:-

```
<taglib>
                **********
                <body-content>empty</body-content>
                **********
</taglib>
```

Case 2:*If we provide <body-content> value as jsp in tld file, if we provide body to custom tag in jsp page page and if we return SKIP_BODY constant in the respective TagHandler class then container won't raise any Exception but container won't evaluate custom tag body.*

Attributes in Custom Tgas:-

In custom tag it is possible to provide multiple attributes.

If we want to provide attributes in custom tags then we have to perform the following steps.

Step 1: Define attribute in the custom tag.

<mytags:hello name="Ratan"/>

Step 2: Provide attributes description in the respective tld file.

To provide attributes description in tld file we have to use the following tags in tld file.

```
<taglib>
            ************
    <tag>
    *******
    <attribute>
<name>attribute_name</name>
<required>true/false</required>
<rtexprvalue>true/false</rtexprvalue>
</attribute>
</tag>
</taglib>
```

Where <attribute> tag can be used to represent a single attribute in the tld file.

Where <name> tag will take attribute name.

Where <required> tag is a boolean tag, it can be used to specify whether the attribute is mandatory or optional.

Where <rtexprvalue> tag can be used to specify whether the attribute accept runtime values or not.

Step 3: Declare a property and setter method in TagHandler class with the same name of the attribute defined in custom tag.

```
public class MyHandler implements Tag {
   private String name;
   public void setName(String name) {
      this.name=name;
   }
       *****************
 }
```

2. Iterator Tags:

- Iterator tags are the custom tags, it will allow to evaluate custom tag body repeatedly.
- If we want to prepare iterator tags the respective TagHandler class must implement javax.servlet.jsp.tagext.IterationTag interface.

Predefined support:-

```
package javax.servlet.jsp.tagext;
import javax.servlet.jsp.JspException;
public interface IterationTag extends Tag
{
        public abstract int doAfterBody()throws JspException;
        public static final int EVAL_BODY_AGAIN = 2;
}
```

Userdefined class:-

```
class MyHandler implements IterationTag
{       //write the body here
}
```

In general there are 2 possible return values from **doStartTag()** method.

1) ***EVAL_BODY_INCLUDE***
   If we return EVAL_BODY_INCLUDE constant from doStartTag() method then container will execute the custom tag body.
2) ***SKIP_BODY***
   If we return SKIP_BODY constant from doStartTag() method then container will skip the custom tag body.

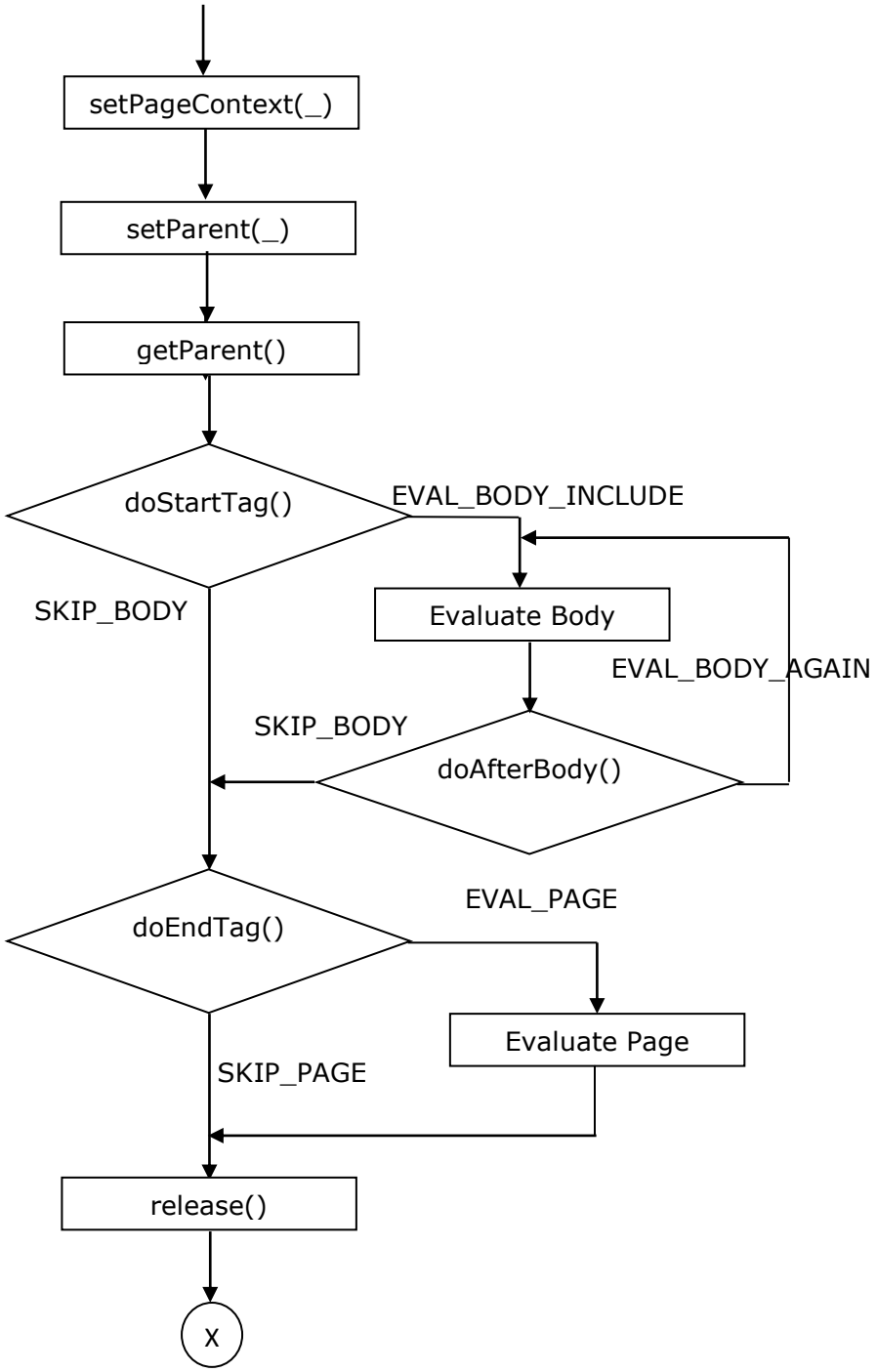
Note: In case of iterator tags, we must return EVAL_BODY_INCLUDE from doStartTag() method.

After evaluating the custom tag body in case of iterator tags, container will access doAfterBody() method.

In the above context, evaluating the custom tag body again or not is completely depending on the return value which we are going to return from doAfterBody() method.

1) ***EVAL_BODY_AGAIN***
   If we return EVAL_BODY_AGAIN constant from doAfterBody() method then container will execute the custom tag body again.
2) ***SKIP_BODY***
   If we return SKIP_BODY constant from doAfterBody() method then container will skip out custom tag body evaluation and encounter end tag of the custom tag.

Life Cycle of IterationTag interface:



If we want to design custom tagsby using above approach then the respective TagHandler class must implement Tag interface and IterationTag interface i.e. we must provide the implementation for all the methods which are declared in Tag and IterationTag interfaces in our TagHandler class.

This approach will increase burden to the developers and unnecessary methods in TgaHandler classes.

To overcome the above problem Jsp API has provided an alternative in the form of TagSupport class.

TagSupport is a concrete class, which was implemented Tag and IterationTag interfaces with the default implementation.

If we want to prepare custom tags with the TagSupport class then we have to take an user defined class, which must be a subclass to TagSupport class.

Predefined support:-

```
public interface TagSupport implements IterationTag {
    public static final int EVAL_BODY_INCLUDE;
    public static final int SKIP_BODY;
    public static final int EVAL_PAGE;
    public static final int SKIP_PAGE;
    public static final int EVAL_BODY_AGAIN;
    public PageContext pageContext;
    public Tag t;
    public void setPageContext(PageContext pageContext) {
        this.pageContext=pageContext;
    }
    public void setParent(Tag t) {
        this.t=t;
    public Tag getParent() {
        return t;
    }
    public int doStartTag()throws JspException {
        return SKIP_BODY;
    }
    public int doAfterBody()throws JspException {
        return SKIP_BODY;
    }
    public int doEndTag()throws JspException {
        return EVAL_PAGE;
    }
    public void release() { }
  }
```

User defined class:-

```
public class MyHandler implements TagSupport
{
}
```

Application:-

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:iterate times="10"><br>
        Hi Ratan how r u
</mytags:iterate>
```

hello.tld:-

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>iterate</name>
                <tag-class>com.sravya.Iteration</tag-class>
                <body-content>jsp</body-content>
                <attribute>
                        <name>times</name>
                        <required>true</required>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
</taglib>
```

Iteration.java:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
publicclass Iteration extends TagSupport{
        privateint times;
        privateint count=1;
        publicvoid setTimes(int times) {
                this.times = times;
        }
        @Override
        publicint doAfterBody() throws JspException {
                if(count<times)
                {       count++;
                        returnEVAL_BODY_AGAIN;
                }
                else
                {       returnSKIP_BODY;
                }
        }
        @Override
        publicint doStartTag() throws JspException {
                returnEVAL_BODY_INCLUDE;
        }
}
```

Application:-

Hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:loop start="1" end="20"><br>
        Hi Ratan how r u
</mytags:loop>
remaining page of the JSP
```

hello.tld:-

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>loop</name>
                <tag-class>com.sravya.Iteration</tag-class>
                <body-content>jsp</body-content>
                <attribute>
                        <name>start</name>
                        <required>true</required>
                </attribute>
                <attribute>
                        <name>end</name>
                        <required>true</required>
                </attribute>
        </tag>
</taglib>
```

Iteration.jsp:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
publicclass Iteration extends TagSupport{
        privateint start;
        privateint end;
        publicvoid setStart(int start) {
                this.start = start;          }
        publicvoid setEnd(int end) {
                this.end = end;  }
        @Override
        publicint doAfterBody() throws JspException {
                if(end>start)
                {       start++;
                        returnEVAL_BODY_AGAIN;      }
                else
                {       returnEVAL_PAGE;       }
        }
        @Override
        publicint doStartTag() throws JspException {
                returnEVAL_BODY_INCLUDE;    }
}
```

Nested Tags:-

- Defining a tag inside a tag is called as Nested Tag.
- In custom tags application, if we declare any nested tag then we have to provide a separate configuration in tld file and we have to prepare a separate TagHandler class under classes folder.

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:if condition='<%=10<20%>'>
        <mytags:true>condition is true</mytags:true><br>
        <mytags:false>condition is false</mytags:false><br>
</mytags:if>
rest of the JSP code
```

hello.tld:-

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>if</name>
                <tag-class>com.sravya.If</tag-class>
                <body-content>jsp</body-content>
                <attribute>
                        <name>condition</name>
                        <required>true</required>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
        <tag>
                <name>true</name>
                <tag-class>com.sravya.True</tag-class>
                <body-content>jsp</body-content>
        </tag>
        <tag>
                <name>false</name>
                <tag-class>com.sravya.False</tag-class>
                <body-content>jsp</body-content>
        </tag>
</taglib>
```

If.java:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
publicclassIfextends TagSupport{
	privateboolean condition;
	publicvoid setCondition(boolean condition) {
		this.condition = condition;
	}
	publicboolean getCondition()
	{		return condition;
	}
	@Override
	publicint doStartTag() throws JspException {
		returnEVAL_BODY_INCLUDE;
	}
	@Override
	publicint doEndTag() throws JspException {
		returnEVAL_PAGE;
	}
}
```

True.java:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
publicclass True extends TagSupport {
	@Override
	publicint doStartTag() throws JspException {
		If f = (If)getParent();
		boolean b = f.getCondition();
		if(b==true)
		{	returnEVAL_BODY_INCLUDE;
		}
		else {
			returnSKIP_BODY;
		}
	}
}
```

False.java:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.TagSupport;
publicclassFalseextends TagSupport {
        @Override
        publicint doStartTag() throws JspException {
                If f = (If)getParent();
                boolean b = f.getCondition();
                if(b==true)
                {       returnSKIP_BODY;
                }
                else
                {       returnEVAL_BODY_INCLUDE;
                }
        }
}
```

empdetails.jsp:

```
<%@taglib uri="/WEB-INF/emp.tld" prefix="emp"%>
<emp:empDetails/>
```

emp.tld:

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>empDetails</name>
                <tag-class>com.dss.EmpDetails</tag-class>
                <body-content>empty</body-content>
        </tag>
</taglib>
```

EmpDetails.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
public class EmpDetails extends TagSupport
{
        Connection con;
        Statement st;
        ResultSet rs;
        public EmpDetails()
        {
                try
```

```
		{
			Class.forName("oracle.jdbc.driver.OracleDriver");
	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","
ratan");
			st=con.createStatement();
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}

public int doStartTag() throws JspException
	{
		try
		{
			JspWriter out=pageContext.getOut();
			rs=st.executeQuery("select * from emp");
			ResultSetMetaData rsmd=rs.getMetaData();
			int count=rsmd.getColumnCount();
			out.println("<html><body bgcolor='pink'>");
			out.println("<center><br><br>");
			out.println("<table border='1' bgcolor='lightyellow'>");
			out.println("<tr>");
			for (int i=1;i<=count;i++)
			{
				out.println("<td><b><font size='6'
color='red'><center>"+rsmd.getColumnName(i)+"</center></font></b></td>");
			}
			out.println("</tr>");
			while (rs.next())
			{
				out.println("<tr>");
				for (int i=1;i<=count;i++)
				{
					out.println("<td><b><font
size='5'>"+rs.getString(i)+"</font></b></td>");
				}
				out.println("</tr>");
			}
			out.println("</table></center></body></html>");
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
		return SKIP_BODY;
	}
}
```

3. Body Tags:

- Up to now in our custom tags (simple classic tags, iteration tags) we prepared custom tags with the body, where we did not perform updations over the custom tag body, just we scanned custom tag body and displayed on the client browser.
- If we want to perform updations over the custom tag body then we have to use Body Tags.
  If we want to design body tags in Jsp technology then the respective TagHandler class must implement BodyTag interface either directly or indirectly.

BodyTag:-

```
package javax.servlet.jsp.tagext;
import javax.servlet.jsp.JspException;
public interface BodyTag extends IterationTag
{public abstract void setBodyContent(BodyContent bodycontent);
  public abstract void doInitBody()throws JspException;
  public static final int EVAL_BODY_TAG = 2;
  public static final int EVAL_BODY_BUFFERED = 2;}
```

All methods at inherited level:-

```
public interface BodyTag extends IterationTag
  { public static final int EVAL_BODY_INCLUDE;
    public static final int SKIP_BODY;
    public static final int EVAL_PAGE;
    public static final int SKIP_PAGE;
    public static final int EVAL_BODY_AGAIN;
    public static final int EVAL_BODY_BUFFERED;
    public void setPageContext(PageContext pageContext);
    public void setParent(Tag t);
    public Tag getParent();
    public int doStartTag()throws JspException;
    public void doInitBody()throws JspException;
    public void setBodyContent(BodyContent bodyContent);
    public int doAfterBody()throws JspException;
    public int doEndTag()throws JspException;
    public void release(); }
```

User defined class:-

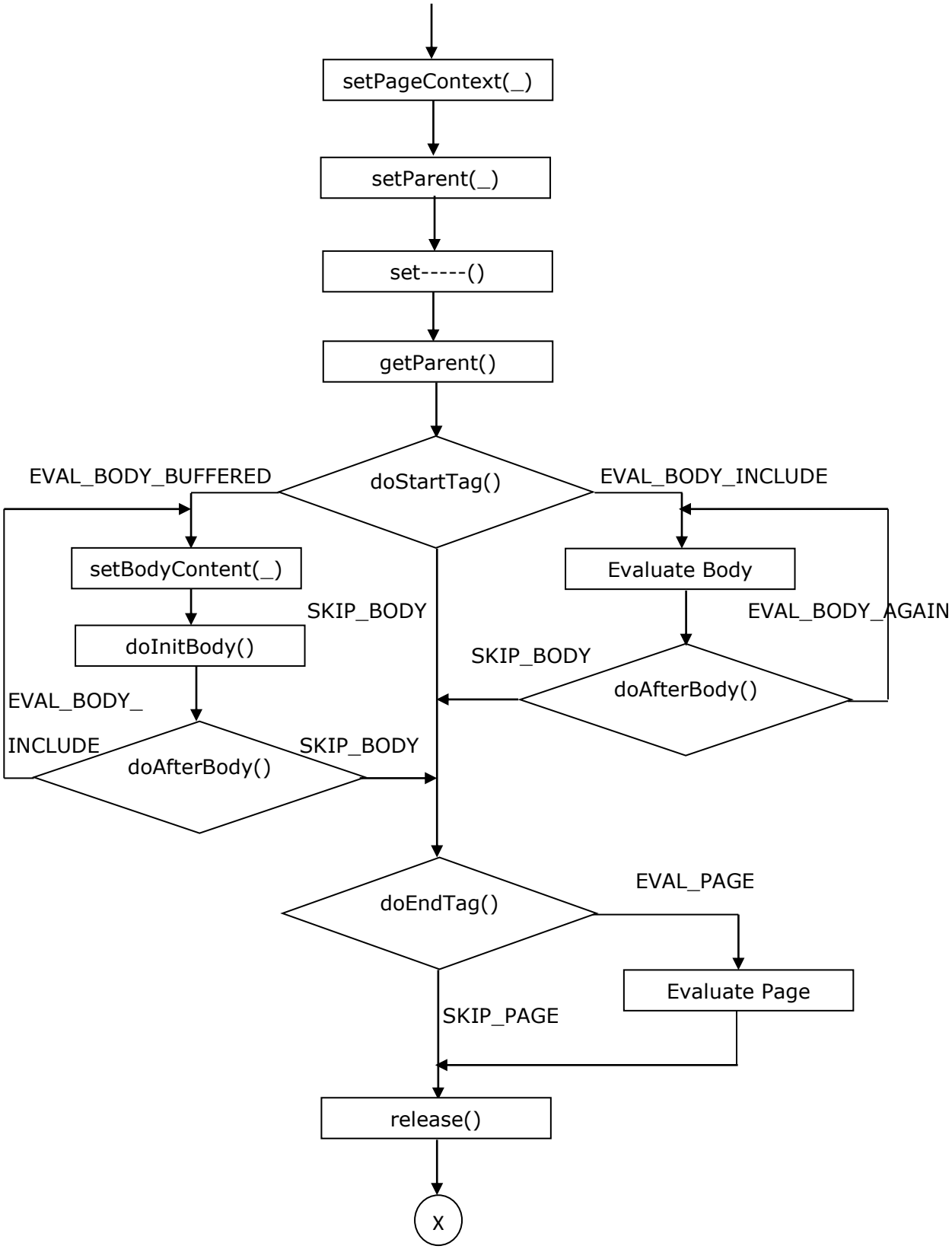
```
  public class MyHandler implements BodyTag
  {     //write the logics here;     }
```

In case of body tags, there are 3 possible return values from doStartTag() method.

a) EVAL_BODY INCLUDE
   If we return EVAL_BODY_INCLUDE constant from doStartTag() method then container will evaluate the custom tag body i.e. display as it is the custom tag body on client browser.
b) SKIP_BODY
   If we return SKIP_BODY constant from doStartTag() method then container will skip the custom tag body.
c) EVAL_BODY_BUFFERED
   If we return EVAL_BODY_BUFFERED constant from doStartTag() methodthen container will store custom tag body in a buffer then access setBodyContent(_) method.

To access setBodyContent(_) method container will prepare BodyContent object with the buffer. After executing setBodyContent(_) method container will access doInitBody() method in order to prepare BodyContent object for allow modifications.

Life Cycle of BodyTag interface:



- To prepare custom tags if we use above approach then the respective TagHandler class must implement all the methods declared in BodyTag interface irrespective of the application requirement.

This approach may increase burden to the developers and unnecessary methods in the custom tag application.

- To overcome this problem we will use an alternative provided by Jsp technology i.e. ***javax.servlet.jsp.tagext.BodyTagSupport*** class.

BodyTagSupport class is a concrete class, a direct implementation class to BodyTag interface and it has provided the default implementation for all the methods declared in BodyTag interface.

If we want to prepare custom tags with BodyTagSupport class then the respective TagHandler class must extend BodyTagSupport and overrides the only required methods.

```
public class BodyTagSupport implements BodyTag {
    public static final int EVAL_BODY_INCLUDE;
    public static final int SKIP_BODY;
    public static final int EVAL_PAGE;
    public static final int SKIP_PAGE;
    public static final int EVAL_BODY_AGAIN;
    public static final int EVAL_BODY_BUFFERED;
    public PageContext pageContext;
    public Tag t;
    public BodyContent bodyContent;
    public void setPageContext(PageContext pageContext) {
       this.pageContext=pageContext;
    }
    public void setParent(Tag t) {
       this.t=t;
    }
    public Tag getParent() {
       return t;
    }
    public int doStartTag()throws JspException {
       return EVAL_BODY_BUFFERED;
    }
    public void setBodyContent(BodyContent bodyContent) {
       this.bodyContent=bodyContent;
    }
    public void doInitBody()throws JspException
    {  }
    public int doAfterBody()throws JspException {
       return SKIP_BODY;
    }
    public int doEndTag()throws JspException {
       return EVAL_PAGE;      }
    public void release() {  }
}
```

- In case of body tags, custom tag body will be available in BodyContent object, to get custom tag body from BodyContent object we have to use the following method.

public String getString()

- To send modified data to the response object we have to get JspWriter object from BodyContent, for this we have to use the following method.

public JspWriter getEnclosingWriter()

Application:-

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:reverse>
        Sravya Infotech
</mytags:reverse>
```

hello.tld:-

```
<taglib>
    <jsp-version>2.1</jsp-version>
    <tlib-version>1.0</tlib-version>
    <tag>
        <name>reverse</name>
        <tag-class>com.sravya.Reverse</tag-class>
        <body-content>jsp</body-content>
    </tag>
</taglib>
```

Reverse.java:-

```
package com.sravya;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.tagext.BodyTagSupport;
publicclass Reverse extends BodyTagSupport{
    publicint doEndTag()throws JspException
    {   try
        {   String data=bodyContent.getString();
            StringBuffer sb=new StringBuffer(data);
            StringBuffer rdata=sb.reverse();
            bodyContent.getEnclosingWriter().println(rdata);
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
        returnEVAL_PAGE;
    }
}
```

----------Application11----------

custapp6:

editor.html:

```
<html>

<body bgcolor="lightgreen">
	<b><font size="6" color="red">
	<form action="./result.jsp">
	<pre>
		Enter SQL Query
		<textarea name="query" rows="5" cols="50"></textarea>
		<input type="submit" value="GetResult"/>
	</pre>
	</form></font></b></body>
</html>
```

result.jsp:

```
<%@taglib uri="/WEB-INF/result.tld" prefix="dbtags"%>
<jsp:declaration>
	String query;
</jsp:declaration>
<jsp:scriptlet>
	query=request.getParameter("query");
</jsp:scriptlet>
<dbtags:query>
	<jsp:expression>query</jsp:expression>
</dbtags:query>
```

result.tld:

```
<taglib>
	<jsp-version>2.1</jsp-version>
	<tlib-version>1.0</tlib-version>
	<tag>
		<name>query</name>
		<tag-class>com.dss.Result</tag-class>
		<body-content>jsp</body-content>
	</tag>
</taglib>
```

Result.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.sql.*;
public class Result extends BodyTagSupport
{
	Connection con;
	Statement st;
	ResultSet rs;
	public Result()
	{
		try
		{
			Class.forName("oracle.jdbc.driver.OracleDriver");

	con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","
ratan");
			st=con.createStatement();
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
	}
	public int doEndTag() throws JspException
	{
		try
		{
			JspWriter out=pageContext.getOut();
			String query=bodyContent.getString();
			boolean b=st.execute(query);
			if (b==true)
			{
				rs=st.getResultSet();
				ResultSetMetaData rsmd=rs.getMetaData();
				out.println("<html>");
				out.println("<body bgcolor='lightblue'>");
				out.println("<center><br><br>");
				out.println("<table border='1' bgcolor='lightyellow'>");
				int count=rsmd.getColumnCount();
				out.println("<tr>");
				for (int i=1;i<=count;i++)
				{
					out.println("<td><center><b><font size='6'
color='red'>"+rsmd.getColumnName(i)+"</font></b></center></td>");
				}
				out.println("</tr>");
				while (rs.next())
				{
					out.println("<tr>");
					for (int i=1;i<=count;i++)
```

```
                                        {

        out.println("<td><h1>"+rs.getString(i)+"</h1></td>");
                                        }
                                        out.println("</tr>");
                                }
                                out.println("</table></center></body></html>");
                        }
                        else
                        {
                                int rowCount=st.getUpdateCount();
                                out.println("<html>");
                                out.println("<body bgcolor='lightyellow'>");
                                out.println("<center><b><font size='7' color='red'>");
                                out.println("<br><br>");
                                out.println("Record Updated  :  "+rowCount);

                                out.println("</font></b></center></body></html>");
                        }
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
                return EVAL_PAGE;
        }
}
```

2. Simple Tags:

Classic tags Vs Simple tags:-

- Classic tags are more API independent, but Simple tags are less API independent.
- If we want to design custom tags by using classic tag library then we have to remember 3 types of life cycles.

  If we want to design custom tags by using simple tag library then we have to remember only 1 type of life cycle.
- In case of classic tag library, all the TagHandler class objects are cacheable objects, but in case of simple tag library, all the TagHandler class objects are non-cacheable objects.
- In case of classic tags, all the custom tags are not body tags by default, but in case of simple tags, all the custom tags are having body tags capacity by default.

If we want to design custom tags by using simple tag library then the respective TagHandler class must implement SimpleTag interface either directly or indirectly.

Predefind support:-

```
package javax.servlet.jsp.tagext;
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
public interface SimpleTag extends JspTag
{   public abstract void doTag()throws JspException, IOException;
    public abstract void setParent(JspTag jsptag);
    public abstract JspTag getParent();
    public abstract void setJspContext(JspContext jspcontext);
    public abstract void setJspBody(JspFragment jspfragment);
}
```

User defined class:-

```
public class MyHandler implements SimpleTag
    { //write the logics here
}
```

Where jspContext is an implicit object available in simple tag library, it is same as pageContext, it can be used to make available all the Jsp implicit objects.

Where jspFragment is like bodyContent in classic tag library, it will accommodate custom tag body directly.

Where setJspContext(_) method can be used to inject JspContext object into the present web application.

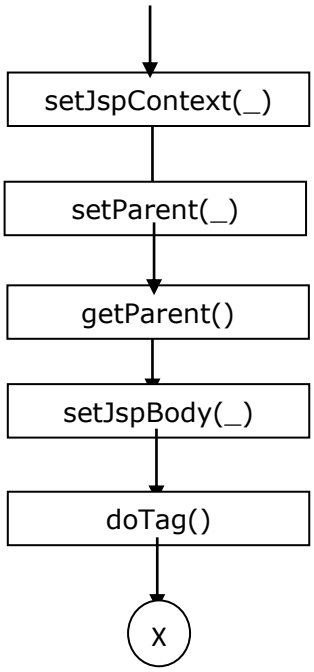
Where setParent(_) method can be used to inject parent tags TagHandler class object reference into the present TagHandler class.

Where getParent() method can be used to get parent tags TagHandler class object.

Where setJspBody(_) method is almost all equal to setBodyContent(_) method in order to accommodate custom tag body.

Where doTag() method is equivalent to doStartTag() method and doEndTag() method in order to perform an action.

Life Cycle of SimpleTag interface:



- To design custom tags if we use approach then the respective TgaHandler class must implement SimpleTag interface i.e. it must implement all the methods which are declared in SimpleTag interface.
- This approach will increase burden to the developers and unnecessary methods in the custom tags. To overcome the above problem Jsp technology has provided an alternative in the form of javax.servlet.jsp.tagext.SimpleTagSupport class.
- SimpleTagSupport is a concrete class provided by Jsp technology as an implementation class to SimpleTag interface with default implementation.
- If we want to design custom tags by using SimpleTagSupport class then the respective TagHandler class must be extended from SimpleTagSupport class and where we have to override the required methods.

Predefined support:-

```
public interface SimpleTag extends JspTag {
    private JspContext jspContext;
    private JspFragment jspFragment;
    private JspTag jspTag;
    public void setJspContext(JspContext jspContext) {
       this.jspContext=jspContext;
    }
    public void setParent(JspTag t) {
       this.jspTag=jspTag;
    }
    public void setJspBody(JspFragment jspFragment) {
       this.jspFragment=jspFragment;
    }
    public JspTag getParent() {
       return jspTag;
    }
```

```
    public JspFragment getJspBody() {
       return jspFragment;
    }
    public JspContext getJspContext() {
       return jspContext;
    }
    public void doTag()throws JspException, IOException
    {  }
  }
```

User defined class:-

```
public class MyHandler extends SimpleTagSupport
{       //write the logics here
 }
```

Application:

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello/>
```

hello.tld:-

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>hello</name>
                <tag-class>com.sravya.Hello</tag-class>
                <body-content>empty</body-content>
        </tag>
</taglib>
```

Hello.java:-

```
package com.sravya;
import java.io.IOException;
import javax.servlet.jsp.JspContext;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
publicclass Hello extends SimpleTagSupport{
        @Override
        publicvoid doTag() throws JspException, IOException {
                JspContext context =  getJspContext();
                JspWriter writer =  context.getOut();
                writer.println("this is Sample Tag application");
        }
}
```

Accessing Tag Body:-

To print custom tag body use below code.

Application:-

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello>
	hi this is body of custom tag<br>
</mytags:hello>
```

hello.tld:-

```
<taglib>
	<jsp-version>2.1</jsp-version>
	<tlib-version>1.0</tlib-version>
	<tag>
		<name>hello</name>
		<tag-class>com.sravya.Hello</tag-class>
		<body-content>scriptless</body-content>
	</tag>
</taglib>
```

Hello.java:-

```
package com.sravya;
import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
publicclass Hello extends SimpleTagSupport{
	@Override
	publicvoid doTag() throws JspException, IOException {
		JspWriter writer =  getJspContext().getOut();
		StringWriter stringWriter = new StringWriter();
		getJspBody().invoke(stringWriter);
		writer.println("this is Sample Tag application");
		writer.println(stringWriter.toString());
	}
}
```

Attributes in custom tags:-

To provide the attributes in custom tags use below code.

Application:-

hello.jsp:-

```
<%@taglib uri="/WEB-INF/hello.tld" prefix="mytags"%>
<mytags:hello msg="ratansoft">
	hi this is body of custom tag<br>
</mytags:hello>
```

hello.tld:-

```
<taglib>
	<jsp-version>2.1</jsp-version>
	<tlib-version>1.0</tlib-version>
	<tag>
		<name>hello</name>
		<tag-class>com.sravya.Hello</tag-class>
		<body-content>scriptless</body-content>
		<attribute>
			<name>msg</name>
	</attribute>
	</tag>
</taglib>
```

Hello.java:-

```
package com.sravya;

import java.io.IOException;
import java.io.StringWriter;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.SimpleTagSupport;
publicclass Hello extends SimpleTagSupport{
	private String msg;
	publicvoid setMsg(String msg) {
		this.msg = msg;
	}
	@Override
	publicvoid doTag() throws JspException, IOException {
		JspWriter writer =  getJspContext().getOut();

		StringWriter stringWriter = new StringWriter();
		getJspBody().invoke(stringWriter);
		writer.println("this is Sample Tag application");
		writer.println(stringWriter.toString());
		writer.println("<br>");
		writer.println(msg);
	}
}
```

Custom URI in jsp custom tags:-

- *We can use custom uri in jsp custom tags by specifying **tld** file location in web.xml file.*
- *The web container will get the information about tld file from web.xml file.*

Application:-

hello.jsp:-

```
<%@taglib uri="mytags" prefix="xxx"%>
<xxx:hello>
        Hi Ratan how r u <br>
</xxx:hello>
this is rest of the JSP Application
```

web.xml:-

```
<web-app>
        <jsp-config>
                <taglib>
                <taglib-uri>mytags</taglib-uri>
                <taglib-location>/WEB-INF/hello.tld</taglib-location>
                </taglib>
        </jsp-config>
</web-app>
```

hello.tld:-

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <uri>mytags</uri>
        <tag>
                <name>hello</name>
                <tag-class>com.sravya.TaglibUri</tag-class>
                <body-content>jsp</body-content>
        </tag>
</taglib>
```

TaglibUri.java:-

```
package com.sravya;
import java.io.IOException;
import java.util.Date;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.PageContext;
import javax.servlet.jsp.tagext.TagSupport;
publicclass TaglibUri extends TagSupport{
        @Override
        publicint doStartTag() throws JspException {
                JspWriter writer =   pageContext.getOut();
                try {
                        Date d = new Date();
                        writer.println("this is custom tag application URI attribute");
                        writer.println("<br/>");
                        writer.println("today date is="+d);
```

```
                    writer.println("<br/>");
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return EVAL_BODY_INCLUDE;
        }
        @Override
        publicint doEndTag() throws JspException {
                return EVAL_PAGE;
        }
}
```

Jsp Standard Tag Library(JSTL):

In Jsp technology, by using scripting elements we are able to provide Java code inside the Jsp pages.

To preserve Jsp principles we have to eliminate scripting elements, for this we have to use Jsp Actions.

In case of Jsp Actions, we will use standard actions as an alternative to scripting elements, but which are limited in number and having bounded functionality so that standard actions are not specified the required application format.

Still it is required to provide Java code inside the Jsp pages.

In the above context, to eliminate Java code completely from Jsp pages we have to use custom actions.

In case of custom actions, to implement simple Java syntaxes like if condition, for loop and so on we have to provide a lot of Java code internally.

To overcome the above problem Jsp technology has provided a separate tag library with simple Java syntaxes implementation and frequently used operations.

JSTL is an abstraction provided by Sun Microsystems, but where implementations are provided by all the server vendors.

With the above convention Apache Tomcat has provided JSTL implementation in the form of the jar files as standard.jar, jstl.jar.

Apache Tomcat has provided the above 2 jar files in the following location.

C:\Tomcat7.0\webapps\examples\WEB_INF\lib

If we want to get JSTL support in our Jsp pages then we have to keep the above 2 jar files in our web application lib folder.

JSTL has provided the complete tag library in the form of the following 5 types of tags.

1. Core Tags
2. XML Tags
3. Internationalization or I18N Tags (Formatted tags)
4. SQL Tags
5. Functions tags

To get a particular tag library support into the present Jsp page we have to use the following standard URL's to the attribute in the respective taglib directive.

http://java.sun.com/jstl/core

http://java.sun.com/jstl/xml

http://java.sun.com/jstl/fmt

http://java.sun.com/jstl/sql

http://java.sun.com/jsp/jstl/functions

1. Core Tags:

In JSTL, core tag library was divided into the following 4 types.

1. General Purpose Tags
    1. <c:set----->
    2. <c:remove----->
    3. <c:catch----->
    4. <c:out----->
2. Conditional Tags
    1. <c:if----->
    2. <c:choose----->
    3. <c:when----->
    4. <c:otherwise----->
3. Iterate Tags
    1. <c:forEach----->
    2. <c:forTokens----->
4. Url-Based Tags
    1. <c:import----->
    2. <c:url----->
    3. <c:redirect----->

1. General Purpose Tags:

1. <c:set----->:

This tag can be used to declare a single name value pair onto the specified scope.

Syntax:<c:set var="--" value="--" scope="--"/>

Where var attribute will take a variable i.e. key in key-value pair.

Where value attribute will take a particular value i.e. a value in key-value pair.

Where scope attribute will take a particular scope to include the specified key-value pair.

2. <c:out----->:

This tag can be used to display a particular value on the client browser.

Syntax:<c:out value="--"/>

Where value attribute will take a static data or an expression.

To present an expression with value attribute we have to use the following format.

Syntax:${expression}

Ex:<c:out value"${a}"/>

If the container encounters above tag then container will search for 'a' attribute in the page scope, request scope, session scope and application scope.

----------Application13----------

jstlapp1:
core.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
		<c:set var="a" value="AAA" scope="request"/>
		<br>
		<c:out value="core tag library"/>
		<br><br>
		<c:out value="${a}"/>
		</font></b></center>
	<body>
<html>
```

3. <c:remove----->:

This tag can be used to remove an attribute from the specified scope.

Syntax:<c:remove var=”--” scope=”--”/>

Where scope attribute is optional, if we have not specified scope attribute then container will search for the respective attribute in the page scope, request scope, session scope and application scope.

----------Application14----------

core1.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
		<c:set var="a" value="AAA" scope="request"/>
		<br>
		a-----><c:out value="${a}"/>
		<br><br>
		<c:remove var="a" scope="request"/>
		a-----><c:out value="${a}"/>
		</font></b></center>
	<body>
<html>
```

4. <c:catch----->:

This tag can be used to catch an Exception raised in its body.

Syntax:<c:catch var="--">

---------- </c:catch>

Where var attribute will take a variable to hold the generated Exception object reference.

----------Application15----------

core2.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
		<c:catch var="e">
			<jsp:scriptlet>
				java.util.Date d=null;
				out.println(d.toString());
			</jsp:scriptlet>
		</c:catch>
		<c:out value="${e}"/>
		</font></b></center>
	<body>
<html>
```

2. Conditional Tags:

1. <c:if----->: This tag can be used to implement if conditional statement.

Syntax:<c:if test="--"/> Where test attribute is a boolean attribute, it may take either true or false values.**----------Application16----------**

core3.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<c:set var="a" value="10"/>
			<c:set var="b" value="20"/>
			<c:if test="${a<b}">
				condition is true
			</c:if>
			<br>
			out of if
		</font></b></center>
	<body>
<html>
```

2. <c:choose----->, <c:when-----> and <c:otherwise----->:

These tags can be used to implement switch programming construct.

Syntax:
```
<c:choose>
<c:when test="--">
            ----------
</c:when>
            ----------
<c:otherwise>
            ----------
</c:otherwise>
</c:choose>
```

----------Application17----------

core4.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<c:set var="a" value="10"/>
			<c:choose>
				<c:when test="${a==10}">
					TEN
				</c:when>
				<c:when test="${a==15}">
					FIFTEEN
				</c:when>
				<c:when test="${a==20}">
					TWENTY
				</c:when>
				<c:otherwise>
					Number is not in 10,15 and 20
				</c:otherwise>
			</c:choose>
		</font></b></center>
	<body>
<html>
```

3. Iterator Tags:

1. <c:forEach----->:

This tag can be used to implement for loop to provide iterations on its body and it can be used to perform iterations over an array of elements or Collection of elements.

Syntax 1:<c:forEach var="--" begin="--" end="--" step="--">

</c:forEach>

Where var attribute will take a variable to hold up the loop index value at each and every iteration.

Where begin and end attribute will take start index value and end index value.

Syntax 2:<c:forEach var="--" items="--">

</c:forEach>

Where var attribute will take a variable to hold up an element from the respective Collection at each and every iteration.

Where items attribute will take the reference of an array or Collection from either of the scopes page, request, session and application.

----------Application18----------

core5.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<c:forEach var="a" begin="0" end="10" step="2">
				<c:out value="${a}"/><br>
			</c:forEach>
			<%
				String[] s={"A","B","C"};
				request.setAttribute("s",s);
			%>
			<br>
			<c:forEach var="x" items="${s}">
				<c:out value="${x}"/><br>
			</c:forEach>
		</font></b></center>
	<body>
<html>
```

2. <c:forTokens----->:

This tag can be used to perform String Tokenization on a particular String.

Syntax :<c:forTokens var=”--” items=”--” delims=”--”>

</c:forTokens>

Where var attribute will take a variable to hold up token at each and every iteration.

Where items attribute will take a String to tokenize.

Where delims attribute will take a delimiter to perform tokenization.

----------Application19----------

core6.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<c:forTokens var="token" items="Ratan Software Solutions" delims="
">
				<c:out value="${token}"/><br>
			</c:forTokens>
		</font></b></center>
	<body>
<html>
```

4. Url-Based Tags:

1. <c:import----->:

This tag can be used to import the content of the specified target resource into the present Jsp page.

----------Application20----------

second.jsp:

```
<center><h1>This is second.jsp</h1></center>
```

core7.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			Start
			<br>
			<c:import url="second.jsp"/>
			<br>
			End
		</font></b></center>
	<body>
<html>
```

2. <c:url----->:

This tag can be used to represent the specified url.

Syntax :<c:url value="--"/>

----------Application21----------

core8.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<a href='<c:url value="http://localhost:2020/loginapp"/>'>Login
Application</a>
		</font></b></center>
	<body>
<html>
```

2. <c:redirect----->:

This tag can be used to implement Send Redirect Mechanism from a particular Jsp page.

Syntax:<c:redirect url="--"/>

----------Application22----------

core9.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@page isELIgnored="true"%>

<html>
	<body>
		<center><b><font size="7">
			<c:redirect url="http://localhost:2020/registrationapp"/>
		</font></b></center>
	<body>
<html>
```

4. SQL Tags:

The main purpose of SQL tag library is to interact with the database in order to perform the basic database operations.

JSTL has provided the following set of tags as part of SQL tag library.

1. <sql:setDataSource----->
2. <sql:update----->
3. <sql:query----->
4. <sql:transaction----->
5. <sql:param----->
6. <sql:dateParam----->

1. <sql:setDataSource----->:

This tag can be used to prepare the Jdbc environment like Driver loading, establish the connection.

Syntax:<sql:setDataSource driver="--" url="--" user="--" password="--"/>

Where driver attribute will take the respective Driver class name.

Where url attribute will take the respective Driver url.

Where user and password attributes will take database user name and password.

2. <sql:update----->:

This tag can be used to execute all the updation group SQL queries like create, insert, update, delete, drop and alter.

Syntax 1:<sql:update var="--" sql="--"/>

Syntax 2:<sql:update var="--"> ------ query ------ </sql:update>

In the above <sql:update> tag we are able to provide the SQL queries in 2 ways.

1. Statement style SQL queries, which should not have positional parameters.
2. PreparedStatement style SQL queries, which should have positional parameters.

If we use PreparedStatement style SQL queries then we have to provide values to the positional parameters, for this we have to use the following SQL tags.

1. <sql:param----->:

This tag can be used to set a normal value to the positional parameter.

Syntax 1:<sql:param value="--"/>

Syntax 2:<sql:param> ------ value ------ </sql:param>

2. <sql:dateParam----->:

This tag can be used to set a value to parameter representing data.

Syntax 1:<sql:dateParam value="--"/>

Syntax 2:<sql:dateParam>value</sql:dateParam>

----------Application23----------

jstlapp2:

sql.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
			<sql:update var="result" sql="create table emp1(eid number, ename
varchar2(10), esal number)"/>
			Row Count ..... <c:out value="${result}"/>
		</font></b></center>
	</body>
</html>
```

----------Application24----------

sql1.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
			<sql:update var="result" sql="insert into emp1
values(101,'aaa',5000)"/>

			Row Count ..... <c:out value="${result}"/>
		</font></b></center>
	</body>
</html>
```

----------Application25----------

sql2.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
			<sql:update var="result" sql="insert into emp1 values(?,?,?)">
				<sql:param value="103"/>
				<sql:param>ccc</sql:param>
				<sql:param value="7000"/>
			</sql:update>
			Row Count ..... <c:out value="${result}"/>
		</font></b></center>
	</body>
</html>
```

----------Application26----------

sql3.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
			<sql:update var="result">
				update emp1 set esal=esal+? where esal>?
				<sql:param>500</sql:param>
				<sql:param>5000</sql:param>
			</sql:update>
			Row Count ......  <c:out value="${result}"/>


</font></b></center></body>
</html>
```

----------Application27----------

sql4.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
			<sql:update var="result">
				delete emp where esal>1000
			</sql:update>
			Row Count ......  <c:out value="${result}"/>
		</font></b></center>
	</body>
</html>
```

3. <sql:query----->:

This tag can be used to execute selection group SQL queries in order to fetch the data from database table.

Syntax 1:<sql:query var="--" sql="--"/>

Syntax 2:<sql:query var="--"> ----- query ----- </sql:query>

If we execute selection group SQL queries by using <sql:query> tag then SQL tag library will prepare result object to hold up fetched data.

In SQL tag library, result object is a combination of ResultSet object and ResultSetMetaData object.

In result object, all the column names will be represented in the form a single dimensional array referred by columnNames predefined variable and column data (table body) will be represented in the form of 2-dimensionnal array referred by rowsByIndex predefined variable.

sql5.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jstl/sql" prefix="sql"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<sql:setDataSource driver="oracle.jdbc.driver.OracleDriver"
url="jdbc:oracle:thin:@localhost:1521:xe" user="system" password="ratan"/>
			<sql:query var="result" sql="select * from emp"/>
			<table border="1" bgcolor="lightyellow">
			<tr>
				<c:forEach var="columnName"
items="${result.columnNames}">
					<td><center><b><font size="6" color="red">
					<c:out value="${columnName}"/>
					</font></b></center></td>
				</c:forEach>
			</tr>
			<c:forEach var="row" items="${result.rowsByIndex}">
				<tr>
					<c:forEach var="column" items="${row}">
						<td><b><font size="5">
							<c:out value="${column}"/>
						</font></b></td>
					</c:forEach>
				</tr>
			</c:forEach>
			</table>
		</font></b></center>
	</body>
</html>
```

4. <sql:transaction----->:

This tag will represent a transaction, which includes collection of <sql:update> tags and <sql:query> tags.

3. I18N Tags(Formatted Tags):

1. <fmt:setLocale----->:

This tag can be used to represent a particular Locale.

Syntax:<fmt:setLocale value="--"/>

Where value attribute will take Locale parameters like en_US, it_IT and so on.

2. <fmt:formatNumber----->:

This tag can be used to represent a number w.r.t the specified Locale.

Syntax:<fmt:formatNumber var="--" value="--"/>

Where var attribute will take a variable to hold up the formatted number.

Where value attribute will take a number.

3. <fmt:formatDate----->:

This tag can be used to format present system date w.r.t. a particular Locale.

Syntax:<fmt:formatDate var="--" value="--"/>

Where var attribute will take a variable to hold up the formatted date.

Where value attribute will take the reference of Date object.

4. <fmt:setBundle----->:

This tag can be used to prepare ResourceBundle object on the basis of a particular properties file.

Syntax:<fmt:setBundle var="--" basename="--"/>

Where var attribute will take a variable to hold up ResourceBundle object reference.

Where basename attribute will take base name of the properties file.

5. <fmt:message----->:

This tag can be used to get the message from ResourceBundle object on the basis of provided key.

Syntax:<fmt:message var="--" key="--"/>

Where var attribute will take a variable to hold up message.

Where key attribute will take key of the message defined in the respective properties file.

----------Application29----------

jstlapp3:

abc_en_US.properties:

```
#abc_en_US.properties
#------------------------------
welcome=Welcome to US user
```

abc_it_IT.properties:

```
#abc_it_IT.properties
#------------------------------
welcome=Welcome toe Italiano usereo
```

fmt.jsp:

```
<%@taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

<%@taglib uri="http://java.sun.com/jstl/fmt" prefix="fmt"%>
<%@page isELIgnored="true"%>
<html>
	<body>
		<center><b><font size="7">
			<fmt:setLocale value="it_IT"/>
			<fmt:formatNumber var="num" value="123456.789"/>
			<c:out value="${num}"/><br><br>
			<jsp:useBean id="date" class="java.util.Date">
				<fmt:formatDate var="fdate" value="${date}"/>
				<c:out value="${fdate}"/>
			</jsp:useBean>
			<br><br>
			<fmt:setBundle basename="abc"/>
			<fmt:message var="msg" key="welcome"/>
			<c:out value="${msg}"/>
		</font></b></center>
	</body>
</html>
```

JSTL Functions:

The main purpose of functions tag library is to perform all the String operations which are defined in the String class.

Ex:<%taglib uri="http://java.sun.com/jstl/core" prefix="c"%>

<%taglib uri="http://java.sun.com/jstl//jspfunctions" prefix="fn"%>

<c:set var="a" value="Ratan Software Solutions"/>

${fn.length(a)}

${fn.concat(a, "Hyderabad")}

${fn.toLowerCase(a)}

${fn.toUpperCase (a)}

${fn.contains(a, "Software")}

${fn.startsWith(a, "Ratan")}

${fn.endsWith(a, "Solutions")}

${fn.substring(a, 4, 20)}