ADVANCE JAVA

COURSE MATERIAL

BY

MR.NAGOOR BABU

DURGA SOFT

sri raghavendra Xerox

All software language materials available

beside banglore iyyengars bakery opp:cdac balkampetroad ameerpet Hyderabad

cell :9951596199

Importance of ADVANCED JAVA: DURGASOFT

- Every Software Company will do projects on J2EE Technologies
- If you attend any JAVA based Interview, 40% of the Questions are coming from ADVANCED JAVA only.
- If you are Strong in ADVANCED JAVA then it is very easy to learn all the futer technologies and frameworks like STRUTS, JSF, SPRING,.....
- If you are Strong in ADVANCED JAVA surveying in software companies is very easy.
- ADVANCED JAVA is basis for all the enterprise applications.

Why NAGOORBABU sir ADVANCE JAVA:

- Covered Advance java topics in more depth.
- Provide live execution for each and every application in the class itself.
- Very good notes dictation for each and every topic which contains 500+ pages.
- Certification Oriented training on each and every topic (OCWCD).
- Conducting classes Even in Sundays for the benefit of Students i.e Covering each and every topic from scratch and also to complete course with in the time.
- Reviewing previous day topic before starting the class.
- Conducting interview questions sessions at the end of each and every topic.
- Covering interview questions and answers in detail in the notes dictation.
- Providing 2 mini projects on Advance JAVA.

Java Database Connectivity:

1.Storage Areas

1)Temporary Storage Areas

2)Permanent Storage Areas

2.QueryProccessing System

1)Query Tokenization

2)Query Processing

3)Query Optimization

4)Query Execution

3.Driver and Driver Types

1)Type 1 Driver

2)Type 2 Driver

3)Type 3 Driver

4)Type 4 Driver

4.Steps To design Jdbc Applications

1)Load and register the Driver.

2)Establish the connection between Java Application.

3)Prepare either Statement or preparedStatement or CallableStatement Objects.

4)Write and execute SQL Queries.

5)close the connection.

6.ResultSet and ResultSet Types

1)Read only ResultSet

2)UpdatableResultSet

3)Forward only ResultSet

4)ScrollableResultSets

1)Scroll Sensitive ResultSet

2)Scroll Insensitive ResultSet

7.Prepared Statement

1)PreparedStatement with insert sql query

2)PreparedStatement with update sql query

3)PreparedStatement with select sql query

8.Callable Statement

1)CallableStatement with procedure

2)CallableStatement with function

3)CallableStatement with CURSOR Type Procedure

4)CallableStatement with CURSOR type function

9.Transaction Management

1. Atomicity
2. Consistency
3. Isolation
4. Durability

10.Savepoint

11.BatchUpdations

12.Connection Pooling

13.BLOB and CLOB

Servlets:

1.Introduction

1) Standalone Applications

2) Enterprise Applications

2.Client-Server Arch

1)Client

2)Server

3)Protocol

3.Servlets Design

4.Servlets Lifecycle

5.User Interface

1.Static Form Generation

2.Dynamic Form Generation

6.ServletConfig

7.Servlet Context

8.Servlet Communication

1. Browser-servlet
2. Web-component
3. Applet-Servlet

9.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

10.Servlets Filters

11.Servlets Wrappers

12.Servlets Listeners

Java Server Pages:

1.Introduction

2.JSP Deployment

3.JSPLifeCycle

4.Jsp Elements

1.Jsp Directives

2.Scripting Elements

3.Jsp Actions

5.JSP Directives

1. Page Directive
2. Include Directive
3. Taglib Directive

6.JSP Scripting Elements

1. Declarations
2. Scriptlets
3. Expressions

7.JSP implicit objects

1.out
2. request
3. response
4. config
5. application
6. session
7. exception
8. page
9. pageContext

8.JSP Scopes

1.Page Scope

2.request Scope

3.Application Scope

4.SessionScoper

9.JSP Standard Actions

10.JSP Custom Actions

11.JSTL

1. Core Tags
2. XML Tags
3. Internationalization or I18N Tags (Formatted tags)
4. SQL Tags
5. Functions tags

12.Expression Language

1)EL operators

2)EL implicit objects.

3)EL functions.

Storage Areas

As part of the Enterprise Application development it is essential to manage the organizations data like Employee Details, CustomerDetails, Products Details..etc

-->To manage the above specified data in enterprise applications we have to use storage areas (Memoryelements).There are two types of Storage areas.

1) Temporary Storage Areas:

These are the memory elements, which will store the data temporarily.
Eg:Buffers,Java Objects

2) Permanent Storage Objects:

These are the memory elements which will store data permanently.
Eg:FileSystems,DBMS,DataWareHouses.

File Systems:
It is a System, it will be provided by the local operating System.

--->Due to the above reason File Systems are not suitable for platform independent technologies like JAVA.
--->File Systems are able to store less volumes of the data.
--->File Systems are able to provide less security.
--->File Systems may increases data Redundancy.
--->In case of File Systems Query Language support is not available. So that all the database operations are complex.

DBMS:

-->Database Management System is very good compare to file System but still it able to store less data when compared to DataWareHouses.
-->DBMS is very good at the time of storing the data but which is not having Fast Retrieval Mechanisms.

DURGASOFT

DataWareHouses:

When Compared to File Systems and DBMS it is able to store large and large volumes of data.

-->Data ware houses having fast retrieval mechanisms in the form of data mining techniques.

Q)What is the difference between database and database management system?
Ans:
DataBase is a memory element to store the data.
Database Management System is a Software System,it can be used to manage the data by storing it on database and retrieving it form Database.
Database is a collection of interrelated data as a single unit.
DBMS is a collection of interrelated data and a set of programs to access the data.

There are three types of DBMS:

1)RDMS(Relational Database Management Systems)
2)OODBMS(Object Oriented DataBase Management Systems)
3)ORDBMS(Object Relational DataBase Management Systems)

1)Relational Database Management Systems:

-->It is a DBMS,it can be used to represent the data in the form of tables.
-->This DBMS will use SQL3 as a Query Language to perform DataBase Operations.

2)Object Oriented DataBase Management System:

-->It is DataBase Management System,it will represents the data in the form of Objects.
-->This database management system will require OQL(Object Query Language)as Query language to perform database operations.

3)Object Relational DataBase Management System:

-->It is a DataBaseManagement System,it will represents some part of data in the form of Tables and some other part of the data in the form of objects
-->This DataBaseManagement System will require SQl3 as Query Language to perform database operations.

Where SQL3 is the combination of SQL2 and OQL

DURGASOFT

SQL3=SQL2+OQL

Query Processing System:

When we submit an SQL Query to the Database then Database Engine will Perform the following Steps.

Step1:

Query Tokenization:

This Phase will take SQL Query as an Input,divided into no.of tokens andGenerate Stream of tokens as an output.

Step2:

Query Processing:

This phase will take Stream of tokens as an Input,constructs Query Tree with the Tokens,if Query Tree Success then no Syntax error is available in the provided SQL Query.If Query Tree is not Success then there are some syntax errors in the provided SQL Query.

Step3:

Query Optimization:

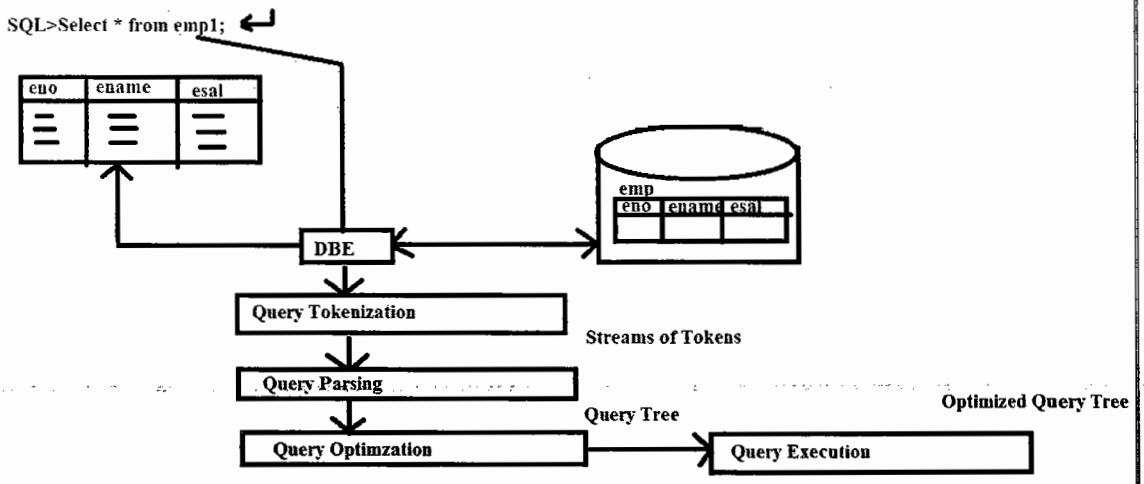
The main purpose of Query Optimization phase is to perform optimization on Query Tree in order to reduce execution time and to optimize memory utilization.

Step4:

Query Execution:

This phase will take optimized Query Tree as an input and execute the Query by using interpreters.

DURGASOFT



JDBC(Java DataBase Connectivity):

-->The process of interacting with the database from Java Applications is called as JDBC.

-->JDBC is an API,which will provide very good predefined library to connect with database from JAVAApplications in order to perform the basic database operations:

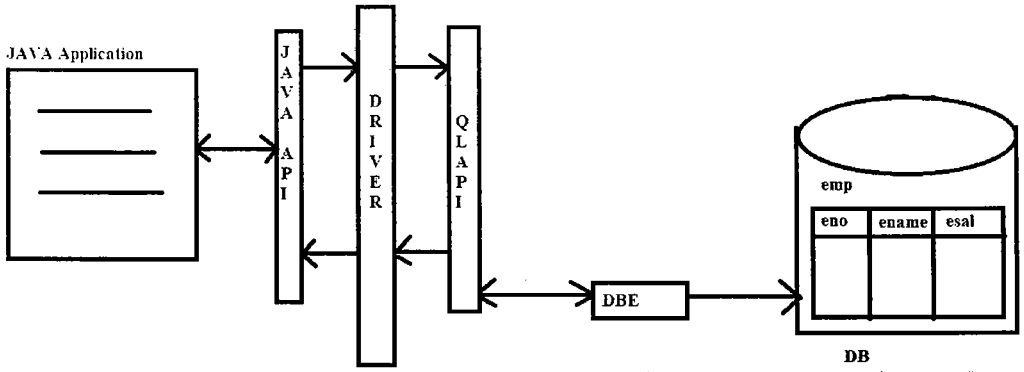
-->In case of JDBC Applications we will define the database logic and Java application and we will send a Java represented database logic to Database Engine.But database engine is unable to execute the Java represented database logic,it should required the database logic in Query Language Representations.

-->In the above context, to execute JDBC applications we should require a conversion mechanism to convert the database logic from Java representations to Query language representations and fromQuery language representations to Java representations.

-->In the above situation the required conversion mechanisms are available in the form of a software called as "Driver".

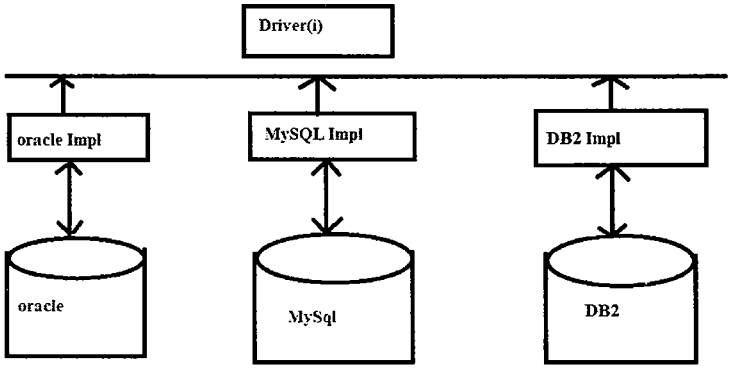
Driver:

-->Driver is an interface existed between Java application and database to map Java API calls to Query language API calls and Query language API calls to Java API calls.



-->To provide driver as a product Sun MicroSystems has provided Driver as an interface and Sun MicroSystems.lets the database vendors to provide implementation classes to the driver interface as part of their database software's.

-->If we want to use Drivers in JDBC applications then we have to get Driver implementation from the respective database software's.



-->There are 180+ number of drivers but all these drivers could be classified into the following four types
1)Type 1
2)Type 2
3)Type 3
4)Type 4

1)Type 1 Driver:

-->Type 1 Driver is also called as JDBC-ODBC Driver and Bridge Driver.

-->JDBC-ODBC Driver is a driver provided by Sun Micro Systems as an Implementation to Driver Interface.

-->Sun MicroSystems has provided JDBC-ODBC Driver with the inter dependent on the Microsoft's productODBC Driver.

-->ODBC Driver is a Open Specification,it will provide very good environment to interact with any type of database from JDBC-ODBC Driver.

-->If we want to use JDBC-ODBC Driver in our JDBC Applications first we have to install the MicroSoft Product ODBC Driver native library.

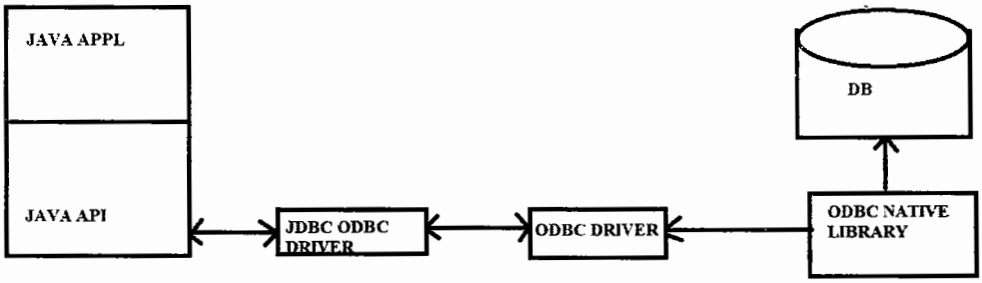
-->To interact with the database from Java Application if we use JDBC-ODBC Driver then we shouldrequire two types conversions so that JDBC-ODBC Driver is Slower Driver.

-->JDBC-ODBC Driver is highly recommended for stand alone applications,it is not suitable for web applications,distributed applications and so on.

-->JDBC-ODBC Driver is suggestable for Simple JDBC applications,not for complex JDBC applications.

-->The portability of the JDBC-ODBC Driver is very less.

DURGASOFT



2)Type 2 Driver:

-->Type 2 Driver is also called part java,part native driver that is Type 2 Driver was implemented by using Java implementations and the database vendor provided native library.

-->When compared to Type1 Driver Type2 Driver is faster Driver because it should not require two times conversions to interact with the Database from Java Applications.

-->When compared to Type1 Driver Type2 driver portability is more.

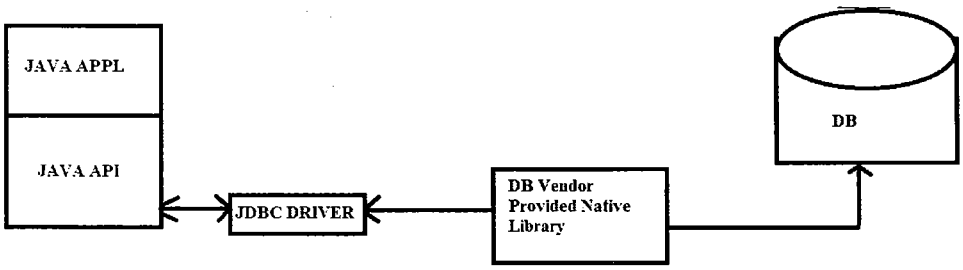
-->Type2 Driver is still recommended for standalone application not suggestible for web applications and Enterprise applications.

-->If we want to use Type2 Driver in our Jdbc applications then we have to install the database vendor provided native library.

-->Type2 Driver is cast full Driver among all the drivers.

-->Type2 Driver's portability is not good when compared to Type3 Driver and Type4 Driver.

SOFT



3)Type 3 Driver:

-->Type 3 Driver is also called as MiddleWare DataBase Server Access Driver and NetWorkDriver.

-->Type 3 Driver is purely designed for Enterprise applications it is not suggestible for stand alone applications.

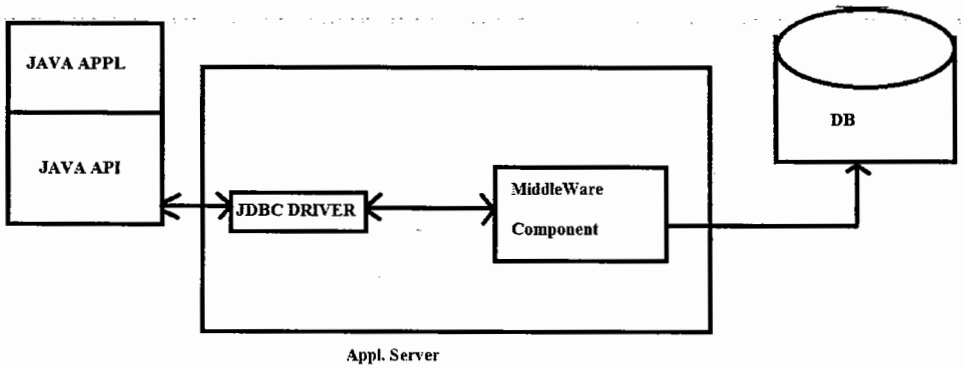
-->Type 3 Driver portability is very good when compared to Type1 and Type2 Driver's.

-->Type 3 Driver will provide very good environment to interact with multiple no.of databases.

-->Type 3 Driver will provide very good environment to switch from one database to another database without having modifications in client applications.

-->Type 3 Driver should not require any native library installations, it should require the Compatibility with the application server.

-->Type 3 Driver is fastest Driver when compared to all the Drivers.



4)Type 4 Driver:

-->Type 4 Driver is also called as pure Java Driver and Thin Driver because Type 4 Driver was implemented completely by using java implementations.

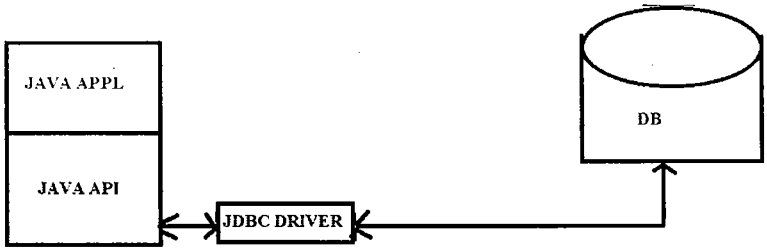
-->Type 4 Driver is the frequent used Driver when compared to all the remaining Drivers.

-->Type 4 Driver is recommended for any type application includes standalone applications, Network Applications....

-->Type 4 Driver portability is very good when compared to all the remaining Drivers.

-->Type 4 driver should not require any native library dependences and it should require one time conversion to interact with database from Java Applications.

-->Type 4 is the cheapest Driver among all.



Steps to design JDBC Application:

1)Load and register the Driver.

2)Establish the connection between Java Application.

3)Prepare either Statement or preparedStatement or CallableStatement Objects.

4)Write and execute SQL Queries.

5)close the connection.

1)Load and Register the Driver:

In general Driver is an interface provided by Sun Microsystems and whose implementation classes are provided by the Database Vendors as part of their Database Softwares.

-->To load and Register the Driver first we have to make available Driver implementation to JDBC application.For this we have to set classpath environment variable to the location Where we have Driver implementation class.

DURGASOFT

-->If we want to use Type1 Driver provided by Sun MicroSystems in our JDBC applications then it is not required to set classpath environment variable because Type1 Driver was provided by Sun MicroSystems as part of Java Software in the form of**sun.jdbc.odbc.JdbcOdbcdriver**

-->If we want to use Type1 Driver in our JDBC applications then before loading we have to Configure the **Microsoft product odbc Driver.**

-->To configure Microsoft product odbc Driver we have to use the following path.

To setting DSN:-

Start
↓
Control Panel
↓
System and Security
↓
Administrative Tools
↓
Data Sources (ODBC)
↓
user DSN
↓
Click on Add button
↓
Select Microsoft ODBC for the Oracle
↓
Click on Finish button
↓
Provide DSN name (provide any name)
↓
Click on OK

-->To load and register Driver in our Jdbc applications we have to use the following method from class 'Class'

Public static Class forName(String class_Name)

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

-->When JVM encounter the above instruction JVM will pickup the parameter that is **JDBCOdbcDriver** Class name and JVM will search for its .class file in the current location,if it is not available then JVM will search for it in Java predefined library.

DURGASOFT

-->If JVM identify JDBCODBCDriver.class file in Java pre-defined library(rt.jar) then JVM will load **JdbcOdbcDriver** class byte code to the memory.

-->At the time of loading JdbcOdbcDriver class byte code to the memory JVM will execute a static block, As part of this JVM wiil execute a method call like **DriverManager.registerDriver(--);** by the execution of registerDriver() method only JDBCOdbcDrvier will be available to our Jdbc applications.

-->In case of Type1 Driver if we use either Jdbc 4.0 version or Jdk 6.0 version then it is optional To perform loading and register the driver step because JVM will perform Drvier registration automatically at the time of establishing the connection between Java application and Database.

NOTE:To prepare Jdbc applications Java API has provided the required pre-defined library in the form of java.sql package so that we have to import this package in our Java file.

Import java.sql.*;

-->java.sql package includes the following pre-defined library to design Jdbc applications.

java.sql package includes the following predefined library:-
I----->interface C------->class

1. Driver (I)
2. DriverManager (C)
3. Connection (I)
4. Statement (I)
5. PreparedStatement (I)
6. ResultSet (I)
7. ResultSetMetaData (I)
8. DatabaseMetaData (I)
9. Savepoint(i)

DURGASOFT

2)Establish the Connection between Java application and Database:

-->To establish the connection between Java application and Database we have to use the following Method from **DriverManager class.**

Public static Connection getConnection(String url,String db_user_name,String db_password)

Ex:Connection con=DriverManager.getConnection("jdbc:odbc:dsnName","system","durga");

-->When JVM encounter the above instruction JVM will access getConnection method,as part of the getConnection method JVM will access connect() method to establish virtual socket connection Between Java application and database as per the url which we provided.

-->wher getConnection() method will take three parameters
1.Driver URL
2.Database username
3.Database password
-->In general from Driver to Driver Driver class name and Driver url will varied.

-->If we use Type1 Driver then we have to use the following Driver class name and URL

d-class : sun.jdbc.odbc.JdbcOdbcDriver
url :jdbc:odbc:dsnName

-->In general all the Jdbc Drivers should have an url with the following format.

main-protocol: sub-protocol

-->where main-protocol name should be Jdbc for each and every Driver but the sub protocol name should be varied from Driver to Driver.

Q) In Jdbc applications getConnecction() method will establish the connection between Java application and Database and return connection object but connection is an interface how it is possible to Create connection object?

Ans:In general in Java technology we are unable to create objects for the interfaces directly,if we want to accommodate interface data in the form of objects then we have to take either an implementation class or Anonymous Inner class.

-->If we take implementation class as an alternative then it may allow its own data a part from the data Declared in interface and implementation class object should have its own identity instead of interface identity.

-->If we want to create an object with only interface identity and to allow only interface data we have to use Anonymous inner class as an alternative.

DURGASOFT

-->In jdbc applications getConnection() method will return connection object by returning anonymous Inner class object of connection interface.

NOTE: To create connection object taking an implementation class or anonymous inner class is Completely depending on the Driver Implementation.

3)Create either Statement or PreparedStatment or CallableStatement objects as per the requirement:

As part of the Jdbc applications after establish the connection between Java application and Database We have to prepare SQL Queries,we have to transfer SQL Queries to the databseEngine and we have to make Database Engine to execute SQL Queries.

-->To write and execute SQL Queries we have to use same predefined library from Statement prepared Statement and callableStatement.

-->To use the above required predefined library we have to prepare either Statement or preparedStatement or CallableStatement objects.

Q)What is the difference between Statement,PreparedStatement and Callable Statement Objects.
Ans:
-->In Jdbc applications when we have a requirement to execute all the SQL Queries independently we have to use Statement.

-->In jdbc applications when we have a requirement to execute the same SQL Query in the next Sequence where to improve the performance of JDBC application we will use prepared Statement.

-->In jdbc applications when we have a requirement to access stored procedures and functions available At Database from Java application we will use Callable Statement object.

-->To prepare Statement object we have to use the following method from Connection.

Public Statement createStatement()

Ex: Statement st=con.createStatement();

DURGASOFT

Where createStatement() method will return Statement object by creating Statement interfaces Anonymous inner class object.

4)Write and execute SQL Queries:

1.executeQuery()
2.executeUpdate()
3.execute()

Q)What are the differences between executeQuery(),executeUpdate() and execute() method?

Ans:Where executeQuery() method can be used to execute "selection group SQL Queries" in order to fetch(retrieve) data from Database.

-->when JVM encounter executeQuery() method with selection group SQL query then JVM will pickup

Selection group SQL Query,send to JdbcOdbcDriver,it will send to connection.Now connection will carrythat SQL Query to the database engine through Odbc Driver.

-->At database database engine will execute the selection group SQL Query by performing Query

Tokenization,Query parsing,Query optimization and Query Execution.

-->By the execution of selection group SQL Query database engine will fetch the data from database and

return to Java Application.

-->As Java technology is pure object oriented,Java application will store the fetched data in the form of

an object at heap memory called as "ResultSet".

-->As per the predefined implementation of executeQuery method JVM will return the generated

ResultSet object reference as return value from executeQuery() method.

Public ResultSet executeQuery(String sql_Query) throws SQLException

Ex: ResultSet rs=st.executeQuery("select * from emp1");

-->where executeUpdate() method can be used to execute updation group SQLQueries in order to

perform the database operations like create,insert,update,delete,Drop....
-->when JVM encounter updation group SQL Query with execteUpdate() method the JVM will pickup

That Sql Query and send to Database through Database connection.At Database side Database engine

DURGASOFT

Will execute it,perform updation from Database,identify rowCount value (number of records got updated) and return to Java application.

-->As per the predefined implementation of executeUpdate() method JVM will return row count value From executeUpdate() method.

Public int executeUpdate(String sql_Query) throws Exception

Ex: int rowCount=st.executeUpdate("update emp1 set esal=esal+500 where esal<1000");

-->In Jdbc applications execute() method can be used to execute both selection group and updation Group SQL Queries.

-->when JVM encounter selection group SQL Query with execute() method then JVM will send selection Group SQL Query to database engine,where database engine will execute it and send back the fetched Data to Java Application.

-->In Java application ResultSet object will be created with the fetched data but as per the predefined implementation of execute() method JVM will return "true" as a Boolean value.

-->when JVM encounter updation group SQL Query as parameter to execute() method then JVM Will send it to the database engine,where database engine will perform updations on database and return row Count value to Java application.But as per the predefined implementation of execute() method JVM will return "false" as Boolean value from execute() method

public Boolean execute(String sql_Query) throws SQLException.

Ex:
boolean b1=st.execute ("select * from emp1");

boolean b2=st.execcute("update emp1 set esal=esal+500 where esal<10000");

DURGASOFT

5)Close the connection:

In Jdbc applications after the database logic it is convention to close Database connection for this we have to used the following method.

Public void close() throws SQLException

Ex:con.close();

1)The following example demonstrate how to create a table on Database through a JDBC applicationby taking table name as Dynamic input.

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

//load a register driver

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

//establish connection between Java application and database
Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");

//prepare Statement
Statement st=con.createStatement();

//create BufferedReader
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

//take table name as dynamic input

System.out.println("Enter table name");

String tname=br.readLine();

//prepare SQLQuery

String sql="create table " + tname + "(eno number,ename varchar2(10),esal number)";

//execute SQL Query

st.executeUpdate(sql);
```

DURGASOFT

```
System.out.println("table created successfully");

//close the connection

con.close();

}}
```

2)The following example demonstrates how to insert no.of records on database table by taking records data as dynamic input.

```
import java.io.*;
import java.sql.*;
public class JdbcApp2 {
    public static void main(String[] args)throws Exception {
        Class.forName("oracle.jdbc.OracleDriver");
        Connection
        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","d
        urga");
        Statement st=con.createStatement();
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        while(true)
        {
            System.out.print("Employee Number  :");
            int eno=Integer.parseInt(br.readLine());
            System.out.print("Employee Name    :");
            String ename=br.readLine();
            System.out.print("Employee Salary  :");
            float esal=Float.parseFloat(br.readLine());
            System.out.print("Employee Address :");
            String eaddr=br.readLine();
            st.executeUpdate("insert into emp1
            values("+eno+",'"+ename+"','"+esal+",'"+eaddr+"')");
            System.out.println("Employee Inserted Successfully");
            System.out.print("Onemore Employee[Yes/No]? :");
            String option=br.readLine();
            if(option.equals("No"))
            {
                break;
            }
        }
        con.close();
    }}
```

DURGASOFT

In Jdbc applications if we want to used Type1 driver provided by sun micro systems then we have to usethe following Driver class and URL

driver.class:sun.jdbc.odbc.JdbcOdbcDriver
url:jdbc:odbc:dsnName

-->Similarly if we want to use Type4 Driver provided by oracle we have to use the following Driver class and URL

driver_class:oracle.jdbc.driver.OracleDriver
url:jdbc:oracle:thin:@localhost:1521:xe

-->Oracle Driver is a class provided by oracle software in the form of Ojdbc14.jar file

-->Oracle Software has provided ojdbc14.jar file at the following location

C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc.jar

-->If we want to use Type4 Driver provided by oracle in our Jdbc applications we have to set classpath environment variable to the location where we have ojdbc14.jar

D:\jdbc4>set
classpath=%classpath%;C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

3)The following example Demonstrate how to perform updations on Database table through Jdbc Application

```
import java.io.*;
import java.sql.*;
public class JdbcApp3 {
    public static void main(String[] args)throws Exception {
        //Class.forName("oracle.jdbc.OracleDriver");
        Connection  con=DriverManager.getConnection
        ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
        Statement st=con.createStatement();
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
        System.out.print("Bonus Amount  :");
        int bonus_Amt=Integer.parseInt(br.readLine());
        System.out.print("Salary Range  :");
        float sal_Range=Float.parseFloat(br.readLine());
        int rowCount=st.executeUpdate
        ("update emp1 set esal=esal+"+bonus_Amt+" where   esal<"+sal_Range);
        System.out.println("Employees Updated  :"+rowCount);
        con.close();
    }}
```


DURGASOFT

4)The following example demonstrates how to delete no.of records from database table through a Jdbc application

```
import java.io.*;
import java.sql.*;
public class JdbcApp4 {
    public static void main(String[] args)throws Exception {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
            Connection  con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
            System.out.print("Salary Range  :");
            float sal_Range=Float.parseFloat(br.readLine());
            int rowCount=st.executeUpdate("delete from emp1 where esal<"+sal_Range);
            System.out.println("Records Deleted  :"+rowCount);
            con.close();
    }
}
```

-->In jdbc application we will use executeUpdate() method to execute the Updation group SQL queries like create,insert,update,delete,drop,alter and so on.

-->If we execute the SQL Queries like insert,update and delete then really some no.of record will be updated on database table then that number will be return as rowCount value from executeUpdate().

-->If we execute the SQL Queries like create,alter,drop with executeUpdate() method then records manipulation is not available on database,in this context the return value from executeUpdate() method is completely depending on the type of Driver which we used in JDBC application.

-->In the above context if we use type1 Driver provided by SunMicroSystems the executeUpdate() method will return "-1" as rowCount value.

-->For the above requirement if we use Type4 Driver provided by oracle then executeUpdate() method will return "0" as rowCount value

DURGASOFT

```
5) import java.sql.*;
   public class JdbcApp5 {
   public static void main(String[] args)throws Exception {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
            Statement st=con.createStatement();
            int rowCount1=st.executeUpdate("create table emp1(eno number)");
            System.out.println(rowCount1);
            int rowCount2=st.executeUpdate("drop table emp1");
            System.out.println(rowCount2);
            con.close();
   }
}
```

```
6) import java.sql.*;
   public class JdbcApp6 {
   public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            int rowCount1=st.executeUpdate("create table emp1(eno number)");
            System.out.println(rowCount1);
            int rowCount2=st.executeUpdate("drop table emp1");
            System.out.println(rowCount2);
            con.close();
   }
}
```

DURGASOFT

ResultSet:

In Jdbc applications if we use Selection group SQL Query as parameter to executeQuery() method then JVM will send that selection group SQL Query to the Database Engine,where Database Engine will execute that SQL Query,fetch the data from Database and send back to Java application.

--->At Java Application the fetched data will be stored in the form of an object at heap memory called as ResultSet.

-->As per the predefined implementation of executeQuery method JVM will return the generated ResultSet object reference as return value.

ReslutSet rs=st.executeQuery("select * from emp1");

-->When ResultSet object is created automatically a cursor will be created positioned before the first record.

-->If we want to read the records data from resultset object the for each and every record we have to check whether the next record is available or not from resultset cursor position, if it is available then we have to move resultset cursor to the next record position.

-->To perform the above work we have to use the following method from resultset

public boolean next()

-->After getting ResultsSet cursor to a particular Record position we have to retrieve the data from respective columns,for this we have to use the following overloaded method

public xxx getxxx(int field_No)

public xxx getxxx(String field_Name)

where xxx may be byte,short,int.....

```
Ex: while(rs.next())
{
System.out.println(rs.getInt(1));
System.out.println(rs.getString(2));
System.out.println(rs.getFloat(3));
}
```

7)The following example demonstrate how to fetch the data from database through ResultSet object

```
import java.sql.*;
import oracle.jdbc.*;
public class JdbcApp7 {
    public static void main(String[] args)throws Exception {
        OracleDriver driver=new OracleDriver();
        Connection  con=DriverManager.getConnection
        ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery("select * from emp1");
        System.out.println("ENO\tENAME\tESAL\tEADDR");
        System.out.println("---------------------------");
        while(rs.next()){
        System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
```


DURGASOFT

```
                +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
                }
                con.close();
        }}
```

-->In jdbc applications execute() method can be used to execute both selection group SQL Queries and updation group SQL Queries one at a time.

-->If we use execute() method to execute selection group SQL Query then JVM will send that SQL Query to database engine where Database engine will execute Selection group SQL Query, fetch data from database and return to Java application.At java application the fetched data will be stored in the form of ResultSet object but as per the internal implementation of execute() method JVM will return "true" as a boolean value.

-->In the above context to retrieve the data from resultset object we have to get ResultSetObject reference explicitly.

-->To get ResultSet object reference explicitly we have to use the following method from Statement.

Public ResultSet getResultSet() throws SQLException

```
8.  import java.sql.*;
    public class JdbcApp8 {
    public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
            Statement st=con.createStatement();
            boolean b=st.execute("select * from emp1");
            System.out.println(b);
            ResultSet rs=st.getResultSet();
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("----------------------------");

            while(rs.next()){
                    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
                    +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));

            }
```


DURGASOFT

```
                con.close();
        }}
```

-->If we use updation group SQL Query as parameter to execute() method then JVM will send that updation group SQL Query to Database engine,where database engine will execute it,perform updations on Database table and return the generated row Count value to Java application.

-->As per the predefined implementation of execute() method JVM will return "false" as a boolean value from execute() method

-->In the above context to retrieve the generated rowCount value we have to use the following method from statement

public int getUpdateCount() throws SQLException

```
9. import java.sql.*;
  public class JdbcApp9
  {
      public static void main(String[] args)throws Exception
      {
                Class.forName("oracle.jdbc.OracleDriver");
                Connection con=DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
                Statement st=con.createStatement();
                boolean b=st.execute("update emp1 set esal=esal+500 where esal<10000");
                System.out.println(b);
                int rowCount=st.getUpdateCount();
                System.out.println("Records Updated   :"+rowCount);
                con.close();
      }
  }
```

Q)If we use updation group SQL Query as parameter to executeQuery() method then what will be the response from JDBC application?
Ans:In Jdbc applications if we use updation group SQL Query as parameter to executeQuery() method then JVM will send updation group SQL Query to DatabaseEngine,where Database Engine will perform updations and Database and return rowCount Value to Java application but as per the predefined implementation of executeQuery() method JVM will expect ResultSet Object.

-->In the above context raising an exception or not to raise an exception is completely depending on the Type of Driver which we used.

DURGASOFT

-->For the above requirement if we use Type1 driver then JVM will raise an Exception like java.sql.SQLException:no ResultSet was produced.

```
10.import java.sql.*;
public class JdbcApp12 {
public static void main(String[] args) {
        Statement st=null;
        try{
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                Connection con=DriverManager.getConnection
                ("jdbc:odbc:nag","system","durga");
                st=con.createStatement();
                ResultSet rs=st.executeQuery
                ("update emp1 set esal=esal+500 where esal<10000");
        }
        catch (Exception e){
                e.printStackTrace();
                try{
                        int rowCount=st.getUpdateCount();
                        System.out.println("Row Count :"+rowCount);
                }catch(Exception e1){
                        e1.printStackTrace();
                }
}}}
```

NOTE:In the above application if we provide getUpdateCount method then we are able to generated the rowCount value in the catch block.

-->For the above requirement if we use Type4 Driver provided by oracle then JVM will not raise any exception,it will prepare a default ResultSet object implicitly.

```
import java.util.*;
class TestTwo{
public static void main(String args[]){
Statement st=null;
try{
Class.forName("oracle.jdbc.driver.oracleDriver");
Connection
con=DriverManager.getConnection(jdbc:oracle:thin:@localhost:1521:xe","system","durga");
st=con.createStatement();
ResultSet rs=st.executeQuery("update emp2 set esal=esal+500 where esal<10000");
```


DURGASOFT

```
int rowCount=st.getUpdateCount();
System.out.println("Records Updated.."+rowCount);
}
catch(Exception e){
e.printStackTrace();
}}}
```

Q)In Jdbc applications if we provide selection group SQL Query as parameter to executeUpdate() Method then what will be the response from Jdbc application?

Ans:In jdbc applications if we provide selection group SQL Query as parameter to executeUpdate() method then JVM will send that SQL Query to database Engine,where Database engine will fetch the data from database and return to Java application.

-->At java application the returned data will be stored in the form of ResultSet object.

-->As per the predefined implementation of executeUpdate() method JVM will expecting Integer value.

-->In the above context getting an exception or not is completely depends on the Driver which we used in our Jdbc application

-->For the above requirement if we use Type1 Driver then JVM will raise an exception like java.sql.SQLException:no rwoCount was produced.

NOTE:In the above situation if we getResultSet() method in catch block then we are able to get Resultset object.

```
11. import java.sql.*;
    public class JdbcApp10 {
    public static void main(String[] args) {
        Statement st=null;
        try{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
        st=con.createStatement();
        int rowCount= st.executeUpdate("select * from emp1");
        }catch(Exception e){
            e.printStackTrace();
            try{
                ResultSet rs=st.getResultSet();
                System.out.println("ENO\tENAME\tESAL\tEADDR");
                System.out.println("------------------------");
```


DURGASOFT

```
                        while(rs.next())
                        {
                                System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
                                +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
                        }
                }catch(Exception e1){
                        e1.printStackTrace();
                }
        }
    }
}
```

-->For the above requirement if we use Type4 Driver then JVM will not raise any Exception, executeUpdate() method will return how many no.of records are retrieved from Database table.

```
12. import java.sql.*;
    public class JdbcApp11 {
    public static void main(String[] args) {
            try{
                    Class.forName("oracle.jdbc.OracleDriver");
                    Connection con=DriverManager.getConnection
                    ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
                    Statement st=con.createStatement();
                    int rowCount=st.executeUpdate("select * from emp1");
                    System.out.println("Row Count  :"+rowCount);
                    ResultSet rs=st.getResultSet();
                    System.out.println("ENO\tENAME\tESAL\tEADDR");
                    System.out.println("----------------------------");



                    while(rs.next())
                    {
                            System.out.println(rs.getInt(1)+"\t"+rs.getString(2)
                            +"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
                    }
            }
            catch (Exception e)
            {
                    e.printStackTrace();
            }
    }
}
```


DURGASOFT

13.The following example demonstrated how to retrieve the Data from Database and how to display that data through an HTML page

```
import java.sql.*;
import java.io.*;
public class JdbcApp14
{
    public static void main(String[] args)throws Exception
    {
        Class.forName("oracle.jdbc.OracleDriver");
        Connection con=DriverManager.getConnection
        ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
        Statement st=con.createStatement();
        ResultSet rs=st.executeQuery("select * from emp1");
        String data="";
        data=data+"<html><body><center><table border='1' bgcolor='lightblue'>";

    data=data+"<tr><td>ENO</td><td>ENAME</td><td>ESAL</td><td>EADDR</td></tr>";
        while(rs.next())
        {
            data=data+"<tr>";
            data=data+"<td>"+rs.getInt(1)+"</td><td>
            "+rs.getString(2)+"</td><td>"+rs.getFloat(3)+"</td><td>
            "+rs.getString(4)+"</td>";
            data=data+"</tr>";

        }
        data=data+"</table></center></body></html>";
        FileOutputStream fos=new FileOutputStream("emp.html",true);
        byte[] b=data.getBytes();
        fos.write(b);
        System.out.println("Open emp.html file to get Employees data");
        fos.close();
        con.close();
    }
}
```

DURGASOFT

Jdbc-awt app1:

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class LoginFrame extends Frame implements ActionListener {
    Label l1,l2;
    TextField tf1,tf2;
    Button b1;
    String status="";
    public LoginFrame() {
            this.setVisible(true);
            this.setSize(500,500);
            this.setTitle("Login Frame");
            this.setBackground(Color.green);
            this.setLayout(new FlowLayout());
            this.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent we){
                            System.exit(0);
                    }
            });

            l1=new Label("User Name");
            l2=new Label("Password");
            tf1=new TextField(20);
            tf2=new TextField(20);
            tf2.setEchoChar('*');
            b1=new Button("Login");
            b1.addActionListener(this);

            Font f=new Font("arial",Font.BOLD,20);
            l1.setFont(f);
            l2.setFont(f);
            tf1.setFont(f);
```

DURGASOFT

```
            tf2.setFont(f);
            b1.setFont(f);

            this.add(l1);
            this.add(tf1);
            this.add(l2);
            this.add(tf2);
            this.add(b1);
    }
    public void actionPerformed(ActionEvent ae) {
            String uname=tf1.getText();
            String upwd=tf2.getText();
            UserService us=new UserService();
            status=us.checkLogin(uname,upwd);
            repaint();
    }
    public void paint(Graphics g){
            Font f=new Font("arial",Font.BOLD,30);
            g.setFont(f);
            g.drawString("Status  :"+status, 50,250);
    }
}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class UserService {
    Connection con;
    Statement st;
    ResultSet rs;
    String status="";
    public UserService() {
            try {
                    Class.forName("oracle.jdbc.OracleDriver");
                    con=DriverManager.getConnection
                     ("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");
                    st=con.createStatement();
            } catch (Exception e) {
                    e.printStackTrace();
            }
    }


    public String checkLogin(String uname, String upwd){
            try {
```

DURGASOFT

```
                rs=st.executeQuery("select * from registered_Users where
                uname='"+uname+"' and upwd='"+upwd+"'");
                boolean b=rs.next();
                if(b==true){
                        status="Login Success";
                }else{
                        status="Login Failure";
                }
        } catch (Exception e) {
                e.printStackTrace();
        }
        return status;
    }

}



package com.durgasoft;
public class JdbcApp15 {

    public static void main(String[] args) {
        LoginFrame lf=new LoginFrame();

    }

}
```

Jdbc-awt app2

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
```


DURGASOFT

```
public class EmployeeAddFrame extends Frame implements ActionListener {
    Label l1,l2,l3,l4;
    TextField tf1,tf2,tf3,tf4;
    Button b1;
    String status="";
    public EmployeeAddFrame(){
        this.setVisible(true);
        this.setSize(500, 500);
        this.setTitle("Employee Registration Frame");
        this.setBackground(Color.cyan);
        this.setLayout(null);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        });

        l1=new Label("Employee Number");
        l2=new Label("Employee Name");
        l3=new Label("Employee Salary");
        l4=new Label("Employee Address");

        tf1=new TextField(20);
        tf2=new TextField(20);
        tf3=new TextField(20);
        tf4=new TextField(20);

        b1=new Button("ADD");
        b1.addActionListener(this);

        Font f=new Font("arial",Font.BOLD,20);
        l1.setFont(f);
        l2.setFont(f);
        l3.setFont(f);
        l4.setFont(f);
        tf1.setFont(f);
        tf2.setFont(f);
        tf3.setFont(f);
        tf4.setFont(f);
        b1.setFont(f);

        l1.setBounds(50, 100, 200, 25);
        tf1.setBounds(250, 100, 200, 30);
        l2.setBounds(50, 150, 200, 25);
        tf2.setBounds(250, 150, 200, 30);
        l3.setBounds(50, 200, 200, 25);
        tf3.setBounds(250, 200, 200, 30);
        l4.setBounds(50, 250, 200, 25);
```


DURGASOFT

```
            tf4.setBounds(250, 250, 200, 30);
            b1.setBounds(50, 300, 100, 30);
            this.add(l1);
            this.add(tf1);
            this.add(l2);
            this.add(tf2);
            this.add(l3);
            this.add(tf3);
            this.add(l4);
            this.add(tf4);
            this.add(b1);
    }
    public void actionPerformed(ActionEvent ae){
            try {
                    int eno=Integer.parseInt(tf1.getText());
                    String ename=tf2.getText();
                    float esal=Float.parseFloat(tf3.getText());
                    String eaddr=tf4.getText();

                    EmployeeService es=new EmployeeService();
                    status=es.add(eno,ename,esal,eaddr);
                    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:"+status, 50, 400);
    }

}

package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class EmployeeService {
    Connection con;
    Statement st;
    ResultSet rs;
    String status="";
```


DURGASOFT

```
    public EmployeeService() {
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");
            st=con.createStatement();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public String add(int eno,String ename, float esal, String eaddr){
        try {
            rs=st.executeQuery("select * from emp1 where enno="+eno);
            boolean b=rs.next();
            if(b==true){
                status="Employee Existed Already";
            }else{
                st.executeUpdate("insert into emp1
                values("+eno+",'"+ename+"',"+esal+",'"+eaddr+"')");
                status="Employee Registration Success";
            }
        } catch (Exception e) {
            status="Employee Registration Failure";
            e.printStackTrace();
        }
        return status;
    }

}


package com.durgasoft;

public class JdbcApp16 {

    public static void main(String[] args) {
        EmployeeAddFrame f=new EmployeeAddFrame();

    }

}
```

DURGASOFT

Jdbc-Awt App3

```
package com.durgasoft;
import java.awt.Button;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.GraphicsConfiguration;
import java.awt.HeadlessException;
import java.awt.Label;
import java.awt.TextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class StudentSearchFrame extends Frame implements ActionListener {
    Label l1;
    TextField tf1;
    Button b1;
    StudentTo sto;
    public StudentSearchFrame() {
        this.setVisible(true);
        this.setSize(500, 500);
        this.setTitle("Student Search Frame");
        this.setLayout(new FlowLayout());
        this.setBackground(Color.green);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        });

        l1=new Label("Student Id");
        tf1=new TextField(20);
        b1=new Button("Search");
        b1.addActionListener(this);

        Font f=new Font("arial",Font.BOLD,20);
        l1.setFont(f);
        tf1.setFont(f);
        b1.setFont(f);

        this.add(l1);
```


DURGASOFT

```
            this.add(tf1);
            this.add(b1);
        }
        public void actionPerformed(ActionEvent arg0) {
            String sid=tf1.getText();
            StudentService ss=new StudentService();
            sto=ss.search(sid);
            repaint();
        }
        public void paint(Graphics g){
            Font f=new Font("arial",Font.BOLD,30);
            g.setFont(f);
            if(sto==null){
                g.drawString("Student Not Existed", 50, 300);
            }else{
                g.drawString("Student Id :"+sto.getSid(), 50, 250);
                g.drawString("Student Name :"+sto.getSname(), 50, 300);
                g.drawString("Student Address :"+sto.getSaddr(), 50, 350);
            }
        }
    }


    package com.durgasoft;
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;

    public class StudentService {
        Connection con;
        Statement st;
        ResultSet rs;
        StudentTo sto;
        public StudentService() {
            try {
                Class.forName("oracle.jdbc.OracleDriver");
                con=DriverManager.getConnection("jdbc:oracle:oci8:@xe", "system",
"durga");
                st=con.createStatement();
            } catch (Exception e) {

            }
        }
```

DURGASOFT

```
    public StudentTo search(String sid){
        try {
            rs=st.executeQuery("select * from student where sid='"+sid+"'");
            boolean b=rs.next();
            if(b==true){
                sto=new StudentTo();
                sto.setSid(rs.getString(1));
                sto.setSname(rs.getString(2));
                sto.setSaddr(rs.getString(3));
            }else{
                sto=null;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return sto;
    }

}
```

```
package com.durgasoft;
public class StudentTo {
    private String sid;
    private String sname;
    private String saddr;
    public String getSid() {
        return sid;
    }
    public void setSid(String sid) {
        this.sid = sid;
    }
    public String getSname() {
        return sname;
    }
    public void setSname(String sname) {
        this.sname = sname;
    }
    public String getSaddr() {
        return saddr;
    }
    public void setSaddr(String saddr) {
        this.saddr = saddr;
```

DURGASOFT

```
        }

    }
    package com.durgasoft;

    public class JdbcApp17 {

        public static void main(String[] args) {
                StudentSearchFrame sf=new StudentSearchFrame();

        }

    }
```

Jdbc-awt app4

```
    package com.durgasoft;
    import java.awt.Button;
    import java.awt.Color;
    import java.awt.FlowLayout;
    import java.awt.Font;
    import java.awt.Frame;
    import java.awt.Graphics;
    import java.awt.Label;
    import java.awt.TextArea;
    import java.awt.event.ActionEvent;
    import java.awt.event.ActionListener;
    import java.awt.event.WindowAdapter;
    import java.awt.event.WindowEvent;
    import java.util.ArrayList;

    public class EditorFrame extends Frame implements ActionListener {
        Label l;
        TextArea ta;
        Button b;
        EditorService es;
        boolean bol;
        public EditorFrame(){
                this.setVisible(true);
                this.setSize(700, 800);
                this.setTitle("SQL Editor Frame");
                this.setBackground(Color.green);
                this.setLayout(new FlowLayout());
                this.addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e){
                                System.exit(0);
```

DURGASOFT

```
                }
            });

            l=new Label("Provide SQLQuery");
            ta=new TextArea(3,30);
            b=new Button("Execute");
            b.addActionListener(this);

            Font f=new Font("arial",Font.BOLD,20);
            l.setFont(f);
            ta.setFont(f);
            b.setFont(f);

            this.add(l);
            this.add(ta);
            this.add(b);


        }
        public void actionPerformed(ActionEvent ae){
            String query=ta.getText();
             es=new EditorService();
            bol=es.execute(query);
            repaint();
        }
        public void paint(Graphics g){
            Font f=new Font("arial",Font.BOLD,30);
            g.setFont(f);
            if(bol==true){
                ArrayList al=es.getEmps();
                g.drawString("ENO   ENAME   ESAL   EADDR",50,200);
                g.drawString("-------------------------------------", 50, 240);
                int y=300;
                for(int i=0;i<al.size();i++){
                        EmployeeTo eto=(EmployeeTo)al.get(i);
                g.drawString(eto.getEno()+"    "+eto.getEname()+"    "+eto.getEsal()+"
                "+eto.getEaddr(), 50, y);
                        y=y+50;
                }

            }else{
                int rowCount=es.getRowCount();
                g.drawString("Row Count  :"+rowCount, 50, 300);
            }
        }
    }
```

DURGASOFT

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

public class EditorService {
	Connection con;
	Statement st;
	ResultSet rs;
	ArrayList al;
	boolean bol;
	int rowCount;
	public EditorService() {
		try {
			Class.forName("oracle.jdbc.OracleDriver");
			con=DriverManager.getConnection("jdbc:oracle:oci8:@xe", "system",
"durga");
			st=con.createStatement();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	public boolean execute(String sql_Query){
		try {

			bol=st.execute(sql_Query);
		} catch (Exception e) {
			e.printStackTrace();
		}
		return bol;
	}
	public ArrayList getEmps(){
		al=new ArrayList();
		try {
			rs=st.getResultSet();
			while(rs.next()){
				EmployeeTo eto=new EmployeeTo();
				eto.setEno(rs.getInt(1));
				eto.setEname(rs.getString(2));
				eto.setEsal(rs.getFloat(3));
				eto.setEaddr(rs.getString(4));
				al.add(eto);
```

DURGASOFT

```
                        }
                }


        catch (Exception e) {
                        e.printStackTrace();
                }
                return al;
        }
        public int getRowCount(){
                try {
                        rowCount=st.getUpdateCount();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return rowCount;
        }
}



package com.durgasoft;

public class EmployeeTo {
        private int eno;
        private String ename;
        private float esal;
        private String eaddr;
        public int getEno() {
                return eno;
        }
        public void setEno(int eno) {
                this.eno = eno;
        }
        public String getEname() {
                return ename;
        }
        public void setEname(String ename) {
                this.ename = ename;
        }
        public float getEsal() {
                return esal;
        }
        public void setEsal(float esal) {
                this.esal = esal;
        }
        public String getEaddr() {
                return eaddr;
```

DURGASOFT

```
        }
        public void setEaddr(String eaddr) {
                this.eaddr = eaddr;
        }

    }

    package com.durgasoft;

    public class JdbcApp18 {

        public static void main(String[] args) {
                EditorFrame f=new EditorFrame();

        }

    }
```

ResultSet Types:

In jdbc applications ResultSets can be divided into two types:

-->As per the ResultSet concurrency there are two types of ResultSets.

1)Read only ResultSet
It is a ResultSet object,it will allow the users to read the data only.
To represent this ResultSet object ResultSet interface has provided the following constant

public static final int CONCUR_READ_ONLY

2)Updatable ResultSet:

It is a ResultSet object,it will allow the users to perform updations on its content.

To represent this resultset object ResultSet interface has provided the following constant.

public static final int CONCUR_UPDATABLE

-->As per the ResultSet cursor movement there are two types of ResultSets

DURGASOFT

1)Forward only ResultSet:

It is ResultSet object,it will allow the users to iterate the data in forward directiononly.

-->To represent this ResultSet,ResultSet interface has provided the following constant

public static final int TYEP_FORWARD_ONLY

2)Scrollable ResultSet:

These are the Resultset objects,which will allow the users to iterate the data in both

forward and backward directions.

There are two types of Scrollable Resultsets:
1)Scroll sensitive ResultSet
2)Scroll Insensitive ResultSet

Q)What are the differences betwwn Scroll Sensitive ResultSet and Scroll Insensitive ResultSet?

Scroll sensitive ResultSet is a Scrollable resultset object,which will allow the later Database updations.

-->To refer this ResultSet,ResultSet interface has provided the following constant

public static final int TYPE_SCROLL_SENSITIVE

-->Scroll Insensitive ResultSet is scrollable Resultset object,which will not allow the later database updations after creation.

-->To represent this ResultSet,ResultInterface has provided the following constant

public static final int TYPE_SCROLL_INSENDITIVE

-->The default ResultSet type in Jdbc applications is Forward only and Read only.

-->In Jdbc applications if we want to specify a particular type to the ResultSet object then we have to provide the above specified ResultSet constants at the time of creating Statement object,for this we have to use the following method

public Statement createStatement(int forwardonly

DURGASOFT

/scrollsensitive/scrollinsensitive,int Readonly/updatable)

Ex: Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

-->In jdbc applications by using scrollable ResultSet we are able to retrieve the data in both forward direction and backward direction.

-->To retrieve the data in forward direction for each and every record we have to check whether the next record is available or not,if it is available we have to move resultset cursor to the next record position.when we refer a particular record then we have to retrieve the data from the respective columns.To achieve this we have to use the following piece of code:

```
while(rs.next())
{
System.out.println(rs.getInt(1));
System.out.println(rs.getString(2));

}
```

-->If we want to retrieve the data in Backward direction then for each and every record we have to check whether the previous record is available or not from the resultset cursor position,if is available then we have to move resultset cursor to the previous record.To achieve this we have to use the following methods:

public boolean previous()

-->After moving the resultset cursor to particular record then we have to retrieve the data from the corresponding columns for this we have to use the following methods

public xxx getxxx(int field_NUM)
public xxx getxxx(String field_Name)

where xxx may be byte,short,int.....

DURGASOFT

```
Ex: while(rs.previous())
{
System.out.println(rs.getInt(1));
System.out.println(rs.getString(2));
System.out.println(rs.getFloat(3));

}
```

```
14. package com.durgasoft;
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;

public class JdbcApp19 {

    public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection con=DriverManager.getConnection
            ("jdbc:oracle:oci8:@xe", "system","durga");
            Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE );
            ResultSet rs=st.executeQuery("select * from emp1");
            System.out.println("Data In Forward Direction");
            System.out.println("ENO\tENAME\tESAL\tEADDR\t");
            System.out.println("--------------------------------");
            while(rs.next()){
            System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"
            +rs.getFloat(3)+"\t"+rs.getString(4));
            }
            System.out.println("Data In Backward Direction");
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("------------------------------");
            while(rs.previous()){

                    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"
                    +rs.getFloat(3)+"\t"+rs.getString(4));
            }
            con.close();
    }
}
```

DURGASOFT

15.

```
package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp20 {

    public static void main(String[] args)throws Exception {
            Class.forName("oracle.jdbc.OracleDriver");
            Connection con=DriverManager.getConnection
            ("jdbc:oracle:oci8:@xe", "system", "durga");
            Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
            ResultSet.CONCUR_UPDATABLE );
            ResultSet rs=st.executeQuery("select * from emp1");
            rs.afterLast();
            rs.previous();
            System.out.println(rs.getInt(1));
            rs.beforeFirst();
            rs.next();
            System.out.println(rs.getInt(1));
            rs.last();
            System.out.println(rs.getInt(1));
            rs.first();
            System.out.println(rs.getInt(1));
            rs.absolute(3);
            System.out.println(rs.getInt(1));
            rs.absolute(-3);
            System.out.println(rs.getInt(1));
            rs.first();
            rs.relative(2);
            System.out.println(rs.getInt(1));
            rs.last();
            rs.relative(-2);
            System.out.println(rs.getInt(1));
    }

}
```

DURGASOFT

Jdbc-Awt app5

```
package com.durgasoft;

import java.awt.Button;
import java.awt.Color;
import java.awt.Font;
import java.awt.Frame;
import java.awt.Graphics;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

public class PlayerFrame extends Frame implements ActionListener {
    Button b1,b2,b3,b4;
    String label;
    EmployeeTo eto;
    EmployeeService es;
    public PlayerFrame() {
        this.setVisible(true);
        this.setSize(500, 500);
        this.setTitle("Player Frame");
        this.setBackground(Color.green);
        this.setLayout(null);
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we){
                System.exit(0);
            }
        });

        b1=new Button("First");
        b2=new Button("Next");
        b3=new Button("Previous");
        b4=new Button("Last");

        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        b4.addActionListener(this);

        Font f=new Font("arial",Font.BOLD,20);
        b1.setFont(f);
        b2.setFont(f);
        b3.setFont(f);
        b4.setFont(f);
```

DURGASOFT

```
            b1.setBounds(50, 400, 100, 30);
            b2.setBounds(160, 400, 100, 30);
            b3.setBounds(270, 400, 100, 30);
            b4.setBounds(380, 400, 100, 30);
            this.add(b1);
            this.add(b2);
            this.add(b3);
            this.add(b4);
            es=new EmployeeService();
        }
        public void actionPerformed(ActionEvent ae){
            label=ae.getActionCommand();
            eto=es.getEmployee(label);
            repaint();
        }
        public void paint(Graphics g){
            Font f=new Font("arial",Font.BOLD,30);
            g.setFont(f);
            String msg=es.getMsg();
            if(msg.equals("")){
                g.drawString("Employee Number :"+eto.getEno(), 50,100);
                g.drawString("Employee Name   :"+eto.getEname(), 50,150);
                g.drawString("Employee Salary  :"+eto.getEsal(), 50,200);
                g.drawString("Employee Address :"+eto.getEaddr(), 50,250);
            }else{
                g.drawString(msg, 50,300);
            }
        }
    }


package com.durgasoft;

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

public class EmployeeService {
    Connection con;
    Statement st;
    ResultSet rs;
    EmployeeTo eto;
    String msg="";
    boolean b=false;
```

DURGASOFT

```
public EmployeeService() {
        try {
                Class.forName("oracle.jdbc.OracleDriver");
                con=DriverManager.getConnection
                ("jdbc:oracle:thin:@localhost:1521:xe", "system","durga");
                st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
                ResultSet.CONCUR_UPDATABLE);
                rs=st.executeQuery("select * from emp1");
        } catch (Exception e) {
                e.printStackTrace();
        }
}
public EmployeeTo getEmployee(String label){
        try {
                if(label.equals("First")){
                        rs.first();
                        msg="";
                }
                if(label.equals("Next")){
                        b=rs.next();
                        if(b==false){
                                msg="No More Records In Forward Direction";
                        }else{
                                msg="";
                        }
                }
                if(label.equals("Previous")){
                        b=rs.previous();
                        if(b==false){
                                msg="No More Records In Backward Direction";
                        }else{
                                msg="";
                        }
                }
                if(label.equals("Last")){
                        rs.last();
                        msg="";
                }
                eto=new EmployeeTo();
                eto.setEno(rs.getInt(1));
                eto.setEname(rs.getString(2));
                eto.setEsal(rs.getFloat(3));
                eto.setEaddr(rs.getString(4));
        } catch (Exception e) {
                e.printStackTrace();
        }
        return eto;
}
```

DURGASOFT

```
    public String getMsg(){
        return msg;
    }
}



package com.durgasoft;

public class EmployeeTo {
    private int eno;
    private String ename;
    private float esal;
    private String eaddr;
    public int getEno() {
        return eno;
    }
    public void setEno(int eno) {
        this.eno = eno;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
    public float getEsal() {
        return esal;
    }
    public void setEsal(float esal) {
        this.esal = esal;
    }
    public String getEaddr() {
        return eaddr;
    }
    public void setEaddr(String eaddr) {
        this.eaddr = eaddr;
    }

}
```

DURGASOFT

```
package com.durgasoft;


public class JdbcApp24 {

    public static void main(String[] args) {
            PlayerFrame pf=new PlayerFrame();
    }

}
```

Scroll Sensitive ResultSet:

```
16.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class JdbcApp25 {
   public static void main(String[] args)throws Exception {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
       Statement st=con.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
     ResultSet.CONCUR_UPDATABLE);
      ResultSet rs=st.executeQuery("select * from emp1");
      System.out.println("Data Before Updations");
      System.out.println("ENO    ENAME    ESAL    EADDR");
      System.out.println("----------------------------------------");
      while(rs.next()){
            System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"
     "+rs.getString(4));
      }
      System.out.println("Application is in Pausing state, please update database");
      System.out.in.read();
      rs.beforeFirst();
      System.out.println("Data After Updations");
      System.out.println("ENO    ENAME    ESAL    EADDR");
```


DURGASOFT

```
        System.out.println("-------------------------------------");
        while(rs.next()){
          rs.refreshRow();
              System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"
       "+rs.getString(4));
        }
        con.close();
      }

    }
```

-->To move ResultSet cursor to before first record we have to use the following method from ResultSet

public void beforeFirst()

-->To move ResultSet cursor to After the last record we have to use the following method from ResultSet

public void afterLast()

-->To move ResultSet cursor to a particular record we have to use the following method from ResultSet

public void absolute(int rec_position)

-->In case of scroll sensitive ResultSet objects to reflect later database updations into the ResultSet object we have to refresh each and every record for this we have to use the following method from ResultSet

public void refreshRow()

-->where refreshRow() method can be used to refresh only one record.

-->If we use Type4 Driver provided by oracle in the above application then JVM will raise an exception like java.sql.SQLException:unsupportedfeature:refreshrow

-->In jdbc applications scroll sensitive ResultSet object should be supported by Type1 Driver provided by Sun MicroSystems,which could not be supported by Type4 Driver provided by oracle.

-->In Jdbc applications scroll Insensitive ResultSet object could not be supported by both Type1 Driver provided by Sun MicroSystems and Type4 Driver provided by oracle.

-->In jdbc applications the main purpose of UpdatableResultSet object is to perform updations on its content in order to perform the manipulations with the data available at Database

DURGASOFT

-->In jdbc applications Updatable ResultSet objects can be used to insert records on database table to achieve the above requirement we have to use the following steps:

Step1:get Updatable ResultSet object

```
Statement
st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

ResultSet rs=st.executeQuery("select * from emp1");
```

Step2: After creating ResultSet Object we have to move ResultSet cursor to end of the ResultSet object
where we have to take a buffer to insert new Record data temporarily to achieve this we have to use the following method from ResultSet

public void moveToInsertRow()

```
Ex:rs.moveToInsertRow();
```

Step3:Insert record data on resultset object temporarily to do this we have to use the following method

public void updatexxx(int field_Num,xxx value)

```
Ex:rs.updateInt(1,555);
  rs.updateString(2,'xys');
  rs.updateFloat(3,9000);
```

Step4:Make the temporarily insertion as permanent insertion in resultset object as well as on database table to achieve this we have to use the following method

public void insertRow()

```
ex:rs.insertRow();
```

NOTE:The main advantage of this updatable ResultSet object is to perform updations on database table without using SQL Queries.

DURGASOFT

17.
The following example demonstrates how to insert no.of records into database table through a Jdbc application

```
package com.durgasoft;
    import java.io.BufferedReader;
    import java.io.InputStreamReader;
    import java.sql.Connection;
    import java.sql.DriverManager;
    import java.sql.ResultSet;
    import java.sql.Statement;

    public class Jdbcapp26 {

      public static void main(String[] args)throws Exception{
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:nag","system","durga");
         Statement
       st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE
       );

     ResultSet rs=st.executeQuery("select * from emp1");
        rs.moveToInsertRow();
        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

     while(true){
          System.out.print("Employee Number   :");
          int eno=Integer.parseInt(br.readLine());
          System.out.print("Employee Name     :");
          String ename=br.readLine();
          System.out.print("Employee Salary    :");
          float esal=Float.parseFloat(br.readLine());
          System.out.print("Employee Address  :");
          String eaddr=br.readLine();
          rs.updateInt(1, eno);
          rs.updateString(2, ename);
          rs.updateFloat(3, esal);
          rs.updateString(4, eaddr);
          rs.insertRow();

    System.out.println("Employee inserted Successfully");
          System.out.print("Onemore Employee[Yes/no]  :");
          String option=br.readLine();
          if(option.equals("no")){
            break;
          }
        }
```


DURGASOFT

```
        con.close();
    }

  }
```

NOTE:In jdbc applications updatable ResultSets could not be supported by Type 4 Driver provided by oracle

-->In jdbc applications by using updatable ResultSet object it is possible to update database

-->To perform this we have to use the following steps

Step1:get updatable ResultSet object

Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);

ResultSet rs=st.executeQuery("select * from emp1");

Step2:update Resultset Object Temporarily

To achieve this we have to use the following method

public void updatexxx(int field_Name,xxx value)

ex: rs.updateFloat(3,7000.0f);

Step3:Make temporary updation as permanent updation on resultset obejct as well as on database to achieve this we have to use the following method

public void updateRow()

Ex:rs.updateRow();

```
18.
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp27 {
```


DURGASOFT

```
    public static void main(String[] args)throws Exception {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Connection con=DriverManager.getConnection("jdbc:odbc:nag", "system","durga");
      Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
     ResultSet.CONCUR_UPDATABLE);


      ResultSet rs=st.executeQuery("select * from emp1");
      while(rs.next()){
        float esal=rs.getFloat(3);
        if(esal<10000){
          float new_Sal=esal+500;
          rs.updateFloat(3, new_Sal);
          rs.updateRow();
        }
      }
      con.close();
    }

  }
```

-->In Jdbc applications by using updatable ResultSet object it is possible to delete records on database table to achieve this we have to use the following method

public void deleteRow()

Ex:rs.deleteRow();

```
19.
package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class Jdbcapp28 {
   public static void main(String[] args)throws Exception {
      Class.forName("oracle.jdbc.OracleDriver");
       Connection
     con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
     );
      Statement st=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
     ResultSet.CONCUR_UPDATABLE);
      ResultSet rs=st.executeQuery("select * from emp1");
      rs.last();
```


DURGASOFT

```
        rs.deleteRow();
        st.close();
        con.close();
    }

  }
```

-->To move ResultSet cursor to a particular Record position we have to use the following method from resultset

public void absolute(int position)

-->To move ResultSet cursor over some no.of records we have to use the following method from Resultset

public void relative(int no.of.records)

-->If we have bulk of records in database table,where if we are trying to retrieve all the records at a time into resultset object automatically Jdbc application performance will be reduced.

-->In the above context to improve the performance of Jdbc application we have to fetch the limited no. of records in multiple attempts.

-->To specify the no.of records which we want to fetch at an attempt then we have to use the following method

public void setFetchSize(int size)

-->To get the specified fetch size value from resultset we have to use the following method

public int getFetchSize

What is the difference between Statement and PreparedStatement?
Ans:
In Jdbc applications,when we have a requirement to execute the SQL queries independently,we have to use Statement.
In Jdbc applications,when we have a requirement to execute same SQL query in the next sequence where to improve the performance of Jdbc applications,we have to use PreparedStatement.
To achieve the above requirement,if we use Statement,then for every time of executing the same SQL query,DB engine has to perform Query Tokenization,Query Parsing,Query Optimization and Query Execution without having any validation from one time to another time.

DURGASOFT

This approach will reduce the performance of Jdbc application.

In the above context,to improve the performance of Jdbc applications,we have to use an alternative where we have to perform Query Processing one time inorder to perform or execute the same SQL Query in the next sequence.

To achieve the above alternative,we have to use PreparedStatement over Statement.

If we want to use PreparedStatement in Jdbc applications we have to use the following steps.

1.Create PreparedStatement object by providing generalised SQL Query:

To create PreparedStatement object,we have to use the following method from Connection.

```
public PreparedStatement prepareStatement(String SQL_format)
```

Eg:

```
PreparedStatement pst=con.PrepareStatement("insert into emp1 values(?,?,?)");
```

When JVM encounter the above instruction,JVM will pickup the provided SQL query and send to DB engine through Jdbc Driver and connection.

Upon receiving SQL query,DB engine will perform query processing and prepare query plan with the parameters,as a result PreparedStatement object will be created at Java application with the parameters

2.Set values to the parameters available in PreparedStatement object as per the applicationrequirement:

To set values to the PreparedStatement object,we have to the following method from PreparedStatement

```
public void setXXX(int parameter_Numb,xxx value)
```

where xxx may be byte,short,int....

Eg:

```
pst.setInt(1,111);
pst.setString(2,"Laddu");
pst.setFloat(3,30000.0f);
```

When JVM encounter the above instructions then JVM will set the specified values to the respective parameters in PreparedStatement object,where these values are reflected to Query plan parameters automatically through the Jdbc driver and connection.

DURGASOFT

3.Make Database engine to pickup the values from Query plan and perform the respective operationas per the generalised SQL query which we provided

If the generalised SQL query belongs to selection group,then we have to use the following method.

public ResultSet executeQuery() throws Exception

If the generalised SQL query belongs to updation group,then we have to use the following method

public int executeUpdate() throws SQLException

Eg: int rowCount=pst.executeUpdate();

```
20. package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp31 {

  public static void main(String[] args)throws Exception {
    Class.forName("com.mysql.jdbc.Driver");
    Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
    PreparedStatement pst=con.prepareStatement("insert into emp1 values(?,?,?,?)");
    BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
    while(true){
      System.out.print("Employee Number   :");
      int eno=Integer.parseInt(br.readLine());
      System.out.print("Employee Name     :");
      String ename=br.readLine();
      System.out.print("Employee Salary   :");
      float esal=Float.parseFloat(br.readLine());
      System.out.print("Employee Address  :");
      String eaddr=br.readLine();

      pst.setInt(1, eno);
      pst.setString(2, ename);
      pst.setFloat(3, esal);
      pst.setString(4, eaddr);

      pst.executeUpdate();
      System.out.println("Employee Inserted Successfully");
      System.out.print("Onemore Employee[yes/no]  :");
```


DURGASOFT

```
            String option=br.readLine();
            if(option.equals("no")){
               break;
            }
         }
         con.close();
      }}
```

21.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp32 {

   public static void main(String[] args)throws Exception {
      Class.forName("com.mysql.jdbc.Driver");
      Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
      PreparedStatement pst=con.prepareStatement("update emp1 set esal=esal+? where
esal<?");
      pst.setInt(1, 500);
      pst.setFloat(2, 10000.0f);
      int rowCount=pst.executeUpdate();
      System.out.println("Records Updated    :"+rowCount);
      con.close();
   }

}
```

22.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class JdbcApp33 {
   public static void main(String[] args) throws Exception{
      Class.forName("com.mysql.jdbc.Driver");
      Connection
con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb","root","root");
      PreparedStatement pst=con.prepareStatement("select * from emp1 where esal<?");
      pst.setFloat(1, 10000.0f);
      ResultSet rs=pst.executeQuery();
```


DURGASOFT

```
        System.out.println("ENO    ENAME    ESAL    EADDR");
        System.out.println("------------------------------------");
        while(rs.next()){
          System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "+rs.getFloat(3)+"
"+rs.getString(4));
        }
        con.close();
    }


}
```

23.

```
package com.durgasoft;
 import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Properties;

public class JdbcApp35 {
   public static void main(String[] args) throws Exception{
      Class.forName("com.mysql.jdbc.Driver");
      Properties p=new Properties();
      p.setProperty("user", "root");
      p.setProperty("password", "root");
      Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb",p);
      PreparedStatement pst=con.prepareStatement("insert into student values(?,?,?,?)");
      pst.setString(1, "S-111");
      pst.setString(2, "Durga");
      pst.setString(3, "Java");
      Date  d=Date.valueOf("2015-01-25");
      pst.setDate(4, d);
      pst.executeUpdate();

      pst.setString(1, "S-222");
      pst.setString(2, "Anil");
      pst.setString(3, "Oracle");
      java.util.Date date=new java.util.Date();
      int val=date.getDate();
      java.sql.Date dt=new java.sql.Date(val);
      pst.setDate(4, dt);
      pst.executeUpdate();
      con.close();
```

DURGASOFT

```
    }

  }
```

Batch Updations:

In general in Jdbc applications, it is required to provide number of SQL queries according to application requirement.

With the above, if we execute the Jdbc application then JVM will send all the SQL queries to the database in a sequential manner.

If we use the above convention to execute SQL queries in Jdbc application we have to spend a lot of time only to carry or transfer SQL queries from Java application to database, this approach will reduce the performance of Jdbc application.
In the above context, to improve the performance of the Jdbc applications we have to use Batch updations.

In batch updations, we will gather or collect all the updation group SQL queries as a single unit called as Batch and we will send batch of updation group SQL queries at a time from Java application to database.

At database, Database Engine may execute all the SQL queries and generate respective row count values in the form of an array to Java application.

To add an SQL query to batch we have to use the following method from Statement.

public void addBatch(String query)

To send batch of updation group SQL queries at a time from Java application to database and to make the Database Engine to execute all the batch of updation group SQL queries we have to use the following method from Statement.

public int[] executeBatch()

Where int[] will represent all the row count values generated from the updation group SQL queries.

Batch updations with Statement:

```
24.package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;
```


DURGASOFT

```
public class JdbcApp30 {
   public static void main(String[] args)throws Exception {
      Class.forName("oracle.jdbc.OracleDriver");
          Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
     "system", "durga");
      Statement st=con.createStatement();
      st.addBatch("insert into emp1 values(666,'FFF',9000,'Hyd')");
      st.addBatch("update emp1 set esal=esal-500 where esal<10000");
      st.addBatch("delete from emp1 where eno=555");
     // st.addBatch("select * from emp1");--> java.sql.BatchUpdateException
      int[] rowCounts=st.executeBatch();
      for(int i=0;i<rowCounts.length;i++){
         System.out.println("Records Manipulated    :"+rowCounts[i]);
      }
      st.close();
      con.close();
   }

}
```

Batch Updations with PreparedStatement:

```
25.
 package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;

public class JdbcApp34 {
   public static void main(String[] args)throws Exception {
      Class.forName("com.mysql.jdbc.Driver");
      Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/durgadb",
"root","root");
      PreparedStatement pst=con.prepareStatement("insert into emp1 values(?,?,?,?)");
      pst.setInt(1,666);
      pst.setString(2, "FFF");
      pst.setFloat(3, 6000);
      pst.setString(4, "Hyd");
      pst.addBatch();

      pst.setInt(1,777);
      pst.setString(2, "GGG");
      pst.setFloat(3, 7000);
      pst.setString(4, "Hyd");
      pst.addBatch();
```


DURGASOFT

```
        pst.setInt(1,888);
        pst.setString(2, "HHH");
        pst.setFloat(3, 8000);
        pst.setString(4, "Hyd");
        pst.addBatch();
        int[] rowCounts=pst.executeBatch();
        for(int i=0;i<rowCounts.length;i++){
          System.out.println("Records Manipulated    :"+rowCounts[i]);
        }
        con.close();

      }

    }
```

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.

Transaction Management:-

Transaction:- Transaction is an unit of work performed by the front end application on back end system.
1. Deposit some amount in an account.
2. Withdraw some amount from an account.
3. Transfer some amount from one account to another account.

In database applications, every transaction must satisfy the following 4 properties.
1. Atomicity
2. Consistency
3. Isolation
4. Durability

1. Atomicity:-
In general we are able to perform multiple number of operations in a particular transaction, where performing all the operations or performing none of the operations is called as Atomicity property. If we perform all the operations successfully in a transaction then the state of the transaction should be success.

If we perform none of the operations in a transaction then state of the transaction should be failure.

Note: While performing operations in a transaction, if we encounter a problem with any operation then we should perform rollback operation over all the operations which are available in the transactions.

DURGASOFT

2. Consistency:
In database applications, before the transaction and after the transaction the state of the database must be stable is called as Consistency property.

To achieve consistency we will perform some checkings called as Consistency Checkings. Consistency checkings will check correctness of the results after the transactions.

3. Isolation:
In database applications, if we perform more than one transaction on a single data item then that transactions are called as Concurrent Transactions. In concurrent transactions, one transaction execution should not be effect to the another transaction. This nature of the transaction is called as Isolation.
While performing concurrent transactions, there may be a chance to get concurrency problems, to resolve these problems we will use Isolation Levels in database application.

4. Durability:
In database applications, after committing the transaction if we have any catastrophic failures like power failure, operating system crashing and so on then we have to preserve the modifications which we performed on the database during respective transaction. This nature of the transaction is called as Durability.

In Jdbc applications, when we establish connection then automatically that connection will have a default mode i.e. auto commit mode. In case of auto commit mode, when we submit SQL query to the connection, where connection will carry 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 permanently.

The above auto commit nature of connection may not satisfy the transaction's atomicity property. To preserve transaction atomicity property in Jdbc applications we have to change connection's auto commit mode.

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.

Ex: 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

DURGASOFT

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.

26.

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class JdbcApp40 {

    public static void main(String[] args) {
            Connection con=null;
            try {
                    Class.forName("oracle.jdbc.OracleDriver");
                    con=DriverManager.getConnection("jdbc:oracle:thin:
                    @localhost:1521:xe","system", "durga");
                    con.setAutoCommit(false);
                    Statement st=con.createStatement();
                    st.executeUpdate("insert into student values(111,'AAA',78)");
                    st.executeUpdate("insert into student values(222,'BBB',87)");
                    st.executeUpdate("insert into student values(333,'CCC',96)");
                    con.commit();
                    System.out.println("Transaction Success");
            } catch (Exception e) {
                    try{
                            con.rollback();
                            System.out.println("Transaction Failure");
                    }catch(Exception e1){
                            e1.printStackTrace();
                    }
                    //e.printStackTrace();
            }
    }

}
```

DURGASOFT

27.

```
package com.durgasoft;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Statement;

public class JdbcApp41 {

    public static void main(String[] args) {
        Connection oracle_conn=null;
        Connection mysql_conn=null;
        try {
            Class.forName("oracle.jdbc.OracleDriver");
            Class.forName("com.mysql.jdbc.Driver");
            oracle_conn=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
                mysql_conn=DriverManager.getConnection
            ("jdbc:mysql://localhost:3306/durgadb","root","root");

            oracle_conn.setAutoCommit(false);
            mysql_conn.setAutoCommit(false);

            Statement oracle_st=oracle_conn.createStatement();
            Statement mysql_st=mysql_conn.createStatement();

            BufferedReader br=new BufferedReader
            (new InputStreamReader(System.in));
            System.out.print("Source Account     :");
            String source_Account=br.readLine();
            System.out.print("Target Account     :");
            String target_Account=br.readLine();
            System.out.print("Amount To Transfer  :");
            int trans_Amt=Integer.parseInt(br.readLine());

            int oracle_rowCount=oracle_st.executeUpdate
            ("update account set balance=balance-"+trans_Amt+" where
            accNo='"+source_Account+"'");
            int mysql_rowCount=mysql_st.executeUpdate("update account set
            balance=balance+"+trans_Amt+" where accNo='"+target_Account+"'");
            if((oracle_rowCount==1) && (mysql_rowCount==1)){
                oracle_conn.commit();
                mysql_conn.commit();
                System.out.println(trans_Amt+" Transfered Successfully from
                "+source_Account+" to "+target_Account);
                System.out.println("Transaction Success");
```


DURGASOFT

```
                }else{
                        oracle_conn.rollback();
                        mysql_conn.rollback();
                        System.out.println("Transaction Failure");
                }
                } catch (Exception e) {
                try {
                        oracle_conn.rollback();
                        mysql_conn.rollback();
                        System.out.println("Transaction Failure");
                } catch (Exception e2) {
                        e2.printStackTrace();
                }
                }

        }

    }
```

Java.sql.SavePoint:-

One of the features introduced in jdbc3.0 version. And it is a interface Savepoint is a intermediate point and makes it possible to rollback the transaction upto the save point instead of roll backing the entire transaction.

To represent Savepoint Jdbc has provided a predefined interface java.sql.Savepoint.
To set a Savepoint we have to use the following method form Connection.

public Savepoint setSavepoint()

To perform rollback operation on set of instructions executive w.r.t a particular Savepoint we have to use the following method from Connection.

public void rollback(Savepoint sp)

To release Savepoint we will use the following method form Connection.

public void releaseSavepoint(Savepoint sp)

Note: In Jdbc applications, Savepoint concept could be supported by Type-4 Driver provided by Oracle, which could not supported by Type-1 Driver provided by Sun Microsystems.

Note: Type-4 Driver is able to support Savepoint up to setSavepoint() method and rollback(_) method, not releaseSavepoint(_) method.

DURGASOFT

```
28.
package com.durgasoft;
 import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.Savepoint;
import java.sql.Statement;

public class JdbcApp42 {

    public static void main(String[] args) {
    Connection con=null;
    try {
            Class.forName("oracle.jdbc.OracleDriver");
            con=DriverManager.getConnection
            ("jdbc:oracle:thin:@localhost:1521:xe","system", "durga");
            con.setAutoCommit(false);
            Statement st=con.createStatement();
            st.executeUpdate("insert into student values(111,'AAA',76)");
            Savepoint sp=con.setSavepoint();
            st.executeUpdate("insert into student values(222,'BBB',88)");
            con.rollback(sp);
            st.executeUpdate("insert into student values(333,'CCC',99)");
            con.commit();
            System.out.println("Transaction Success");
            } catch (Exception e) {
            try {
                    con.rollback();
                    System.out.println("Transaction Failure");
            } catch (Exception e2) {
                    e2.printStackTrace();
            }
    }

    }

}
```

DURGASOFT

Stored Procedures And Functions:

What is the difference between Stored procedures and functions?
Ans:Stored procedure is a block of instructions defined at database to represent a particular action. Stored procedures will not use return statement to return a value.

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)
```

Stored function is a block of instructions defined at database to represent a particular action.Stored functions will use return statement to return a value.

```
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 Jdbc applications, to access stored procedures and functions defined at database from Java application then we have to use CallableStatement object.

To represent CallableStatement object Jdbc API has provided the interface in the form of java.sql. CallableStatement.

If we want to use CallableStatement object in Jdbc applications then we have to use the following steps.

Step 1: Get CallableStatement object.

To create CallableStatement object in Jdbc applications we have to use the following method from Connection.

```
public CallableStatement prepareCall(String pro_cal)throws SQLException

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


DURGASOFT

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.

Note: In case of stored procedures and functions we are able to pass the parameters in the following 3 ways.

1. IN Type Parameter:

This parameter will get the value from procedure call or function call and make available to the procedure body or function body.

Syntax: var_name IN data_type

2. OUT Type Parameter:

This parameter will get the value from procedure body or function body and send that value to the respective procedure call or function call.

Syntax: var_name OUT data_type

3. INOUT Type Parameter:

This parameter is acting as both IN type and OUT type parameters.

Syntax: var_name INOUT data_type

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

To set values to IN type parameters we have to use the following method.

public void setXxx(int param_position, xxx value)

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

Ex: cst.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 datatype.

To register OUT type parameter we will use the following method.

DURGASOFT

public void registerOutParameter(int param_position, int data_type)

Where data_type may be the constants from Types class like BYTE, SHORT, INTEGER, FLOAT and so on.

Ex: cst.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.

To achieve this we have to use The following method.

public void execute()throws SQLException

Ex: cst.execute();

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 may be byte, short, int and so on.

EX: float sal=cst.getFloat(2);

Ex:- execution of procedures

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

```
29.
 import java.sql.*;
public class JdbcApp34
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
CallableStatement cst=con.prepareCall("{call getSal(?,?)}");
```


DURGASOFT

```
cst.setInt(1,101);
cst.registerOutParameter(2, Types.FLOAT);
cst.execute();
System.out.println("Salary....."+cst.getFloat(2));
con.close();
}
}
```

Ex:- 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 eno=id1;
select esal into sal2 from emp where eno=id2;
return (sal1+sal2)/2;
END getAvg;
/
```

Ex:-

```
import java.sql.*;
public class Test
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
CallableStatement cst=con.prepareCall("{?=call getAvg(?,?)}");
cst.setInt(2,888);
cst.setInt(3,6666);
cst.registerOutParameter(1, Types.FLOAT);
cst.execute();
System.out.println("Average Salary....."+cst.getFloat(1));
con.close();
}
}

/*
 create or replace procedure getEmps(sal IN number, emps OUT SYS_REFCURSOR)
 AS
 BEGIN
     open emps for
             select * from emp1 where esal<sal;
 END getEmps;
```


DURGASOFT

```
/
*/
package com.durgasoft;

import java.io.FileInputStream;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.Properties;

import oracle.jdbc.internal.OracleTypes;

    public class JdbcApp38 {

    public static void main(String[] args)throws Exception {
            FileInputStream fis=new FileInputStream("db.properties");
            Properties p=new Properties();
            p.load(fis);
            String driver_class=p.getProperty("driver_class");
            String driver_url=p.getProperty("driver_url");
            Class.forName(driver_class);
            Connection con=DriverManager.getConnection(driver_url,p);
            CallableStatement cst=con.prepareCall("{call getEmps(?,?)}");
            cst.setFloat(1, 10000);
            cst.registerOutParameter(2, OracleTypes.CURSOR);
            cst.execute();
            Object obj=cst.getObject(2);
            ResultSet rs=(ResultSet)obj;
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("----------------------------");
            while(rs.next()){

    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
            }
            con.close();
    }

}
```

db.properties

```
driver_class=oracle.jdbc.OracleDriver
driver_url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=durga
```


DURGASOFT

```
/*
 create or replace function getEmployees(no1 IN number,no2 IN number) return SYS_REFCURSOR
 AS
 employees SYS_REFCURSOR;
 BEGIN
 open employees for
     select * from emp1 where eno>=no1 and eno<=no2;
 return employees;
 END getEmployees;
 /
 */
package com.durgasoft;

import java.io.FileInputStream;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.util.Properties;

import oracle.jdbc.internal.OracleTypes;

public class JdbcApp39 {

    public static void main(String[] args)throws Exception {
            FileInputStream fis=new FileInputStream("db.properties");
            Properties p=new Properties();
            p.load(fis);
            String driver_class=p.getProperty("driver_class");
            String driver_url=p.getProperty("driver_url");
            Class.forName(driver_class);
            Connection con=DriverManager.getConnection(driver_url, p);
            CallableStatement cst=con.prepareCall("{?=call getEmployees(?,?)}");
            cst.setInt(2, 111);
            cst.setInt(3, 555);
            cst.registerOutParameter(1, OracleTypes.CURSOR);
            cst.execute();
            ResultSet rs=(ResultSet)cst.getObject(1);
            System.out.println("ENO\tENAME\tESAL\tEADDR");
            System.out.println("------------------------");
            while(rs.next()){

    System.out.println(rs.getInt(1)+"\t"+rs.getString(2)+"\t"+rs.getFloat(3)+"\t"+rs.getString(4));
            }
            con.close();
    }
}
```


DURGASOFT

db.properties

```
driver_class=oracle.jdbc.OracleDriver
driver_url=jdbc:oracle:thin:@localhost:1521:xe
user=system
password=durga
```

Connection Pooling:

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.

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.

DURGASOFT

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)
```

```
Ex: 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()

Ex: 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()

Ex: Connection con=pc.getConnection();

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

Ex: Statement st=con.createStatement();

```
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();
```


DURGASOFT

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

Note: The above approach of implementing Connection pooling is suggestible up to standalone applications, it is not suggestible in enterprise applications. If we want to implement Connection pooling in enterprise applications we have to use the underlying application server provided Jdbc middleware server.

BLOB and CLOB:

BLOB (Binary Large Object):

Up to now in Jdbc applications, we are able to interact with the database in order to insert a record, retrieve a record and so on with the varchar data or number data and so on.

As per the application requirement if we want to insert an image or a document in the database table then Oracle provided datatypes numbers, varchar are not sufficient, we have to use BLOB and CLOB datatypes provided by Oracle.

The main purpose of the BLOB and CLOB datatypes is to represent large volumes of binary data and large volumes of character data in a database table.

To insert large volumes of binary data (an image) on to the database table we have to use the following steps.

Step 1: Prepare a table at database with blob data type.
Ex: create table emp_details(eno number, image blob);

Step 2: Represent an image file in the form of File class object.
Ex: File f=File("Desert.jpg");

DURGASOFT

Step 3: Get File class object content in the form of FileInputStream.
Ex: FileInputStream fis=new FileInputStream(f);

Step 4: Create preparedStatement object with insert SQL query format.

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

Step 5: Set BinaryStream to the blob type positional parameter in PreparedStatement.

To set a BinaryStream with the blob type positional parameter we have to use thefollowing method from PreparedStatement.

public void setBinaryStream(int param_index, InputStream is, int length);

Ex: pst.setBinaryStream(2, fis, (int)f.length());

Step 6: Execute PreparedStatement.

Ex: pst.executeUpdate();

```
import java.sql.*;
import java.io.*;
public class BLOBDemo
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","manager");
File f=new File("Desert.jpeg");
FileInputStream fis=new FileInputStream(f);
PreparedStatement pst=con.prepareStatement("insert into employee values(?,?)");
pst.setInt(1,105);
pst.setBinaryStream(2,fis,(int)f.length());
pst.executeUpdate();
System.out.println("Employee Image inserted Successfully");
con.close();
}
}
```

Steps to retrieve blob data from database table to Jdbc application:

Step 1: Prepare ResultSet object with blob data.

Ex: ResultSet rs=st.executeQuery("select * from emp_details");

Step 2: Read normal data from ResultSet object.

DURGASOFT

Ex: rs.next();

int eno=rs.getInt(1);

Step 3: Get BinaryStream from blob datatype available at ResultSet object

To get a BinaryStream from blob type parameter available at ResultSet object we have to use the following method.

public InputSteram getBinaryStream(int param_index)

Ex: InputSteram is=rs.getBinaryStream(2);

Step 4: Prepare the target resource to hold up the retrieved blob data by using FileOutputStream.

Ex: FileOutputStream fos=new FileOutputStream("myimage.jpeg");

Step 5: Read bit by bit from InputStream and write the same bit by bit on FileOutputStream to store the retrieved data on target file.

Ex: int i=read();

```
while(i!=-1) {
fos.write(i);
i=is.read();
}
```

```
import java.sql.*;
import java.io.*;
public class BLOBDemo1
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
);
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from employee");
rs.next();
System.out.println("Employee Id....."+rs.getInt(1));
InputStream is=rs.getBinaryStream(2);
FileOutputStream fos=new FileOutputStream("myimage.jpg");
int i=is.read();
```


DURGASOFT

```
while (i != -1)
{
fos.write(i);
i=is.read();
}
System.out.println("Image created Successfully with the name
myimage.jpeg");
fos.close();
con.close();
}
}
```

CLOB (Character Large Object):

clob is a datatype provided by Oracle, it can be used to represent large values of character data like documents, pdf files and so on.

If we want to perform operations with clob datatype then we have to use the same steps what we have used with blob datatype, but we need to provide the following replacements.

```
import java.sql.*;
import java.io.*;
public class CLOBDemo
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
);
PreparedStatement pst=con.prepareStatement("insert into webinfo
values(?,?)");
pst.setString(1, "app");
File f=new File("web.xml");
FileReader fr=new FileReader(f);
pst.setCharacterStream(2, fr, (int)f.length());
pst.executeUpdate();
System.out.println("web application stored in database successfully");
fr.close();
con.close();
}
}
```

DURGASOFT

Ex:-

```
import java.sql.*;
import java.io.*;
public class CLOBDemo1
{
public static void main(String[] args) throws Exception
{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","durga"
);
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from webinfo");
rs.next();
System.out.println("Application name is....."+rs.getString(1));
FileWriter fw=new FileWriter("myweb.xml");
Reader r=rs.getCharacterStream(2);
int i=r.read();
while (i != -1)
{
fw.write(i);
i=r.read();
}
System.out.println("web.xml is retrieved successfully with the name
myweb.xml");
fw.close();
con.close();
}
}
```

DURGASOFT

JDBC INTERVIEW QUESTIONS:

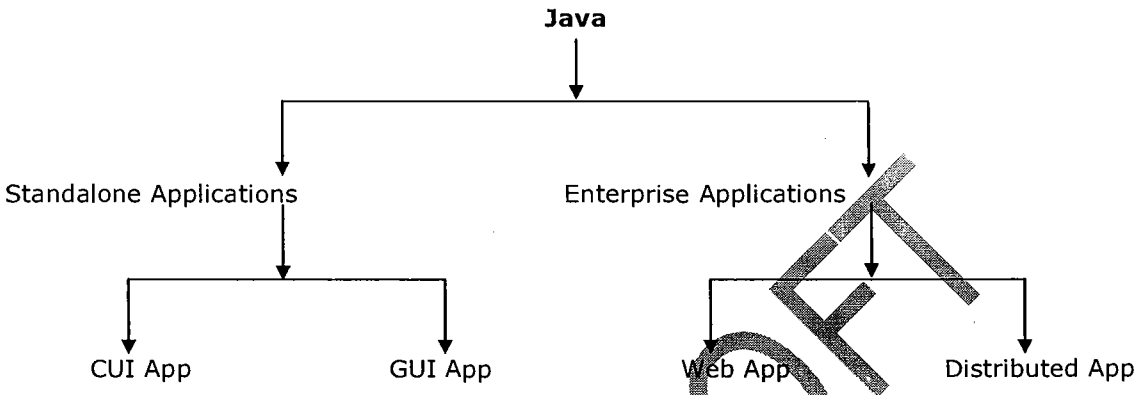
1. What is the difference between Database and Database management system?
2. How a query could be executed when we send a query to Database?
3. What is Driver? How many Drivers are available in JDBC? What are the types?
4. What is JDBC and What are the steps to write a JDBC application?
5. How to load a JDBC driver?
6. How to establish a Database connection between java application and Database?
7. Basically Connection is an interface, how getConnection() will create an object for Connection interface?
8. What is the requirement to use Statement object?
9. How to execute SQL Queries from a java application?
10. What are the differences between executeQuery(...), executeUpdate(...) and execute(...) methods?
11. How to create a table dynamically from a jdbc application?.
12. How to insert records into a table from a JDBC application?

13. How to update a table from a jdbc application?.
14. How to delete records from a table from jdbc application?.
15.What is ment by ResultSet object and How to Fetch the Data from Database?.
16.Ingeneral execute() method can be used to execute selection group SQl queries for getting the data from Database , but execute() return a boolean value true so here how it possible to fetch the data from database?
17.Ingeneral execute() method can be used to execute updatation group SQl queries for updating the data on Database , but execute() return a boolean value false so here how it possible to get the records updated count value(int value)?
18. If we use selection group SQL query to executeUpdate() ,what happened?
19. If we use updatation group SQL query to executeQuery() ,what happened?
20. What is ment by ResultSet and What are the types of ResultSets are available in JDBC application?
21. What is the difference between ScrollSensitive ResultSet and ScrollInsensitive ResultSets?
22. What is the default ResultSet type in JDBC application and How it is possible to create a specific type of ResultSet object?
23. How to iterate the data from Scrollable ResultSet objuect in both forward and backword direction?
24. How to generate ScrollSensitive Result Set and how to reflect the later updations from database automatically to the ResultSet object?
25. How to insert records into Database throws Updatable ResultSet?
26. How to perform updations on Database throws Updatable ResultSet?
27. What is meant by ResultSetMetaData ?How to get The ResultSet metadata of a ResultSet object?
28. How to display the data with the respective field names
29. What are the differences between Statement and PreparedStatement? (or) Tell me the situations where we should go for PreparedStatement over Statement object.
30. Hhow to insert number of records into a table through Prepared Statement object.
31. How to update the database through PreparedStatement object.
32. How to fetch the data from database through PreparedStatement object.
33. What is meant by Transaction? How it is possible to maintain Transactions in JDBC applications?
34. What is meant by SavePoint?How to use Savepoints in JDBC applications?

DURGASOFT

Introduction:

In general by using Java technology we are able to design following 2 types of applications.



1. Standalone Applications:

These are the java applications, which will be designed without using Client-Server Architecture.

There are 2 types of Standalone applications.

1. CUI Applications
2. GUI Applications

Q: What are the differences between CUI Applications and GUI Applications?

Ans: CUI Applications are the standalone applications, which will be designed in such a way to take input and to provide output by using Command prompt, where Command prompt is acting as user interface and it able to support character data. So that the java application which will design on the basis of command prompt i.e. character user interface is called as **CUI application**.

GUI Applications are the standalone applications, which will be designed in such a way to take input and to provide output by using a collection of Graphic component, where the collection of Graphic components is acting as an user interface and it able to support GUI component so that the java application which will be design on the basis of Graphical user interface is called as **GUI Application**.

DURGASOFT

2. Enterprise Applications:

These are the java applications, which will be designed on the basis of Client-Server Architecture. There are 2 types of Enterprise applications.

1. Web Applications
2. Distributed Applications

Q: What are the differences between Web Applications and Distributed Applications?

Ans: 1. Web Application is the server side application, it will be designed without disturbing its application logic over multiple number of JVM's.

Distributed application is the server side application it will be designed by disturbing application logic over multiple number of JVM's.

2. In general web applications will be designed by using a set of server side technologies called as Web technologies like CGI, Servlets, JSP's and so on.

Distributed applications will be designed by using a set of technologies called as Distributed technologies like Socket programming, RMZ, EJB's, Webservices and so on.

3. The main purpose of web applications is to generate dynamic response from server machine, but the main purpose of distributed applications is to establish distributed communication between local machine and remote machine in order to access the remote services.

4. In general web applications will provide services for web clients, but distributed applications will provide services for any type of clients.

5. In general web applications will be executed by using both web servers and application servers, but distributed applications will be executed by using only application servers.

1. Web Applications:

Web Application is a collection of web components, it will be executed by using web containers like Servlet Container to execute servlets, Jsp Container to execute JSP's and so on.

2. Distributed Applications:

Distributed Application is a collection of distributed components, it will be executed by using the distributed containers like EJB's Container to execute EJB's.

DURGASOFT

At the beginning stages of the computer we have Client-Server architecture, where the purpose of server is to hold up some resources and to share that resources to all the clients as per the client request.

In the above context, when we send a request from client to server for a particular resource then server will identify requested resource, pick up the content and send that content as response to client without performing any particular action at server. In this case the response which was generated by server is called as static response.

As per the application requirements we need to generate dynamic response from server, for this we need to execute the application at server called as Web Application. To design web applications at server side we need a set of server side technologies called as **Web Technologies.**

To design web applications we will use web technologies like CGI, Servlets, Jsp's, Perl, PHP and so on. Therefore, the main purpose of servlets and jsp's is to design web applications at server in order to generate dynamic response from server.

Q: To design web applications at server we have already CGI Technology then what is the requirement to go for servlets?

Ans: CGI is the server side technology designed on the basis of C technology and Scripting. C technology is the process based technology, it will make CGI technology as process based technology.

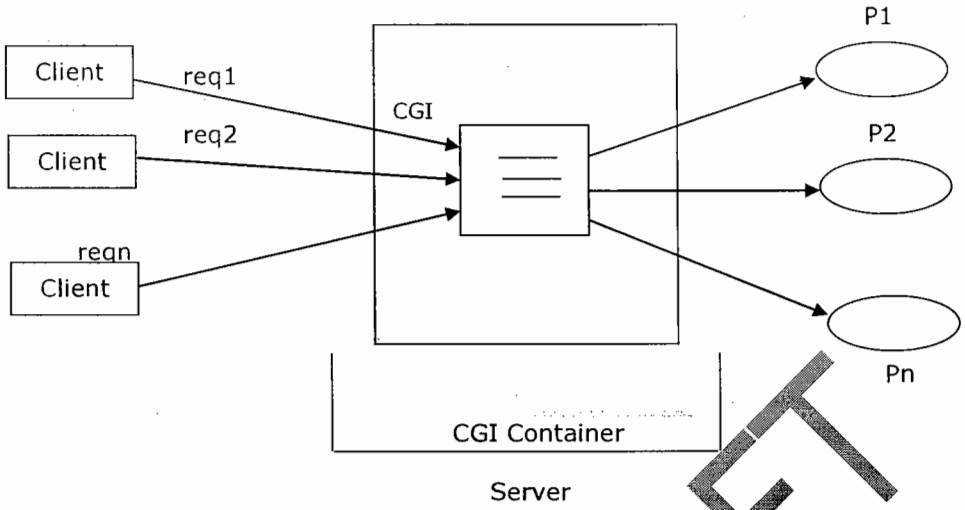
If we deploy any CGI application at server side then container will create a separate process for each and every request. If we increase number of requests automatically CGI container will generate number of processes at server.

Basically process is a heavy weight component, to handle single process system has to consume a lot of system memory and execution time.

Due to the above reason to handle the multiple number of processes at a time server machine may get burden, due to this reason server may generate delayed responses to the client request, it will reduce the performance of server side application.

In the above context, to increase the performance of the server side application we have to use an alternative server side technology i.e. Servlets.

DURGASOFT

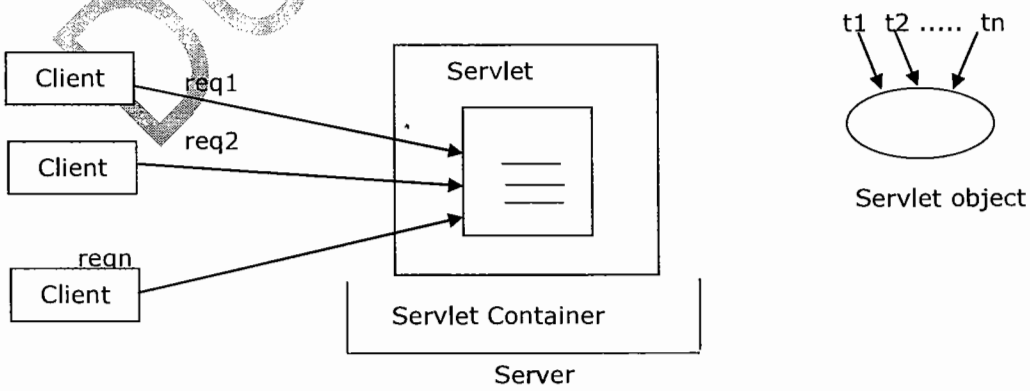


Servlet is the server side technology designed on the basis of java technology. Java technology is a Thread based technology, it will make servlets as Thread based technologies.

If we deploy any servlet application at server then for every client request servlet container will generate a separate thread on the respective servlet object.

In the above context, if we increase number of requests even container will create number of threads instead of processes.

When compared to process thread is light weight, to handle multiple number of requests i.e. thread server machine may not get burden, it may provide quick responses for the client request, it may increase the performance of server side application.



DURGASOFT

Q: What are the differences between servlets and jsps?

Ans: 1. If we want to design web applications by using Servlets we must require very good java stuff.

To design web applications by using Jsp technology it is not required to have java knowledge, it is sufficient to have presentation skills.

Note: The main intention to introduce Jsp technology is to reduce java code as much as possible in web applications.

2. In web applications, we will prefer to use Servlets to pick up the request and process the request.

But we will prefer to use Jsp's to generate dynamic response to the client with very good look and feel.

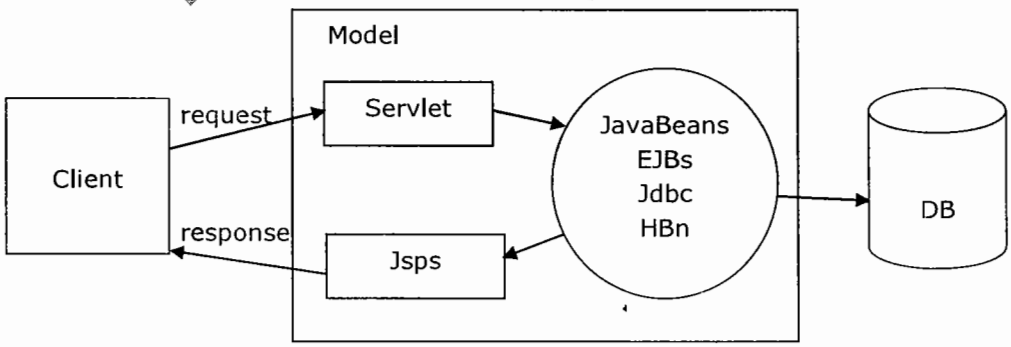
Note: In web applications, we will utilize Servlets to provide controller logic, integration logic, validation logic, implementing business logic and so on. But we will prefer to use Jsp technology only to provide presentation part.

3. In case of Servlets, we are unable to separate both presentation logic and business logic, but in case of Jsp's we are able to separate presentation logic and business logic because in Jsp pages we are able to use html tags to prepare presentation logic and we are able to use Jsp tags to prepare business logic.

4. If we perform any modifications on the existed Servlets then we have to perform recompilation and reloading on to the server explicitly.

If we perform any modifications on the existed Jsp's then it is not required to perform recompilation and reloading because Jsp pages are auto compiled and auto loaded.

5. If we want to design any web application on the basis of MVC architecture then we have to use a servlet as controller and a set of Jsp pages as view part.



DURGASOFT

Server

Note 1: Struts is a web application framework designed on the basis of MVC architecture, where we have to use ActionServlet as controller and we have to use a set of Jsp pages as view part.

Note 2: JSF(Java Server Faces) is a web application framework, where we have to use FacesServlet as a controller and a set of Jsp pages as view part.

Enterprise:

Enterprise is a business organization, a group of organizations running under a single label.

Enterprise Application:

It is a software application, which will be designed for a particular enterprise in order to simplify the internal business processing.

To design enterprise applications we have to use the following 3 layers.

1. User Interface Layer
2. Business Processing Layer
3. Data Storage and Access Layer

1. User Interface Layer:

User Interface Layer is the top most layer in enterprise application, it will provide starting point for the customers to interact with enterprise application.

The main purpose of User Interface Layer in enterprise applications is

1. To improve look and feel for the enterprise applications.
2. To accept user details in order to execute server side application.
3. To provide very good environment for client side validations with Javascript functions.
4. To specify different types of requests at client browser like get, post, head and so on.

To prepare User Interface Layer in enterprise application we will use a separate logic is called as **Presentation Logic.**

DURGASOFT

To prepare presentation logic in enterprise applications we have to use technologies like AWT, SWING, HTML, JSP, Velocity and so on.

2. Business Processing Layer:

Business Processing Layer is the heart of the enterprise application, it will provide very good environment to define and execute all the business rules and regulations which are required by the client.

To prepare Business Processing Layer in enterprise application we will use a separate logic is called as **Business Logic.**

In enterprise application development,to prepare Business logic we have to use technologies like Servlets, Jsp's, JavaBeans, EJBs and so on.

3. Data Storage and Access Layer:

This layer is the bottom most layer in enterprise applications.

The main purpose of this layer is to provide data persistency in enterprise application.

Data Storage and Access Layer will provide very good environment to provide the basic database operations(CRUD) as per the enterprise application requirement.

To prepare Data Storage and Access Layer we will use a separate logic called as **Persistence Logic.**

In enterprise application development,to prepare persistence logic we have to use technologies like Jdbc, EJB-Entity Beans Hibernate, JPA and so on.

To design enterprise application we need to define the degree of enterprise application, for this we have to use system architectures.

1-Tier Architecture

2-Tier Architecture

---

n-Tier Architecture

DURGASOFT

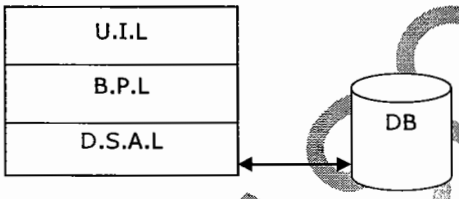
1-Tier Architecture:

To design enterprise application if we use 1-tier architecture then we have to provide User Interface Layer, Business Processing Layer and Data Storage and Access Layer within a single machine.

1-Tier architecture is highly recommended for standalone applications.

In 1-Tier architecture, a single machine we have to use to accommodate the complete application so that a single machine resources may not be sufficient to execute applications, it may reduce the performance of enterprise application.

1-Tier architecture will not provide any environment to handle multiple number requests, it is able to provide less sharability.



1-Tier Architecture

2-Tier Architecture:

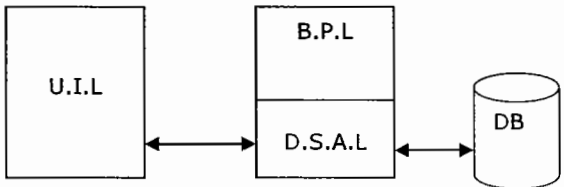
To design any enterprise application it is minimum to use 2-Tier architecture. The best example for 2-Tier architecture is Client-Server architecture.

2-Tier architecture will allow to design and execute the application in 2 layers of machines.

In case of 2-Tier architecture, tier-1 will manage User Interface Layer and tier-2 will manage Business Processing Layer and Data Storage and Access Layers.

2-Tier architecture will provide loosely coupled design when compared with 1-Tier architecture.

2-Tier architecture will provide very good environment to handle multiple number of requests and it is able to improve sharability.



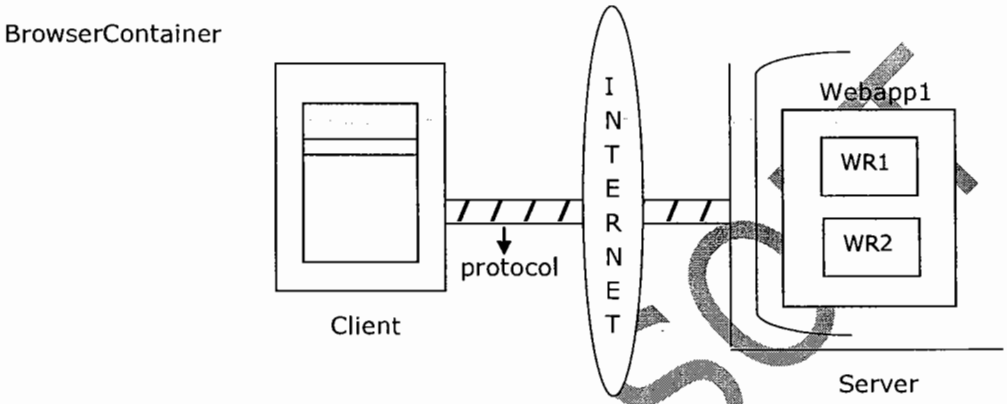
Client-Tier/Tier-1 Server-Tier/Tier-2

DURGASOFT

2-Tier Architecture

Note: As part of the enterprise application development always it is suggestible to use Tiered architectures because it will improve sharability, able to provide more loosely coupled design and so on, but we should not increase number of tiers in enterprise applications without having the requirement otherwise maintenance problems will be increased.

Client-Server Architecture:



From the above representation, there are three major components in client-server architecture.

1. Client
2. Protocol
3. Server

1. Client:

The main purpose of the Client in client-server architecture is to send request to the server and to set the responses from server.

To send request and to set response from server we need to use a tool at client machine called as **Browser.** In client-server application browser is acting as client.

To access a particular resource available at server from client browser we need to specify a particular string at browser address bar called **URI.**

There are two types of URI's

1. URL 2. URN

Q: What are the differences between URI, URL and URN?

Ans: URIis a string specification provided at client address bar, it can be used to refer a particular resource available at server machine.

DURGASOFT

URL is a string specification provided at client address bar, it can be used to refer a particular resource available at server machine through its locator.

URN is a string specification, it can be used to refer a particular resource available at server machine through its logical name.

Note 1: In case of servlets, locator is an URL pattern defined in web.xml file.

Note 2: In case of servlets, logical name is a name specified along with <servlet-name> tag in web.xml file.

Note 3: Almostall the servers are able to accept URL kind of request, but almost all the servers are not accept URN kind of request.

If we want to provide URL at client address bar then we have to use the following syntax.

Protocol_Name://Server_IP_Address:Server_Port_No/Application_Context/Resource_Name[Query_String]

Ex: http://121.120.92.98.8080/loginapp/logon?uname=abc&upwd=abc

Here Query_String, i.e. uname=abc&upwd=abc is optional.

Q: What is the difference between IP Address and Port Number?

Ans: IP Address is an unique identification for each and every machine over Network, which will be provided by Network Manager at the time of Network Configuration.

Port Number is an unique identification to each and every process being executed at the same machine and which could be provided by the local operating system.

Q: What is Query String and what is the purpose of Query String in web applications?

Ans: Query String is a collection of name-value pair appended to the URL, which can be used to pass input parameters to the respective server side resource in order to perform the required server side action.

DURGASOFT

2. Protocol:

The main job of the Protocol in client-server architecture is to carry the request data from client to server and to carry the response data from server to client.

Protocol is a set of rules and regulations, which can be used to carry the data from one machine to another machine over the Network.

Ex: TCP/IP, FTP, HTTP, SMTP, ARP, RARP.....

In general in web applications, we will use http protocol to send request from client to server and to set response from server to client.

Q: What is the requirement of http protocol in web applications?

Ans: In web applications, to transfer the data between client and server we require a protocol, it should be

1. A Connectionless Protocol
2. A Stateless Protocol
3. A compatible Protocol to carry hypertext data.

Where Connectionless Protocol is protocol, it should not require a physical connection, but require a logical connection to carry the data.

Where Stateless Protocol is a protocol, which should not remember previous request data at the time of processing the later request.

In general in client server application, request data will be transferred from client to server in the form of hypertext data and the response data will be transferred from server to client in the form of hypertext data so that we require a Compatible Protocol to carry hypertext data between client and server.

Among all the protocols http protocol is able to satisfy all the above requirements so that we will use http protocol in web applications.

Q: How http protocol is able to manage stateless nature?

Ans: In client server applications, when we send a request from client to server protocol will pick up the request and perform the following actions.

1. Protocol will establish a virtual socket connection between client and server as per the server IP address and protocol which we provided in URL.
2. Protocol will prepare the request format with header part and body part, where header part will manage all the request headers(metadata about client) and body part will manage request parameters(the data which was provided by the user at client browser).
3. After preparing request format protocol will carry request format to server through the virtual socket connection.

Upon receiving request from protocol server will identify the request resource, execute generate dynamic response and dispatch that dynamic response to client. When server dispatch the dynamic response to the client protocol will pick up the response and perform the following actions.

1. Protocol will prepare response format with header part and body part, where header part will manage response headers(metadata about the dynamic response) and body part will manage the actual dynamic response.

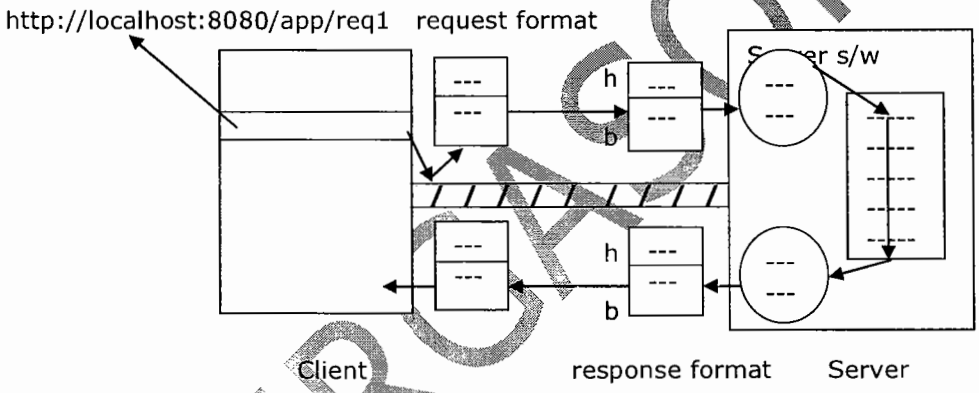
DURGASOFT

2. After setting response format protocol will carry response format to client.
3. When the dynamic response reached to client protocol will terminate the virtual socket connection, with this protocol will eliminate the present request data from its memory.

In the above context, the present request data will be managed by the protocol up to the connections existence, will protocol connection has terminated then protocol will not manage request data.

Due to the above reason http protocol is unable to manage clients previous request data at the time of processing later request. Therefore, http protocol is a stateless protocol.

Note: If we use http protocol, a stateless protocol in our web applications then we are unable to manage clients previous request data, but as per the application requirements we need to manage clients previous request data at the time of processing later request. In this context, to achieve the application requirement we have to use a set of explicit mechanism at server side called as **Session Tracking Mechanisms.**



In web applications, with http protocol we are able to specify different types of requests at client browser. The above flexibility is possible for the http protocol due to the availability of 7 number of http methods called as BIG 7 http methods.

Http protocol has provided the following http methods along with http1.0 version.

1. GET
2. POST
3. HEAD

Http protocol has provided the following http methods as per http1.1 version.

1. OPTIONS
2. PUT
3. TRACE

4. DELETE

Http1.1 version has provided a reserved http method i.e. CONNECT.

Q: What are the differences between GET and POST methods?

or

What are the differences between doGet(_,_) and doPost(_,_) methods?

Ans: 1. GET request type is default re quest type in web applications, but POST request type is not default request type.

2. GET request type should not have body part in the request format, but POST request type should have body part in the request format.

3. If we specify request parameters along with GET request then that request parameters will be transferred to server through request format header part due to the lack of body part.

In general request format header part will have memory limitation so that it is able to carry maximum 256 no. of characters. Therefore, GET request is able to carry less data from client to server

Due to the availability of body part in POST request all the request parameters will be transferred to server through request format body part, here there is no memory limitation in request format body part so that the POST request is able to carry large data from client to server.

4. If we specify request parameters along with GET request then GET request will display all the request parameters at client address bar as query string. Therefore, GET request is able to provide less security for the client data.

If we provide request parameters along with POST request then GET request will not display the request parameters at client address bar. Therefore, POST request is able to provide very good security for the client data.

5. In general in web applications, GET request can be used to get the data from server i.e. download the data from server.

In general in web applications, POST request can be used to post the data to server i.e. upload the data on to the server.

Note: Bookmarks are supported by GET request and which are not supported by POST request.

Q: What is the difference between GET request and HEAD request?

Ans: If we send Get request for a particular resource available at server machine then server will send only the requested resource as a response to client.

DURGASOFT

If we send HEAD request for a particular resource available at server then server will send requested resource as well as the metadata about the requested resource as response.

Note: Internally HEAD request uses GET request to get the requested resource from server.

OPTIONS Request:

The main purpose of the OPTIONS request type is to get the http methods which are supported by the present server.

Note: In general http protocol has provided by 7 http methods conceptually, supporting all the methods or some of the methods or none is completely depending on the server implementation provided by the server providers.

With this convention we are unable to credit how many number of http methods are supported by the present application server, where to credit the http methods which are supported by the present server we have to use OPTIONS request type.

Q: What is the difference between POST request and PUT request?

Ans: Both POST request and PUT request can be used to upload the data on the server machine. To upload the data on server machine if we use POST request then it is not mandatory to specify particular address location along with POST request.

To upload the data on server machine if we use PUT request then it is mandatory to specify server side location along with PUT request.

TRACE Request:

The main purpose of the TRACE request is to get the working status of a particular resource available at the server machine. TRACE request type is able to execute its functionality like echo server.

DELETE Request:

The main purpose of this request type is to delete a particular resource available at server machine.

Note: Almost all the servers may not support PUT, DELETE and request types as per their security constraints. In general almost all the servers are able to support GET and POST request types.

DURGASOFT

Status Codes:

In web applications, the main purpose of the status codes is to give the status of the request processing to the client.

Http1.1 version has provided all the status codes in the form of number representations. As per the web application requirement http1.1 version has provided the following status codes.

1xx --→ 100 to 199 --→ Inforamtional status codes

2xx --→ 200 to 299 --→ Success related status codes

3xx --→ 300 to 399 --→ Redirectional status codes

4xx --→ 400 to 499 --→ Client side error status codes

5xx --→ 500 to 599 --→ Server side error status codes

In general in web applications, when we send a request from client to server, server will identify the requested resource, execute it and generate dynamic response to client.

In the above context, when server dispatches the response protocol will pick up the response and prepare response format, where the dynamic response will be stored in the response format body part.

At the time of processing response format server will provide the respective status code value in the response format header part i.e. with status line field.

When protocol carry the response format to the client then client will pick up the status code value from status line field, prepare itself to get the response from response format body part.

DURGASOFT

3. Server:

The main purpose of the server in client server applications is to pick up the request from client, identify the requested resource, generate the dynamic response and dispatch dynamic response to client.

Note: Servlet is a program available at server machine, it is not capable to pick up the request and dispatch response to client, if server execute servlet program then some dynamic response will be generated.

Examples of Servers: Apache Tomcat, BEA Weblogic, IBM Websphere, Macromedia JRun, SUN Sunone, J2EE, GlassFish and so on.

There are 2 types of servers to execute enterprise applications.

1. Web Servers
2. Application Servers

Q: What are the differences between Web Servers and Application Servers?

Ans: 1.Web Server is a server, which will provide very good environment to execute web applications only. But Application servers will provide very good environment to execute any type of J2EE applications like web applications, distributed applications and so on.

2. In general web servers will not provide all the middle ware services. But application servers will provide all the middle ware services like JND, Jdbc and so on as in-built support.

Application server = Web server + Middleware services

Note: Initially the main intention of web servers is to execute static resources in order to generate static response and the main intention of application servers is to execute dynamic resources in order to generate dynamic response.

If we want to specify a particular machine has server machine then we have to install a particular server software, when we install server software on server machine automatically that server software will be available in the form of the following 2 modules.

1. Main Server
2. Container

Q: What is the difference between Main server and Container?

Ans: When we send a request from client to server then main server will pick up the request from protocol and check whether the request data is in well-formed formator not, if it is not in well-formed format then main server will stop request their itself and generate the respective response to client.

If the request data is in well-formed format then that request will by pass to container, wherecontainer will identify the requested resource, execute it, generate dynamic response and dispatch dynamic response to main server, where main server will bypass response to client through the protocol.

In general containers could be classified into the following 2 ways.

1. As per the technology which we used to design server side component. There are some containers

1. Servlet Container --→ To execute servlets

2. Jsp Container --→ To execute Jsp's

DURGASOFT

3. EJB Container --→ To execute EJB's

Note: All the above specified containers can be used to execute the respective components because the above containers have implemented the respective technology API.

2. As per the containers physical existency there are 3 types of containers.

1. Standalone Container:

It is an integration of main server and container as a single program.

2. Inprocess Container:

It is a container existed inside the main server.

3. Out of process Container:

It is a container existed outside of the main server.

Steps to design first web application(Servlet application):

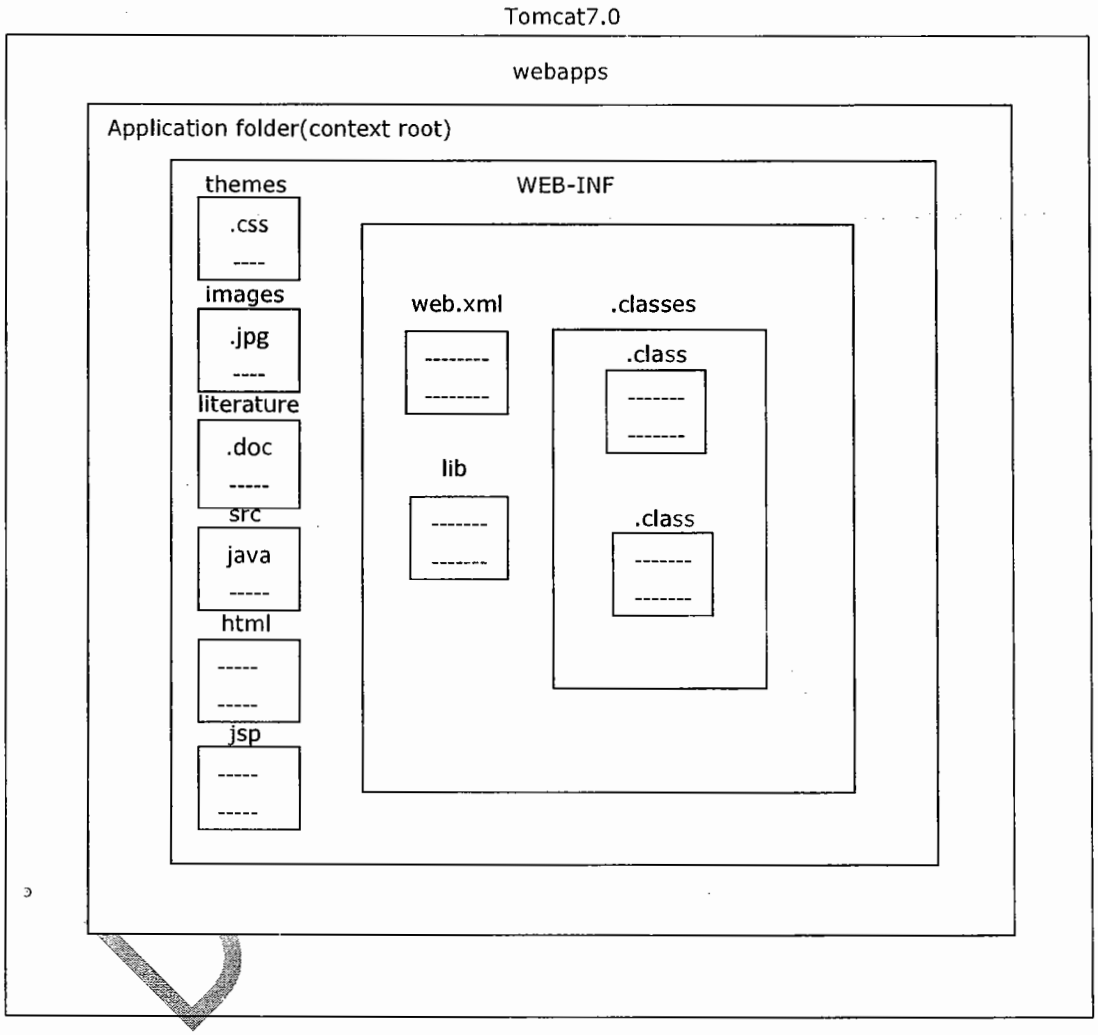
To design first web application we have to use the following steps.

1. Prepare web application directory structure
2. Prepare deployment descriptor
3. Prepare the required web resources like html, servlets, Jsp's and so on
4. Start the server
5. Access the web application

DURGASOFT

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.



When we install Tomcat server automatically Tomcat7.0 folder will be created. In Tomcat7.0 folder, the main purpose of webapps folder is to accommodate all the web applications provided by the developer.

To represent our own web applications in server we have to prepare a separate folder under webapps folder i.e. application folder or context root. Inside the application folder,

1. Themes: To store .css files(cascade style sheet) to generate reports.

2. Images: To store logos of the organizations, background sceneries and so on in the form of .jpg, .jpeg, .gif files.

3. Literature: To store documentations in the form of .doc, .docex and so on.

4. src(Source code): It can be used to store all the source files like Java files.

5. Along with all these folders it is possible to provide some static resources directly like .html files and dynamic resources like .jsp files.

6. WEB-INF folder will include

1. web.xml: This xml file can be used to store the metadata about the web application required by the container to perform a particular action.

2. lib: It is able to manage jar files which are required by the server side components.

3. classes: It will include .class files of servlets, filters, listeners and so on.

The above web application directory structure was divided into the following

1. 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.

Note: In general we will keep servlet .class files under classes folder i.e. private area so that we are unable to access that servlet by using its name directly from client.

To access any servlet we have to define an URL pattern in web.xml file w.r.t this servlet, with the defined URL pattern only client is able to access the respective servlets.

2. 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.

Note: In general in web applications, we are able to keep html files under application folder i.e. public area so that we are able to access that resources by using its name directly from client.

DURGASOFT

Step 2: Deployment Descriptor or web.xml file:

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 details w.r.t the web application

1. Welcome Files Configuration
2. Display Name Configuration
3. Servlets Configuration
4. Filters Configuration
5. Listeners Configuration
6. Context Parameters Configuration
7. Initialization Parameters Configuration
8. Session Time Out Configuration
9. Load On Startup Configuration
10. Error Page Configuration
11. Tag Library Configuration
12. Security Configuration

In general in web applications, we will deploy the servlets .class files under classes folder of the web application directory structure i.e. private area.

If we deploy any resource under private area then client is unable to access that resource through its name, client is able to access that resource through alias names or locators. In case of servlets, client is able to access servlet classes through the locators called as **URL Patterns.**

If we provide multiple number of servlets under classes folder and we provide a particular request to a particular servlet available under classes folder with an URL pattern then container should require mapping details between URL patterns and servlets class names as metadata in order to identify w.r.t servlet.

In the above context, to provide the required metadata to the container we have to provide servlet configuration in web.xml file. To provide servlet configuration in web.xml file we have to use the following xml tags.

```
<web-app>

    --------

<servlet>

<servlet-name>logical_name</servlet-name>

<servlet-class>fully qualified name of servlet</servlet-class>
```


DURGASOFT

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

Note: In the above servlets configuration, <servlet-name> tag value under <servlet> tag and <servlet-mapping> tag must be same.

Ex:

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

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

http://localhost:8080/loginapp/login

In servlet configuration, there are 3 ways to define URL patterns.

1. Exact Match Method
2. Directory Match Method
3. Extension Match Method

DURGASOFT

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.

Ex: <url-pattern>/abc/xyz</url-pattern>

If we define any URL pattern with exact match method then to access the respective resource we have to provide an URL pattern at client address bar along with URL, it must be matched with the URL pattern which we defined in web.xml file.

Ex: http://localhost:8080/app1/abc/xyz Valid

http://localhost:8080/app1/xyz/abc Invalid

http://localhost:8080/app1/xyz Invalid

http://localhost:8080/app1/abc Invalid

Note: In general in web applications, we will prefer to use exact match method to define an URL pattern for a particular servlet when we have a requirement to access respective servlet independently.

2. Directory Match Method:

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

Ex: <url-pattern>/abc/*</url-pattern>

If we define any URL pattern with this method then to access the respective resource from client we have to specify an URL pattern at client address bar it should match its prefix value with the prefix value of the URL pattern defined in web.xml file.

Ex: http://localhost:8080/app1/abc/xyz Valid

http://localhost:8080/app1/xyz/abc Invalid

http://localhost:8080/app1/abc Valid

http://localhost:8080/app1/abc/abc Valid

Note 1: In general in web applications, we will prefer to use directory match method to define an URL pattern when we have a requirement to pass multiple number of requests to a particular server side resource.

DURGASOFT

Note 2: In web applications we will use Filters to provide preprocessing and post processing to one or more number of servlets. In this context, when we send a request to respective servlet then container will bypass all the requests to the respective Filter.

To achieve this type of requirement we have to use directory match method to define an URL pattern for the respective Filter.

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.

Ex: <url-pattern>*.do</url-pattern>

If we define an URL pattern with this method then to access the respective server side resource from client we have to specify an URL pattern at client address bar, it may start with anything, but must be terminated with an extension which was specified in web.xml file.

Ex: http://localhost:8080/app1/login.do Valid

http://localhost:8080/app1/reg.do Valid

http://localhost:8080/app1/add.xyz Invalid

http://localhost:8080/app1/search.doo Invalid

Note 1: In general in web applications, we will prefer to use extension match method to define an URL pattern when we have a requirement to trap all the requests to a particular server side resource and to perform the respective server side action on the basis of the URL pattern name if we provided at client browser.

Note 2: If we design any web application on the basis of MVC then we have to use a servlet as controller, where the controller servlet has to trap all the requests and it has to perform a particular action on the basis of URL pattern name. Here to define URL pattern for the controller servlet we have to use extension method.

In web applications, web.xml file is mandatory or optional is completely depending on the server which we used.

In Apache Tomcat Server, web.xml file is optional when we have not used servlets, filters and so on. In Weblogic Server, web.xml file is mandatory irrespective of using servlets, filters and so on.

Up to servlets2.5 version[J2EE5.0] it is mandatory to provide web.xml file if we use servlets, listeners, filters and so on in our application. But in servlets3.0 version, there is a replacement for web.xml file i.e. Annotations, Annotations will make web.xml file is optional.

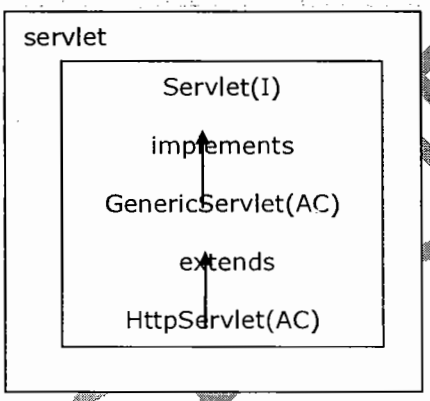
DURGASOFT

In web applications, it is not possible to change the name and location of the deployment descriptor because container will search for deployment descriptor with web.xml name under WEB-INF folder as per its predefined implementation.

Step 3: Prepare Web Resources:

If we want to prepare custom exceptions, threads and so on in Java applications then we have to use some predefined library provided by Java API.

Similarly if we want to prepare servlets in our web applications then we have to use some predefined library provided by Servlet API.



To design servlets Servlet API has provided the following predefined library as part of javax.servlet package and javax.servlet.http package.

Q: What is Servlet? and in how many ways we are able to prepare servlets?

Ans: Servlet is an object available at server machine which must implement either directly or indirectly Servlet interface.

As per the predefined library provided by Servlet API, there are 3 ways to prepare servlets.

1. Implementing Servlet interface:

In this approach, if we want to prepare servlet then we have to take an user defined class which must implement Servlet interface.

```
public class MyServlet implements Servlet
{ --------
  -------- }
```

2. Extending GenericServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to GenericServlet abstract class.

```
public class MyServlet implements GenericServlet
{ --------
  -------- }
```

3. Extending HttpServlet abstract class:

In this approach, if we want to prepare servlet then we have to take an user defined class as a subclass to HttpServlet abstract class.

```
public class MyServlet implements HttpServlet
{ --------
  -------- }
```

Step 4: Start the Server:

There are 3 ways to start the server.

1. Execute either startup.bat file or Tomcat7 Service Runner available under bin folder of Tomcat server.
   C:\Tomcat 7.0\bin
2. Use system program monitor Tomcat
   Start ⟶ All Programs ⟶ Apache Tomcat 7.0 ⟶ Monitor Tomcat
3. Use Apache Tomcat System Service
   Start ⟶ Type services.MVC in search field ⟶ Select Apache Tomcat 7.0 ⟶ Select Start Service Icon.

Step 5: Access the Web Application:

There are 2 ways to access the web applications.

1. Open web browser and type the complete URL on address bar.
   http://localhost:8080/app1/servlet
2. Open web browser type the URL up to
   http://localhost:8080

DURGASOFT

If we do the above automatically the Tomcat Home Page will be opened, where we have to select Manager Applications, provide username and password in Security window and click on OK button.

If we do the above automatically list of applications will be opened, where we have to select the required application.

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.

```
public interface Servlet {

    public void init(ServletConfig config)throws ServletException;

    public void service(ServletRequest req, ServletResponse res)throws ServletExcepption;

    public ServletConfig getServletConfig();

    public String getServletInfo();

    public void destroy();

}

public class MyServlet implements Servlet

 { -------

    ------- }
```

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.

DURGASOFT

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.

1. Specify response content type:

To specify response content type to the client we have to use the following method from ServletResponse.

public void setContentType(String MIME_TYPE)

where MIME_TYPE may be text/html, text/xml, image/jpeg, img/jpg and so on.

Ex: res.setContentType("text/html");

Note: The default content type in servlets is text/html.

When container encounters the above method then container will pick up the specified MIME_TYPE and container will set that value to content type response header in the response format.

When protocol dispatch the response format to client, before getting the response from response format body part first client will pick up content type response header value i.e. MiME_TYPE, client will prepare itself to hold the response as per the MIME_TYPE.

2. While executing the servlet we have to generate some dynamic response on response object, where to carry the dynamic response to the response object Servlet API has provide a predefined PrintWriter object internally.

To get the predefined PrintWriter object we have to use the following method from ServletResponse.

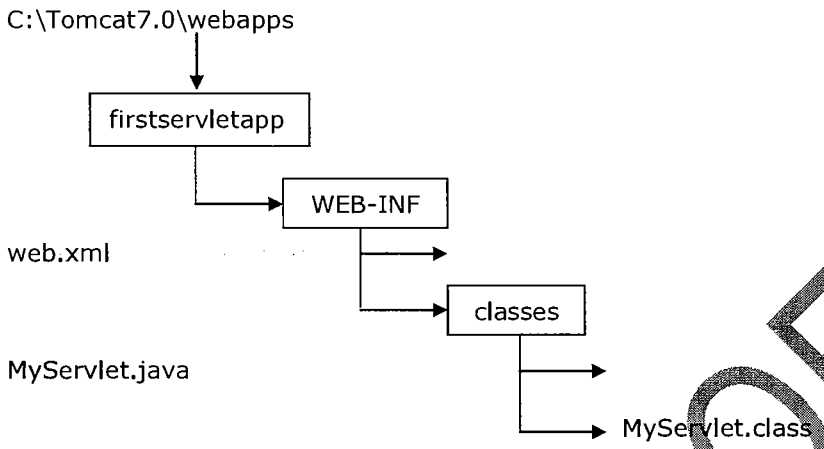
public PrintWriter getWriter()

Ex: PrintWriter out=res.getWriter();

DURGASOFT

Steps to Design First Servlet Application:

1. Web application Directory Structure:



2. Prepare Deployment Descriptor(web.xml):

web.xml:-

```
<web-app>
        <servlet>
                <servlet-name>ms</servlet-name>
                <servlet-class>MyServlet</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>ms</servlet-name>
                <url-pattern>/ms</url-pattern>
        </servlet-mapping>
</web-app>
```

3. Prepare Servlet:

MyServlet.java:-

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

import javax.servlet.Servlet;
```


DURGASOFT

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;


import javax.servlet.ServletResponse;

public class MyServlet  implements Servlet {

   public MyServlet() { }

public void init(ServletConfig config) throws ServletException {}

public void destroy() {}

public ServletConfig getServletConfig() {
        return null;
}

public String getServletInfo() {
        return null;
}

public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        out.println("<h1><center>hello</center></h1>");
}

}
```

To compile the above servlet we have to set classpath environment variable to servlet-api.jar provided by Tomcat server at C:\Tomcat7.0\lib\servlet-api.jar.

Ex: set classpath=%classpath%;C:\Tomcat7.0\lib\servlet-api.jar;

To access the above application we have to use the following URL on client address bar.

http://localhost:2020/firstservletapp/ms

DURGASOFT

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.

DURGASOFT

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.

DURGASOFT

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 ServletContextobject:

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.

Drawbacks of First Approach:

To design servlets if we use this approach we have to provide implementation for each and every method declared in Servlet interface irrespective of the actual application requirement.

The above approach will increase burden to the developers and it will increase unnecessary methods in web applications.

To overcome the above problem we have to use an alternative i.e. GenericServlet.

DURGASOFT

Second Approach to Design Servlets (Extendiging GenericServlet abstract class):

If we want to design servlets by using this approach then we have to take an user defined class which must be a subclass to GenericServlet abstract class.

```
public abstract class GenericServlet implements Servlet, ServletConfig, Serializable {
    private transient ServletConfig config;
    public void init(ServletConfig config)throws ServletException {
        this,config=config;
        init();
    }
    public void init()
    {  }
    public abstract void service(ServletRequest req, ServletResponse res)throws SE, IOE;
    public ServletConfig getServletConfig() {
      return config;
    }
    public String getServletInfo() {
        return null;
    }
    public void destroy()  {  }
}
public class MyServlet extends GenericServlet
{ -------
------- }
```

DURGASOFT

From the above predefined implementation of GenericServlet abstract class,

1. GenericServlet is an idea came from Adapter Design Pattern.
2. GenericServlet predefined abstract class has implemented Serializable interface so that all the GenericServlet objects(subclass objects of GenericServlet abstract class) are eligible for Serialization and Deserialization by default.
3. It is possible to serialize GenericServlet objects, but config predefined reference variable will not be participated in serialization and deserialization because config reference variable has declared as transient.
4. In GenericServlet abstract class, init(_) method is overloaded method.
5. In GenericServlet abstract class, still service(_,_) method is an abstract method.

Q: What is the requirement to override init(_,_) method in servlet applications?

Ans: In general as part of servlets design we will use service(_,_) method to provide application logic. In some cases, application logic may include the actual business logic and its prerequirement.

If we provide both prerequirement code and business logic within service(_,_) method then the performance of the application will be reduced because container will execute business logic and its prerequirement code for every request send by the client, but as per the convention it is sufficient to execute prerequirement code only one time.

In the above context, to improve the performance of servlet applicationwe have to provide the actual business logic in service(_,_) method and the prerequirement code in init(_) method because container will execute init(_) method only one time, but service(_,_) method for every request.

Note: In servlet applications, always it is suggestible to override init() method(second init() method) but if our prerequirement code may include any dat from ServletConfig object then it is suggestible to override init(ServletConfig config) method.

DURGASOFT

----------Application by using GenericServlet----------

Genericservletapp:-

web.xml:-

```
<web-app>
        <servlet>
                <servlet-name>GenericDemo</servlet-name>
                <servlet-class>GenericDemo</servlet-class>
        </servlet>
```

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

GenericDemo.java:-

```
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 GenericDemo extends GenericServlet {

        public void service(ServletRequest request, ServletResponse response) throws
ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                out.println("<h1><center>hello</center></h1>");
        }

}
```

The main difference between first approach and second approach of designing servlets in the point of flow of execution, in first approach container will execute only one init(_) method in servlet initialization, but in second approach container will execute two init() methods in servlet initialization.

Third Approach to Design Servlets (Extendiging HttpServlet abstract class):

Q: Whar are the differences between GenericServlet and HttpServlet?

Ans: 1. GenericServlet is protocol independent, but HttpServlet is http protocol dependent.

2. In case of GenericServlet, container will execute only service(_,_) method for any type of protocol request provided by the client, but In case of HttpServlet, container will execute a separate method on the basis request type which will be specified by the client.

DURGASOFT

Ex: If we specify Get request type at client browser then HttpServlet is able to execute doGet(_,_) method, for POST request type HttpServlet will execute doGet(_,_) method.

3. GenericServlet is not very good compatible with protocols, but HttpServlet is very good compatible with http protocol.

4. GenericServlet will not implement any specific protocol at server side, but HttpServlet has implemented http protocol at server side.

5. GenericServlet will not give any option to the developers to specify different types of requests at client browser, but HttpServlet will provide flexibility to the developers to specify different types of requests at client browser.

If we want to prepare servlets by using HttpServlet abstract class then we have to take an user defined class which must be a subclass to HttpServlet abstract class.

```
public abstract class HttpServlet extends GenericServlet {
  public void service(ServletRequest req, ServletResponse res)throws SE, IOException {
     HttpServletRequest hreq=(HttpServletRequest)req;
      HttpServletResponse hres=(HttpServletResponse)res;
      service(hreq, hres);
  }

  public void service(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE {
      String method=hreq.getMethod();
      if(method.equals("GET")) {
          doGet(hreq, hres);
       }
       if(method.equals("POST")) {
           doPost(hreq, hres);
       }
  }
  public void doGet(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE
  { }
```

DURGASOFT

```
    public void doPost(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE

    { }

}

public class MyServlet extends HttpServlet

{ ------

  ------ }
```

Note: In case of HttpServlet, we have to override either doGet(_,_) method or doPodt(_,_) method and son on doXxx(_,_) method with our web application logic on the basis of request type which we provide at client browser.

----------Application by using HttpServlet----------

<u>httpservletapp:</u>-

<u>web.xml:-</u>

```
<web-app>
        <servlet>
                <servlet-name>HttpDemo</servlet-name>
                <servlet-class>HttpDemo</servlet-class>
        </servlet>
        <servlet-mapping>
                <servlet-name>HttpDemo</servlet-name>
                <url-pattern>/http</url-pattern>
        </servlet-mapping>
</web-app>
```

<u>HttpDemo.java:-</u>

```
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 HttpDemo extends HttpServlet {

   public HttpDemo() {

   }
```


DURGASOFT

```
        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                out.println("<h1>hello this is http</h1>");
        }

}
```

The main difference between GenericServlet flow of execution and HttpServlet flow of execution is in GenericServlet flow of execution container will execute only one service(_,_) method to process the request, but in case of HttpServlet container will execute service(ServletRequest, Servletresponse), service(HttpServletRequest, HttpServletresponse), doXxx(HttpServletRequest, HttpServletresponse) on the basis of request type in order to process the request.

Q: Is it possible to override service(_,_) method in HttpServlet?

Ans: In HttpServlet, it is possible to override any of the service(_,_) methods, doXxx(_,_) methods, but always it is suggestible to override doXxx(_,_) methods on the basis of request types, it is not at all suggestible to override service(_,_) methods.

If we override service(_,_) method in HttpServlet then container will execute user provided service(_,_) method, but container is unable to execute predefined service(_,_) method so that it is not possible to reach doGet(_,_) method or doPost(_,_) method and so on.

Q: Is it possible provide both constructor and init() method in a single servlet?

Ans: Yes, it is possible provide both constructor and init() method in a single servlet.

If we provide a static block, a constructor, init(_) method, doXxx(_,_) method and destroy()method within a single servlet then container will execute static block at the time of servlet loading, constructor at the time of servlet instantiation, init() method at the time of performing servlet initialization, doXxx(_,_) method at the time of request processing and destroy() method at the time of servlet deinstantiation.

Ex: public class MyServlet extends HttpServlet {

```
        static {

            System.out.println("Servlet Loading");

        }

        public MyServlet() {

            System.out.println("Servlet Instantiation");
```


DURGASOFT

```
    }
    public void init() {
        System.out.println("Servlet Initialization");
    }


  public void doGet(HttpServletRequest hreq, HttpServletResponse hres)throws SE, IOE {
        System.out.println("Request Processing");
  }

  public void destroy() {
        System.out.println("Servlet Deinstantiation");
  }
}
```

If we send a request to above servlet then are able to see the following output on Server prompt

Servlet Loading

Servlet Instantiation

Servlet Initialization

Request Processing

Servlet Deinstantiation (when we close the server).

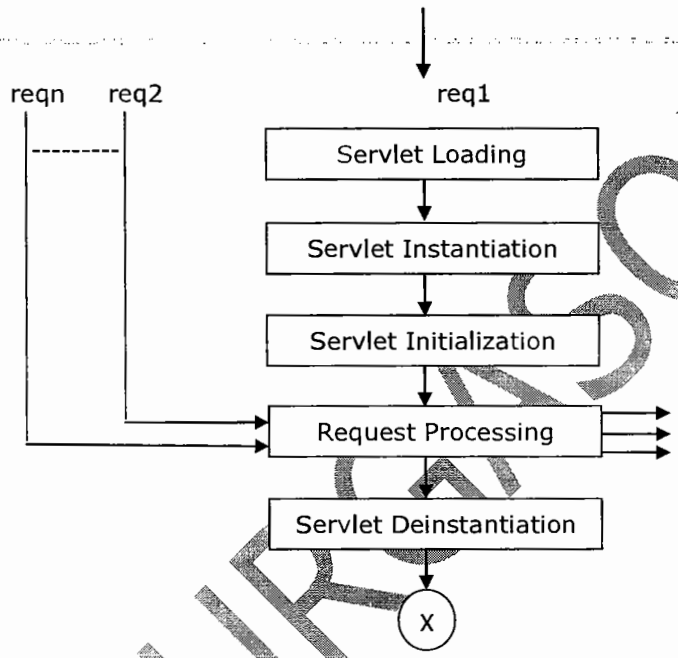
If we want to provide any constructor in servlet class then that constructor should be public and zero argument because container will search and execute public and zero argument constructor as part of servlet instantiation to create servlet object.

If we provide parameterized constructor without zero-argument constructor then container will raise an Exception like

DURGASOFT

javax.servlet.ServletException : Error-Instantiating Servlet class MyServlet with the root case java.lang.InstantiationException:MyServlet.

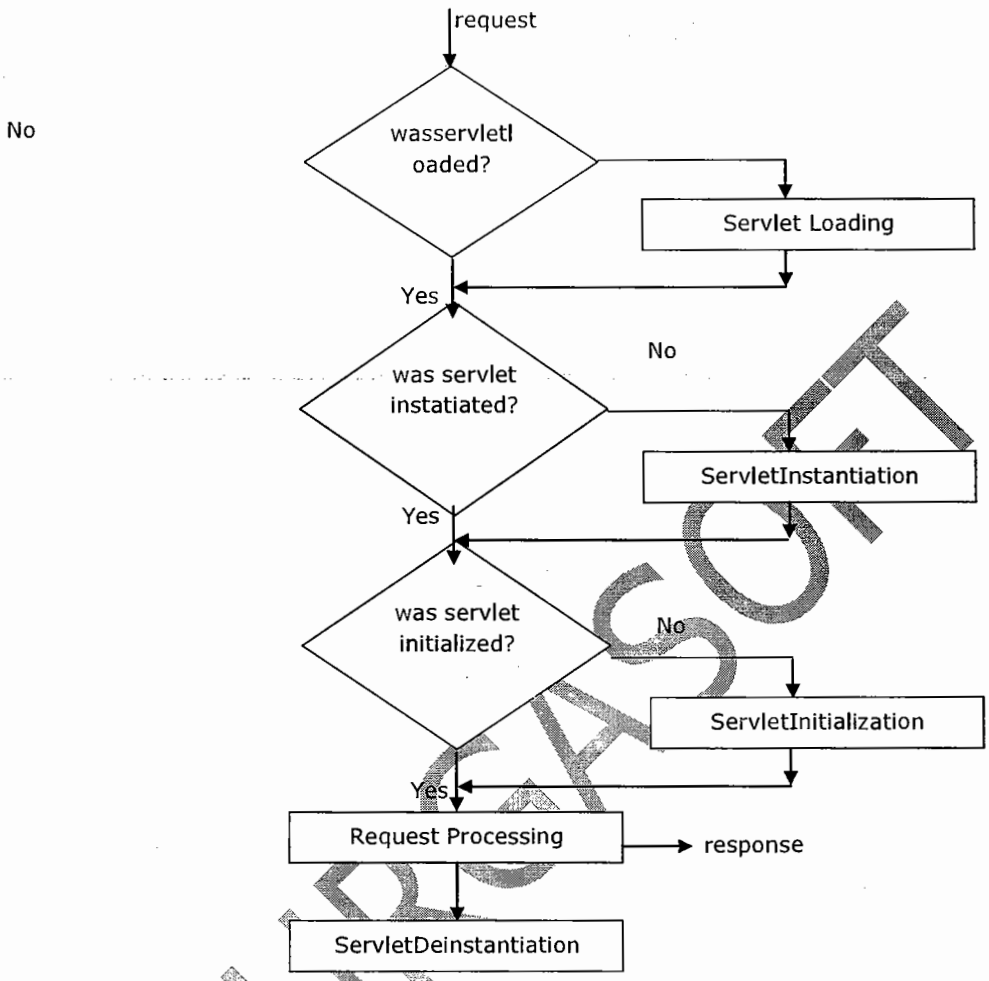
Servlet Life Cycle:



From the above representation when we send multiple number of requests to a particular servlet at a time then container will perform Servlet Loading, Instantiation and Initialization only for first request, container will bring all the later request directly to Request processing phase by skipping Servlet Loading, Instantiation and Initialization phases.

If we send multiple number of requests to a particular servlet then container will perform the following work for each and every requests to execute life-cycle stages.

DURGASOFT



From the above representations all the servlets and Jsp's are able to allow multiple number of requests at a time i.e. multiple number of threads at a time and all the servlets are able to process multiple number of threads at a time without providing data inconsistency. Therefore all the servlets and Jsp's are by default Thread safe.

Note: If any resource is able to allow and process multiple number of threads at a time without having side effects then that resource is called as **Thread Safe Resource.**

In servlet applications, as per the application requirement if we want to allow only one request at a time to a particular servlet then Servlet API has provided javax.servlet.SingleThreadModel marker interface.

```
public class MyServlet extends HttpServlet implements SingleThreadModel
{ ------------------ }
```

Where SingleThreadModel interface is a deprecated interface provided by Servlet API at the initial versions of servlet, which was not supported by the latest versions of servers.

Still, if we want to achieve our requirement about to allow only one request at a time when we have to use synchronization in the respective servlets.

Among the synchronization it is suggestible to use synchronized blocks over synchronized methods in order to improve performance.

```
public class MyServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)throws SE, IOE
    { ---------
      ---------
      synchronized(this)
      { ---------
        --------- }
    }
}
```

Note : In web applications, always it is suggestible to avoid to use SingleThreadModel and Synchronization because they will decrease the performance of web application.

Q: As part of servlet design if we access destroy() method in init() method then what will be the response from servlet?

Ans: As part of servlet design if we access destroy() method in init() then container will not perform servlet deinstantiation as part of servlet initialization.

Container is a software which includes number of modules to perform Servlet life cycle. When we send a request to the respective servlet from a client then container will start Servlet life cycle by executing its internal modules.

While performing servlet initialization, container will access init() method, init() method is not responsible to perform servlet initialization completely.

Similarly container will perform servlet deinstantiation by executing its internal modules. As part of this, container will access destroy() method.

Due to the above reasons eventhough we access destroy() method in init() method then we are able to access our provided destroy() method just like a normal method, it will not perform servlet deinsatntiation.

Q: If we send GET request from user form but if we override doPost(_,_) in the corresponding servlet what will be response from servlet?

Ans: The main convention of HttpServlet is, if we specify xxx request type t

hen container will execute doXxx(_,_) method i.e. we will override doXxx(_,_) method.

With the above convention, if we specify GET request at user form then we must override doGet(_,_) method.

In the above content, if we override do Post(_,_) method for GET request then container will execute predefined doGet(_,_) method provided by HttpServlet as for the predefined implementation of service(_,_) method.

In HttpServlet, doGet(_,_) method was implemented in such a way to send an error message with the following error description.

HTTP Status 405-HTTP method GET is not supported by this URL

Note: In HttpServlet, by default all doXxx(_,_) methods are implemented with the above convention only.

Note: In case of Tomcat server, if we specify any request type except GET and POST at user form then container will treat that request type has default request type and it will execute doGet(_,_) method.

This type of implementation may not be available with all remaining servers. With the above implementation, Tomcat server has provided support for only GET and POST request types.

User Interface[Forms Design]:

In general, in web applications there are 2 ways to prepare user forms.

1. Static Form Generation
2. Dynamic Form Generation

In case of Static Form Generation, we will prepare user form in the form of html file separately under application folder at the time of designing the application.

DURGASOFT

In case of Dynamic Form Generation, we will define user form a servlet. If we require Dynamic form then we have to access required respective servlet.

Headersapp:

web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<display-name>headersapp</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

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

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

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

</welcome-file-list>

<servlet>

<description></description>

<display-name>HeadersServlet</display-name>

<servlet-name>HeadersServlet</servlet-name>

<servlet-class>com.durgasoft.HeadersServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HeadersServlet</servlet-name>

<url-pattern>/headers</url-pattern>
```


DURGASOFT

```
</servlet-mapping>
</web-app>
```

HeadersServlet.java

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class HeadersServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		Enumeration<String> e=request.getHeaderNames();
		out.println("<html>");
		out.println("<body><center><br><br>");
		out.println("<table border='1' bgcolor='lightblue'>");
		out.println("<tr><td align='center'><h3>Header Names</h3></td><td
align='center'><h3>Header Values</h3></td></tr>");
		while(e.hasMoreElements()){
			String header_Name=(String)e.nextElement();
			String header_Value=request.getHeader(header_Name);
```


DURGASOFT

```
out.println("<tr><td>"+header_Name+"</td><td>"+header_Value+"</td></tr>");
            }
            out.println("</table></center></body></html>");
        }

}
```

Parametersapp:

registrationform.java

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color="red">
<h2>Durga Software Solutions</h2>
<h3>Student Registration Form</h3>
</font>
<form method="POST" action="./reg">
<table>
<tr>
        <td>Student Id</td>
        <td><input type="text" name="sid"/></td>
</tr>
<tr>
```

DURGASOFT

```
        <td>Student Name</td>
        <td><input type="text" name="sname"/></td>
</tr>
<tr>
        <td>Student Qualification</td>
        <td>
                <input type="checkbox" value="BSC" name="squal"/>BSC<br>
                <input type="checkbox" value="MCA" name="squal"/>MCA<br>
                <input type="checkbox" value="PHD" name="squal"/>PHD</br>
        </td>
</tr>
<tr>
        <td>Student Gender</td>
        <td>
                <input type="radio" value="Male" name="sgender"/>Male<br>
                <input type="radio" value="Female" name="sgender"/>Female<br>
        </td>
</tr>
<tr>
        <td>Student Technologies</td>
        <td>
                <select size="5" name="stech" multiple="multiple">
                        <option value="C">C</option>
                        <option value="C++">C++</option>
                        <option value="Java">JAVA</option>
                        <option value=".Net">.Net</option>
                        <option value="Oracle">Oracle</option>
```


DURGASOFT

```
                <option value="Testing Tools">Testing Tools</option>
            </select>
        </td>
</tr>
<tr>
        <td>Branch</td>
        <td>
            <select name="branch">
                <option value="Ameerpet">Ameerpet</option>
                <option value="S R Nagar">S R Nagar</option>
                <option value="Madapur">Madapur</option>
                <option value="KPHB">KPHB</option>
            </select>
        </td>
</tr>
<tr>
        <td>Student Address</td>
        <td><textarea rows="10" cols="50" name="saddr"></textarea></td>
</tr>
<tr>
        <td><input type="submit" value="Registration"/></td>
</tr>
</table>
</form>
</body>
</html>
```

DURGASOFT

web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<display-name>parametersapp</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

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

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

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

</welcome-file-list>

<servlet>

<description></description>

<display-name>RegistrationServlet</display-name>

<servlet-name>RegistrationServlet</servlet-name>

<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>RegistrationServlet</servlet-name>

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

</servlet-mapping>

</web-app>
```

DURGASOFT

RegistrationServlet.java

```
package com.durgasoft;

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 RegistrationServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;


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

                response.setContentType("text/html");

                PrintWriter out=response.getWriter();

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

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

                String[] squal=request.getParameterValues("squal");

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

                String[] stech=request.getParameterValues("stech");

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

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

                String qual="";

                for(int i=0;i<squal.length;i++){

                        qual=qual+squal[i]+"<br>";

                }
```


DURGASOFT

```
            String tech="";
            for(int j=0;j<stech.length;j++){
                tech=tech+stech[j]+"<br>";
            }
            out.println("<html>");
            out.println("<body>");
            out.println("<font color='red'>");
            out.println("<h2>Durga Software Solutions</h2>");
            out.println("<h3>Student Registration Details</h3>");
            out.println("</font>");
            out.println("<table border='1'>");
            out.println("<tr><td>Student Id</td><td>"+sid+"</td></tr>");
            out.println("<tr><td>Student Name</td><td>"+sname+"</td></tr>");
            out.println("<tr><td>Student
Qualification</td><td>"+qual+"</td></tr>");
            out.println("<tr><td>Student Gender</td><td>"+sgender+"</td></tr>");
            out.println("<tr><td>Student
Technologies</td><td>"+tech+"</td></tr>");
            out.println("<tr><td>Branch</td><td>"+branch+"</td></tr>");
            out.println("<tr><td>Student Address</td><td>"+saddr+"</td></tr>");
            out.println("</table></body></html>");
        }

}
```

DURGASOFT

Loginapp:

layout.html

```
<!DOCTYPE html>
<frameset rows="20%,65%,15%">
	<frame src="header.html"/>
	<frameset cols="20%,80%">
		<frame src="menu.html"/>
		<frame src="welcome.html" name="body"/>
	</frameset>
	<frame src="footer.html"/>
</frameset>
```

Header.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="maroon">
<center>
<font color="white" size="7">
<b>
DURGA SOFTWARE SOLUTIONS
</b>
</font>
```

DURGASOFT

```
</center>
</body>
</html>
```

menu.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightyellow">
<br><br>
<h3>
<a href="loginform.html" target="body">Login</a>
<br><br>
<a href="registrationform.html" target="body">Registration</a>
</h3>
</body>
</html>
```

welcome.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
```

DURGASOFT

```
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<center>
<font color="red" size="6">
<b><br><br><br>
<marquee>
Welcome To Durga Software Solutions
</marquee>
</b>
</font>
</center>

</body>
</html>
```

footer.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="blue">
<center>
<font color="white" size="5">
```

DURGASOFT

```
<b>
copyright reserved @ Durga Software Solutions, S R Nagar
</b>
</font>
</center>

</body>
</html>
```

loginform.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<br><br>
<form method="POST" action="./login">
<center>
<table>
<tr>
        <td>User Name</td>
        <td><input type="text" name="uname"/></td>
</tr>
<tr>
```

DURGASOFT

```
        <td>Password</td>
        <td><input type="password" name="upwd"/></td>
</tr>
<tr>
<td><input type="submit" value="Login"/></td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

registrationform.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body bgcolor="lightblue">
<br><br>
<form method="POST" action="./reg">
<center>
<table>
<tr>
        <td>User Name</td>
        <td><input type="text" name="uname"/></td>
```


DURGASOFT

```
</tr>
<tr>
        <td>Password</td>
        <td><input type="password" name="upwd"/></td>
</tr>
<tr>
        <td>User Email</td>
        <td><input type="text" name="uemail"/></td>
</tr>
<tr>
        <td>User Mobile Num</td>
        <td><input type="text" name="umobile"/></td>
</tr>

<tr>
<td><input type="submit" value="Registration"/></td>
</tr>
</table>
</center>
</form>
</body>
</html>
```

DURGASOFT

web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<display-name>loginapp</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

<welcome-file>index.htm</welcome-file>

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

<welcome-file>default.html</welcome-file>

<welcome-file>default.htm</welcome-file>

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

</welcome-file-list>

<servlet>

<description></description>

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

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

<servlet-class>com.durgasoft.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

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

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

</servlet-mapping>

<servlet>

<description></description>
```

DURGASOFT

```
<display-name>RegistrationServlet</display-name>
<servlet-name>RegistrationServlet</servlet-name>
<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegistrationServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
</web-app>
```

LoginServlet.html

```
package com.durgasoft;
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 {
	private static final long serialVersionUID = 1L;

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

DURGASOFT

```
            String upwd=request.getParameter("upwd");
            UserService us=new UserService();
            String status=us.checkLogin(uname,upwd);
            out.println("<html>");
            out.println("<body bgcolor='lightblue'>");
            out.println("<center><br><br>");
            out.println("<font color='red' size='7'>");
            if(status.equals("success")){
                out.println("Login Success");
            }
            if(status.equals("failure")){
                out.println("Login Failure");
            }
            out.println("</font></center></body></html>");

        }

}
```

RegistrationServlet.java

```
package com.durgasoft;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
```


DURGASOFT

```
import javax.servlet.http.HttpServletResponse;


public class RegistrationServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;


        protected void 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 uemail=request.getParameter("uemail");

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

                UserService us=new UserService();

                String status=us.registration(uname,upwd,uemail,umobile);

                out.println("<html>");

                out.println("<body bgcolor='lightblue'>");

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

                out.println("<font color='red' size='7'>");

                if(status.equals("success")){

                        out.println("Registration Success");

                }

                if(status.equals("failure")){

                        out.println("Registration Failure");

                }

                if(status.equals("existed")){

                        out.println("User Existed Already");
```


DURGASOFT

```
            }
            out.println("</font></center></body></html>");

        }

}
```

UserService.java

```
package com.durgasoft;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;

public class UserService {
        Connection con;
        Statement st;
        ResultSet rs;
        String status="";
        public UserService() {
            try {
                Class.forName("oracle.jdbc.OracleDriver");

        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"durga");
                st=con.createStatement();
            } catch (Exception e) {
```


DURGASOFT

```
                e.printStackTrace();
        }
}
public String checkLogin(String uname, String upwd){
        try {
                rs=st.executeQuery("select * from reg_Users where
uname='"+uname+"' and upwd='"+upwd+"'");
                boolean b=rs.next();
                if(b==true){
                        status="success";
                }else{
                        status="failure";
                }
        } catch (Exception e) {

        }
        return status;
}
public String registration(String uname, String upwd, String uemail, String umobile){
        try {
                rs=st.executeQuery("select * from reg_Users where
        uname='"+uname+"'");
                boolean b=rs.next();
                if(b==true){
                        status="existed";
                }else{
                        st.executeUpdate("insert into reg_Users
        values('"+uname+"','"+upwd+"','"+uemail+"','"+umobile+"')");
```


DURGASOFT

```
                        status="success";
                    }
            } catch (Exception e) {
                    status="failure";
                    e.printStackTrace();
            }
            return status;
    }

}
```

Dynamicformapp:

updateform.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color="red">
<h2>Durga Software Solutions</h2>
<h3>Student Update Form</h3>
</font>
<form method="GET" action="./edit">
<table>
<tr>
```

DURGASOFT

```
        <td>Student Id</td>
        <td><input type="text" name="sid"/></td>
</tr>
<tr>
        <td><input type="submit" value="GetEditForm"/></td>
</tr>
</table>
</form>
</body>
</html>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
<display-name>dynamicformapp</display-name>
<welcome-file-list>
<welcome-file>updateform.html</welcome-file>
</welcome-file-list>
<servlet>
<description></description>
<display-name>EditFormServlet</display-name>
<servlet-name>EditFormServlet</servlet-name>
<servlet-class>com.durgasoft.EditFormServlet</servlet-class>
</servlet>
```


DURGASOFT

```
<servlet-mapping>
<servlet-name>EditFormServlet</servlet-name>
<url-pattern>/edit</url-pattern>
</servlet-mapping>
<servlet>
<description></description>
<display-name>UpdateServlet</display-name>
<servlet-name>UpdateServlet</servlet-name>
<servlet-class>com.durgasoft.UpdateServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>UpdateServlet</servlet-name>
<url-pattern>/update</url-pattern>
</servlet-mapping>
</web-app>
```

EditFormServlet.java

```
package com.durgasoft;
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;
```

DURGASOFT

```
public class EditFormServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;


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

                response.setContentType("text/html");

                PrintWriter out=response.getWriter();

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

                StudentService ss=new StudentService();

                StudentTo sto=ss.getStudent(sid);

                if(sto==null){

                        out.println("<html>");

                        out.println("<body>");

                        out.println("<br><br>");

                        out.println("<font color='red' size='6'>");

                        out.println("Student Not Existed");

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

                        out.println("<br><br>");

                        out.println("<h3><a href='./updateform.html'>|Update
                        Form|</a></h3>");

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

                }else{

                        out.println("<html>");

                        out.println("<body>");

                        out.println("<font color='red'>");

                        out.println("<h2>Durga Software Solutions</h2>");

                        out.println("<h3>Student Edit Form</h3>");

                        out.println("</font>");
```


DURGASOFT

```
                    out.println("<form method='GET' action='./update'>");

                    out.println("<table>");

                    out.println("<tr><td>Student Id</td><td>"+sid+"</td></tr>");

                    out.println("<input type='hidden' name='sid' value='"+sid+"'/>");

                    out.println("<tr><td>Student Name</td><td><input type='text'
                    name='sname' value='"+sto.getSname()+"'/></td></tr>");

                    out.println("<tr><td>Student Address</td><td><input type='text'
                    name='saddr' value='"+sto.getSaddr()+"'/></td></tr>");

                    out.println("<tr><td><input type='submit'
                    value='Update'/></td></tr>");

                    out.println("</table></form></body></html>");

                }

            }

}
```

UpdateServlet.java

```
package com.durgasoft;

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 EditFormServlet extends HttpServlet {

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


DURGASOFT

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String sid=request.getParameter("sid");
        StudentService ss=new StudentService();
        StudentTo sto=ss.getStudent(sid);
        if(sto==null){
            out.println("<html>");
            out.println("<body>");
            out.println("<br><br>");
            out.println("<font color='red' size='6'>");
            out.println("Student Not Existed");
            out.println("</font>");
            out.println("<br><br>");
            out.println("<h3><a href='./updateform.html'>|Update
            Form|</a></h3>");
            out.println("</body></html>");
            }else{
            out.println("<html>");
            out.println("<body>");
            out.println("<font color='red'>");
            out.println("<h2>Durga Software Solutions</h2>");
            out.println("<h3>Student Edit Form</h3>");
            out.println("</font>");
            out.println("<form method='GET' action='./update'>");
            out.println("<table>");
            out.println("<tr><td>Student Id</td><td>"+sid+"</td></tr>");
```


DURGASOFT

```
                out.println("<input type='hidden' name='sid' value='"+sid+"'/>");

                out.println("<tr><td>Student Name</td><td><input type='text'
                name='sname' value='"+sto.getSname()+"'/></td></tr>");

                out.println("<tr><td>Student Address</td><td><input type='text'
                name='saddr' value='"+sto.getSaddr()+"'/></td></tr>");

                out.println("<tr><td><input type='submit'
                value='Update'/></td></tr>");

                out.println("</table></form></body></html>");

            }

        }

}
```

StudentService.java

```
package com.durgasoft;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;


public class StudentService {

        Connection con;

        Statement st;

        ResultSet rs;

        String status="";

        StudentTo sto;

        public StudentService() {

            try {
```

DURGASOFT

```
                Class.forName("oracle.jdbc.OracleDriver");
con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system","
durga");
                st=con.createStatement();
        } catch (Exception e) {
                e.printStackTrace();
        }
}
public StudentTo getStudent(String sid){
        try {
                rs=st.executeQuery("select * from student where sid='"+sid+"'");
                boolean b=rs.next();
                if(b==true){
                        sto=new StudentTo();
                        sto.setSid(rs.getString(1));
                        sto.setSname(rs.getString(2));
                        sto.setSaddr(rs.getString(3));
                }else{
                        sto=null;
                }
        } catch (Exception e) {

        }
        return sto;
}
public String update(String sid, String sname, String saddr){
        try {
```


DURGASOFT

```
                st.executeUpdate("update student set
        sname='"+sname+"',saddr='"+saddr+"' where sid='"+sid+"'");
                status="success";
        } catch (Exception e) {
                status="failure";
                e.printStackTrace();
        }
        return status;
    }

}
```

StudentTo.java

```
package com.durgasoft;
public class StudentTo {
    private String sid;
    private String sname;
    private String saddr;
    public String getSid() {
        return sid;
    }
    public void setSid(String sid) {
        this.sid = sid;
    }
    public String getSname() {
        return sname;
```

DURGASOFT

```
        }
        public void setSname(String sname) {
                this.sname = sname;
        }
        public String getSaddr() {
                return saddr;
        }
        public void setSaddr(String saddr) {
                this.saddr = saddr;
        }

}
```

Welcome Files:

In general, in all the web applications some pages like login pages, index pages, home pages and so on are the first pages.

In the above context, to access the first pages we have to specify the respective html page name or jsp page name as resource name in URL eventhough they are common for each and every user.

To overcome the problem, we have to declare the respective html or jsp page as welcome file.

Welcome file is the first page of the web application, it must be executed by the container automatically when we access the respective application without specifying resource name in URL.

To declare welcome file in web.xml file, we have to use the following xml tags.

Ex: <web-app>

<welcome-file-list>

<welcome-file>file1</welcome-file>

<welcome-file>file1</welcome-file>

DURGASOFT

```
-------------
</welcome-file-list>
-------------
</web-app>
```

From the above tags representation, it is possible to provide more than one welcome file with in a single web application but w.r.t. multiple no. of modules.

If we provide more than one welcome file with in a single web application w.r.t. modules the container will search for the respective welcome file as per the order in which we configured web.xml file.

Smooth Deployment:

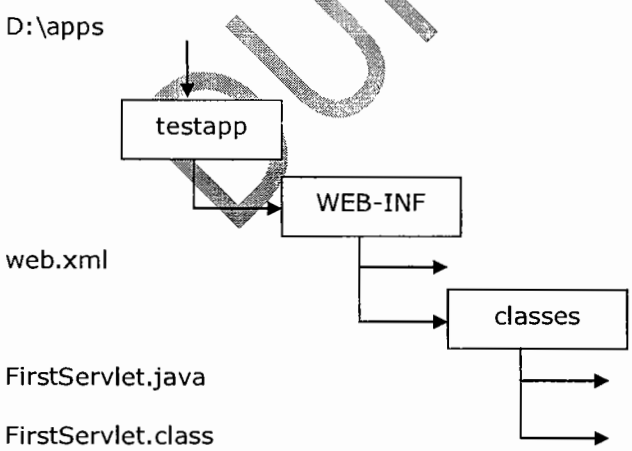
In general, we will prepare web applications with Tomcat server by creating the entire web application directly structure under webapps folder.

In this case, when we start the server then automatically the prepared web application will be deployed into the server.

The above approach to deploy the web applications is called **Hard Deployment**, it is not suggestible.

To perform Smooth Deployment for web applications we have to use the following steps.

Step 1: Prepare web application separately.



Step 2: Prepare war file for the web application by using the following command.

```
D:\apps\testapp>jar –cvf testapp.war *.*
```


DURGASOFT

Step 3: Start the Tomcat server and open Manager Applications.

http://localhost:1010/manager/html/

Step 4: Upload war file in order to deploy web application.

Go to war file to deploy section in Tomcat Web Application Manager, select war file by click on Browse button and click on Deploy button.

If we click on Deploy button then automatically uploaded war file will be deployed onto the Tomcat server with war file name as Application Context.

Step 5: Access the application.

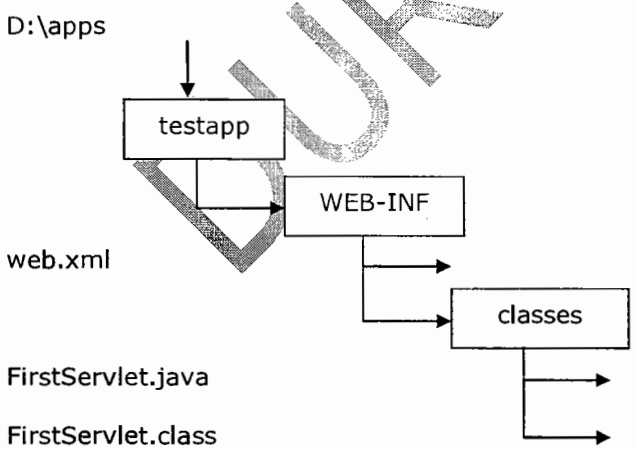
Weblogic Server:

Weblogic Server is an Application server introduced by Bea, it will provide very good middle ware services like JNDA, JTA, Security and so on.

Weblogic_10.3 version is compatible with jdk6 and it able to provide support for servlet2.5 and jsp2.1.

To deploy and execute web applications in Weblogic server we have to use the following steps.

Step 1: Prepare web application and its war file.



In the above application, we have to compile all the servlets under Weblogic environment.

DURGASOFT

To achieve this, we have to set CLASSPATH environment variable to weblogic.jar file provided by Weblogic server.

D:\apps\testapp\WEB-INF\classes>setclasspath=C:\bea\wlserver_10.3\

server\lib\weblogic.jar;

D:\apps\testapp\WEB-INF\classes>javac *.java

To prepare war file we have to use the following command on command prompt.

D:\apps\testapp>jar –cvf testapp1.war *.*

Step 2: Start Weblogic server and open Administration Console.

To start Weblogic server we have to use the following path.

Start
↓
All programs
↓
Oracle Weblogic
↓
Weblogic Server10gR3
↓
Weblogic Server

If we do the above then Weblogic server will start and it will open welcome page of Weblogic server.

To open Administration Console we have to use the following path.

Click on Start the Administration Console button

If we do the above then Weblogic server Administration Console will be open, where we have to provide browser name (weblogic) and password (weblogic) and finally click on login button.

If we click on login button then Weblogic server will open Home page.

Step 3: Deploy and access the web application.

To deploy web application we have to use the following path.

Click on Deployments in Home page
↓

DURGASOFT

Install
↓
Upload your files
↓
Select war file by click on browse button in Upload a deployment to the admin server
↓
Click on next
↓
Next
↓
Next
↓
Next
↓
Next
↓
Finish
↓
Click on Testing
↓
http://127.0.0.1:7001/testapp1

ServletConfig:

ServletConfig is an object, it will store all the configuration details of a particular servlet, where the configuration details include logical name of the servlet, initialization parameters, reference of ServletContext object and so on.

ServletConfig is an object, it will provide the complete view of a particular servlet.

In web application, container will prepare ServletConfig objects individual to each and every servlet.

In web application execution, container will prepare ServletConfig object immediately after servlet instantiation and just before calling init(_) method in servlet initialization.

Container will destroy the ServletConfig object just before servlet deinstantiation.

Due to the above reasons, the life cycle of ServletConfig object is upto a particular servlet.

If we declare any data in ServletConfig object then that data will be shared upto the respective servlet.

Due to the above reason, the scope of ServletConfig object is upto a particular servlet.

In web applications, ServletConfig object will allow only parameters data, it will not allow attributes data.

In web applications, there are 2 ways to get ServletConfig object .

DURGASOFT

1. Use getServletConfig() method from Servlet interface

Ex: ServletConfig config=getServletConfig();

2. Override init(_) method

```
Ex: public class MyServlet extends HttpServlet {
                ServletConfig config;
 public void init(ServletConfig config){
this.config=config;
}
-------------
}
```

To get logical name of the servlet from its ServletConfig object we have to use the following method.

public String getServletName()

Ex: String lname=config.getServletName();

If we want to provide initialization parameters inServletConfig object then first we have to declare them in web.xml file.

To declare initialization parameters in web.xml file we have to use the following xml tags.

```
<web-app>
<servlet>
<init-param>
<param-name>name</param-name>
<param-value>value</param-value>
</init-param>
            -------------
</servlet>
        -------------
</web-app>
```

DURGASOFT

If we declare initialization parameters with the above approach then container will read them and store onto ServletConfig object at the time of creation when it receives request from the client.

To get a particular initialization parameter from ServletConfig object we have to use the following method.

public String getInitParameter (String name)

Ex: String a=config.getInitParameter("a");

To get all the initialization parameters from ServletConfig object we have to use the following method.

public Enumeration getInitParameterNames()

Ex: Enumeration e=config.getInitParameterNames();

----------Application by using ServletConfig----------

configapp:-

web.xml:

```
<web-app>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
<init-param>
<param-name>driver</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
<init-param>
<param-name>url</param-name>
<param-value>jdbc:odbc:sri</param-value>
</init-param>
<init-param>
<param-name>user</param-name>
<param-value>system</param-value>
</init-param>

<init-param>
<param-name>password</param-name>
<param-value>durga</param-value>
</init-param>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/config</url-pattern>
</servlet-mapping>
</web-app>
```


DURGASOFT

MyServlet.java:

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

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

publicclass MyServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                ServletConfig config=getServletConfig();
                String logicalName=config.getServletName();
                String driver=config.getInitParameter("driver");
                String url=config.getInitParameter("url");
                String user=config.getInitParameter("user");
                String password=config.getInitParameter("password");
                out.println("<html><body><h1>");
                out.println("Logical Name: "+logicalName+"<br><br>");
                out.println("Driver      : "+driver+"<br><br>");
                out.println("Url         : "+url+"<br><br>");
                out.println("User        : "+user+"<br><br>");
                out.println("Password    : "+password+"<br><br>");
                out.println("</h1></body></html>");
        }
}
```

ServletContext:

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.

DURGASOFT

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 inServletContext 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.

DURGASOFT

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.

To declare a context parameter in web.xml file we have to use the following xml files.

```
<web-app>
        ------------
<context-param>
<param-name>name</param-name>
<param-value>value</param-value>
</context -param>
-------------
</web-app>
```

When we start the server or at the time of application deployment the main job of the container is to recognize the web application and to prepare ServletContext object.

At the time of recognizing the application container will recognize web.xml file then perform web.xml file loading, parsing and reading the content.

While reading the content container will read all its context parameters and store them in ServletContext object at the time of creation.

DURGASOFT

To get the particular context parameter value from ServletContext object we will use the following method.

public String getInitParameter(String name)

Ex: String value=context.getInitParameter("name");

To get all the context parameter names from ServletContext object we will use the following method.

public Enumeration getInitParameterNames()

Ex: Enumeration e=context.getInitParameterNames();

In web application, ServletContext object is able to allow attributes.

To set an attribute on ServletContext object we will use the following method.

public void setAttribute(String name, Object value)

Ex: context.setAttribute("location", "Hyd");

To get an attribute from ServletContext object we will use the following method.

public Object getAttribute(String name)

Ex: String location=(String)context.getAttribute("location");

To remove an attribute from ServletContext object we will use the following method.

public void removeAttribute(String name)

Ex: context.removeAttribute("location");

To get all the names of attributes from ServletContext object we will use the following method.

public Enumeration getAttributeNames()

Ex: Enumeration e=context.getAttributeNames();

Q: What is ForeignContext?

Ans: ForeignContext is a ServletContext object of another web application being executed in the same server.

To get ForeignContext object we have to use the following method.

public ServletContext getContext(String path)

----------Application by using ServletContext----------

DURGASOFT

contextapp:-

web.xml:

```
<web-app>
<display-name>Context Application</display-name>
<context-param>
        <param-name>a</param-name>
        <param-value>apple</param-value>
</context-param>
<context-param>
        <param-name>b</param-name>
        <param-value>bombay</param-value>
</context-param>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/context</url-pattern>
</servlet-mapping>
</web-app>
```

MyServlet.java:

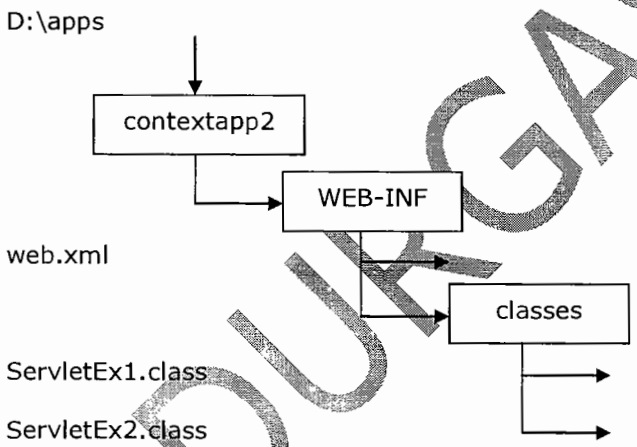
```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Enumeration;

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

publicclass MyServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();


                ServletContext context=getServletConfig().getServletContext();
                String logicalName=context.getServletContextName();
                String a=context.getInitParameter("a");
                String b=context.getInitParameter("b");
                Enumeration e=context.getInitParameterNames();
                context.setAttribute("c", "cat");
                context.setAttribute("d", "dog");
                out.println("<html><body><h1><br>");
```


DURGASOFT

```
                out.println("Logical Name : "+logicalName);
                out.println("<br>");
                out.println("a for ... "+a);
                out.println("<br>");
                out.println("b for ... "+b);
                out.println("<br>");
                while(e.hasMoreElements()){
                        out.println(e.nextElement()+"<br>");
                }
                out.println("c for ... "+context.getAttribute("c"));
                out.println("<br>");
                out.println("d for ... "+context.getAttribute("d")+"<br>");
                e=context.getAttributeNames();
                while(e.hasMoreElements()){
                        out.println(e.nextElement()+"<br>");
                }
                out.println("</h1></body></html>");
        }
}
```

Q: Consider the following web application

D:\apps



web.xml:

A short representation of web.xml

a ——→ aaa

b ——→ bbb Here a,b are context parameters

ServletEx1

c ccc ——→

d ddd ——→ Here c,d are initialization parameters

DURGASOFT

/first

ServletEx2

e eee →

f fff → here e,f are initialization parameters

/second

ServletEx1.java:

```
// import statements
public class ServletEx1 extends HttpServlet{
    public void deGet(HSR req, HSR res)throws SE, IOE {
ServletConfig config=getServletConfig();
 ServletContext context=req.getServletContext();
out.println("a->"+context.getInitParameter("a"));
out.println("b->"+context.getInitParameter("b"));
 out.println("c->"+config.getInitParameter("c"));
  out.println("d->"+config.getInitParameter("d"));
 out.println("e->"+config.getInitParameter("e"));
 out.println("f->"+config.getInitParameter("f"));
}
}
```

ServletEx2.java:

```
// import statements
public class ServletEx2 extends HttpServlet{
 public void deGet(HSR req, HSR res)throws SE, IOE{
ServletConfig config=getServletConfig();
 ServletContext context=req.getServletContext();
```


DURGASOFT

```
 out.println("a->"+context.getInitParameter("a"));
out.println("b->"+context.getInitParameter("b"));
 out.println("c->"+config.getInitParameter("c"));
out.println("d->"+config.getInitParameter("d"));
 out.println("e->"+config.getInitParameter("e"));
out.println("f->"+config.getInitParameter("f"));
}
}
```

Q1: http://localhost:8080/contextapp2/first?

Ans: a-->aaa c-->ccc e-->null

b-->bbb d-->ddd f-->null

Q2: http://localhost:8080/contextapp2/second?

Ans: a-->aaa c-->null e-->eee

b-->bbb d--> null f-->fff

In above web application, we can differentiate the scope of ServletConfig and ServletContext objects i.e. the scope of ServletConfig object is upto the respective servlet where as the scope of ServletContext object is through out the web application.

JBoss Server:

JBoss is an Application Server, it will provide almost all the middle ware services what application servers are provided in general.

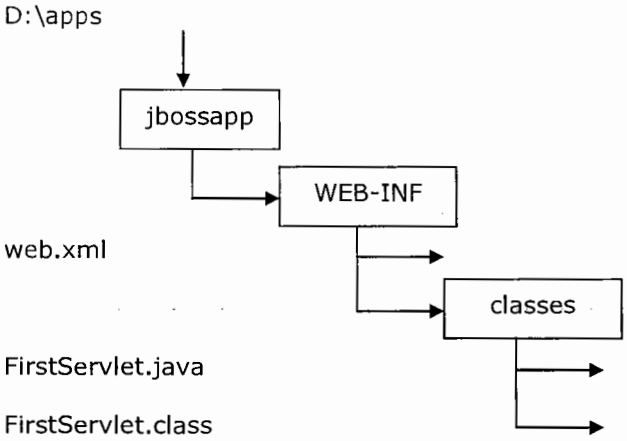
JBoss Server is compatible with jdk7 and it able to provide support for servlet3.0 version, jsp2.1 version and so on.

JBoss server is not having its own web container, it was used Tomcat Container.

If we want to deploy and execute web applications with JBoss server we have to use the following steps.

DURGASOFT

Step 1: Prepare web application and its war file.



In case of JBoss7 to compile all the servlets we need to set CLASSPATH environment variable to jboss-Servlet API_3.0_spec-1.0.0.Final.jar, which has provided by JBoss server in the following location.

C:\jboss-as-7.1.0.Final\modules\javax\servlet\api\main\ jboss-servlet-

api_3.0_spec-1.0.0.Final.jar

D:\apps\jbossapp\WEB-INF\classes>set classpath=C:\jboss-as- 7.1.0.Final

\modules\javax\servlet\api\main\ jboss-Servlet API_3.0_spec-1.0.0.Final.jar;

D:\apps\jbossapp\WEB-INF\classes>javac *.java

To prepare war file we have to use the following command on command prompt.

D:\apps\jbossapp>jar -cvf jbossapp1.war *.*

Step 2: Start JBoss Application Server.

C:\jboss-as-7.1.0.Final\bin, where double click on standalone.bat file

Step 3: Open Administration Console.

To open Administration Console we have to use the following URL on browser.

http://localhost:8888

Actually JBoss port number is 8080, but it was changed to 8888.

DURGASOFT

If we use the above URL then we are able to get JBoss Server Welcome page, where click on Administration Console and provide username(admin) and password(durga) in security window.

Step 4: Deploy web application.

If we click on OK button in security window automatically JBoss Home page will between open, where click on Manage Deployments under Deployments section.

If we do the above list of deployed applications will be displayed, where to deploy a new web application we have to use following path.

Click on Add Content button
↓
Browse
↓
Select war file
↓
Next
↓
Save
↓
Enable respective
↓
Confirm

Step 5: Access the web application.

Open another window, where we have to provide the following URL.

http://localhost:8888/jbossapp1/first

To change JBoss port number we have to use the following path.

C:\jboss-as-7.1.0.Final\standalone\configuration\ standalone.xml,

where search for 8080 and replace our port number(8888).

<select-binding name="http" port="8888"/>

To create an account in JBoss server we have to use the following path.

C:\jboss-as-7.1.0.Final\bin, where double click on add-user.bat file.

If we do the above then a command prompt will be open, where we have to provide the required details.

1. User type : Application User, press enter
2. Management realm : press enter

DURGASOFT

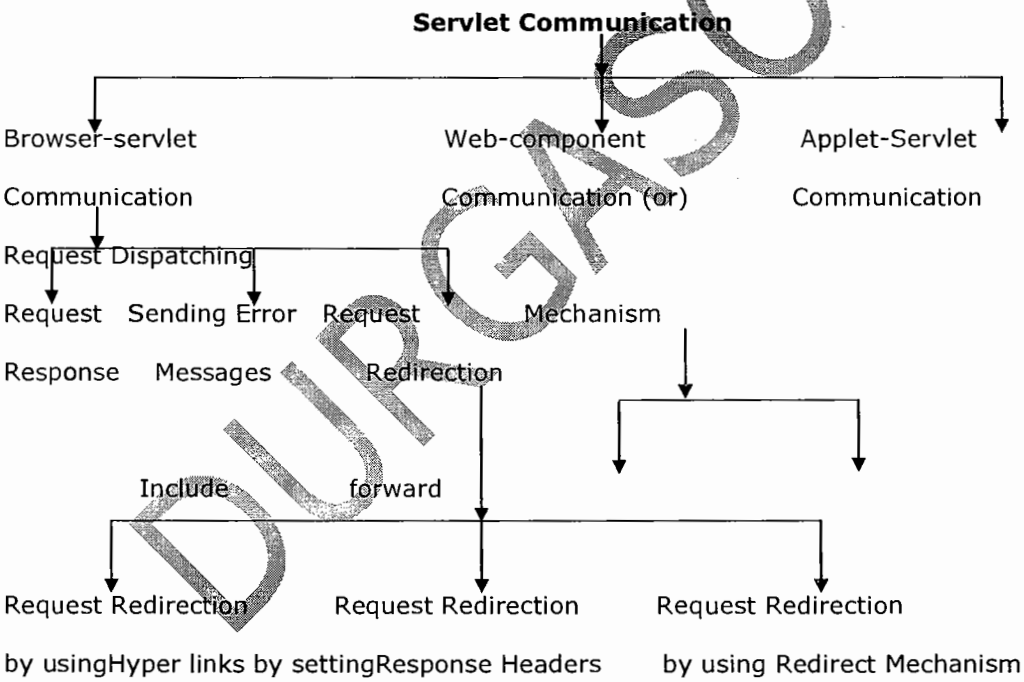
3. Username : durga, press enter
4. Password : durga, press enter
5. Re-enter password : durga, press enter
6. Is this correct yes/no ? yes press enter

Servlet Communication:

In general in web application deployment is not at all suggestible to provide the complete application logic with in a single web resource, it is suggestible to distribute the complete application logic over multiple web resources.

In the above context, to execute the application we must require communication between all the web resources, for this we have to use Servlet Communication.

In web application, we are able to provide servlet communication in the following 3 ways.



1. Browser-Servlet Communication:

In general in web applications, we will use browser as a client at client machine, from the browser we will send a request to a servlet available at server, where the servlet will be executed and generate some response to the client browser.

DURGASOFT

In the above process, we have provided communication between client browser and servlet, so that sending a normal request from client to server and getting a normal response from server to client is an examplefor **Browser-Servlet Communication**.

2. Sending Error Messages:

As part of the web application execution, if the container identify any exception or error then container will send the respective error message to be client in its standalone template.

As part of our application, if we want to send our own messages to the client in the container defined template we have to use the following method from response.

```
public void sendError(int statuscode, String description)
```

where statuscode may be 5xx.

SendErrorApp:

Registrationform.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color='red'>
<h2>Durga Consultency Services</h2>
<h3>User Registration Form</h3>
</font>
<form method="POST" action="./reg">
```


DURGASOFT

```
<table>
<tr>
        <td>User Name</td>
        <td><input type="text" name="uname"/></td>
</tr>
<tr>
        <td>User Age</td>
        <td><input type="text" name="uage"/></td>
</tr>
<tr>
        <td>User Email</td>
        <td><input type="text" name="uemail"/></td>
</tr>
<tr>
        <td>User Mobile</td>
        <td><input type="text" name="umobile"/></td>
</tr>
<tr>
        <td><input type="submit" value="Registration"/></td>
</tr>


</table>
</form>
</body>
</html>
```

DURGASOFT

Web.xml

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

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<display-name>senderrorapp</display-name>

<welcome-file-list>

<welcome-file>registrationform.html</welcome-file>

</welcome-file-list>

<servlet>

<description></description>

<display-name>RegistrationServlet</display-name>

<servlet-name>RegistrationServlet</servlet-name>

<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>RegistrationServlet</servlet-name>

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

</servlet-mapping>

</web-app>
```

RegistrationServlet.java

```
package com.durgasoft;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServlet;
```

DURGASOFT

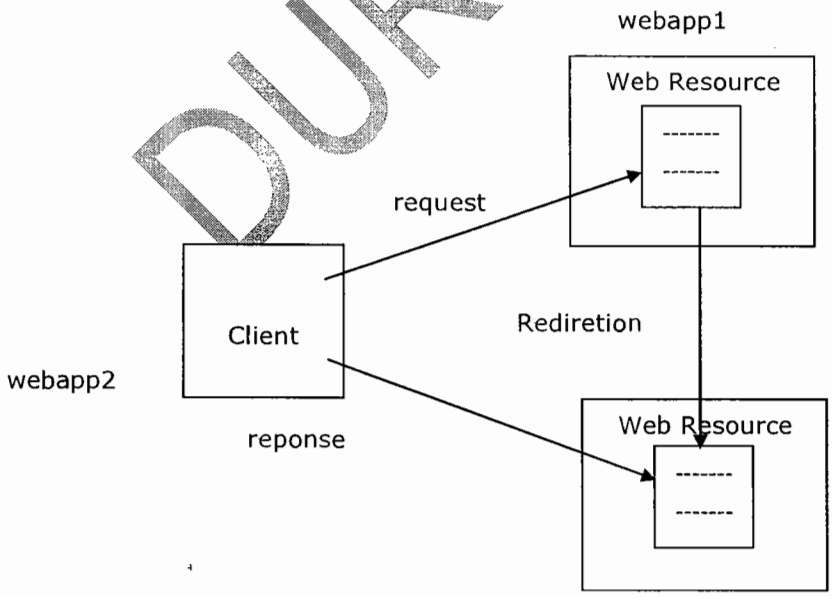
```
import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

public class RegistrationServlet extends HttpServlet {

        private static final long serialVersionUID = 1L;

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

                try {

                        response.setContentType("text/html");

                        PrintWriter out=response.getWriter();

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

                        int uage=Integer.parseInt(request.getParameter("uage"));

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

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


                        if(uage<18 || uage>30){

                        response.sendError(504, "User Age Is Not Sufficient for this
                        Recruitment");

                        }else{

                                out.println("<html>");

                                out.println("<body>");

                                out.println("<font color='red'>");

                                out.println("<h2>Durga Consultency Services</h2>");

                                out.println("<h3>User Registration Details</h3>");

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

                                out.println("<table border='1'>");

                                out.println("<tr><td>User
                                Name</td><td>"+uname+"</td></tr>");

                                out.println("<tr><td>User
                                Age</td><td>"+uage+"</td></tr>");
```


DURGASOFT

```
                    out.println("<tr><td>User
                    Email</td><td>"+uemail+"</td></tr>");

                    out.println("<tr><td>User
                    Mobile</td><td>"+umobile+"</td></tr>");

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

                }

            } catch (Exception e) {

                e.printStackTrace();

            }

        }


}
```

3. Request Redirection:

The process of bypassing the request from one web application to another web application is called as **Request Redirection**.

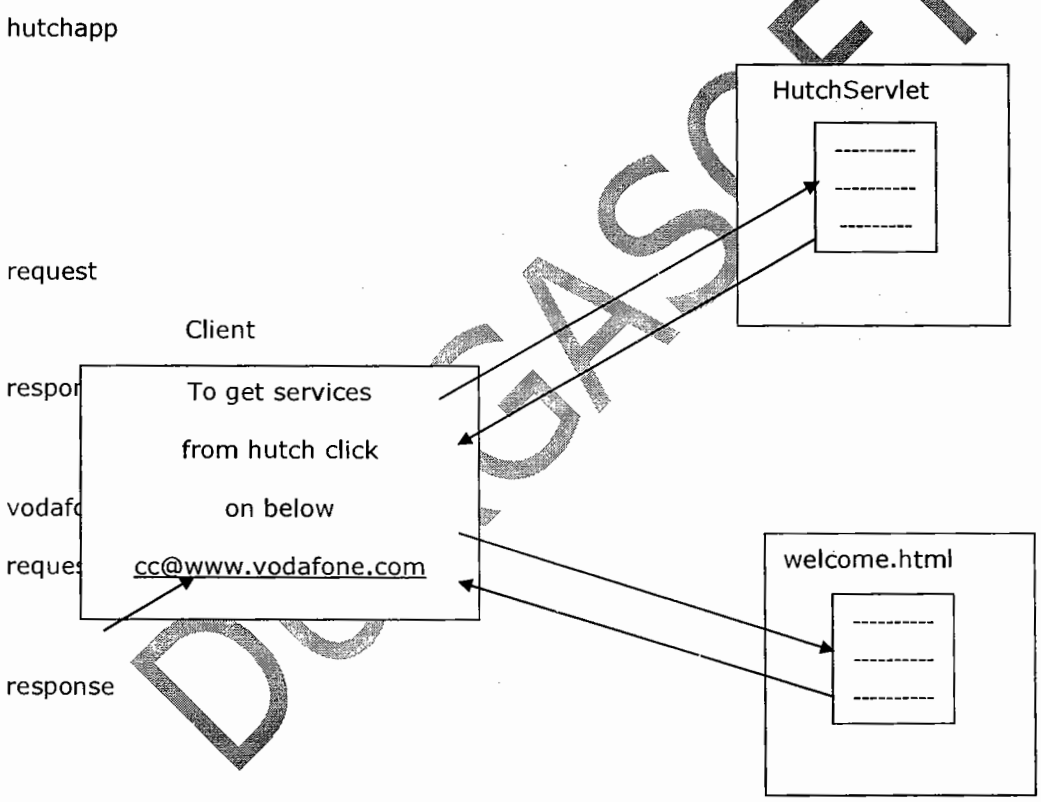


DURGASOFT

In web applications, we are able to achieve Request Redirection in the following 3 ways.

1. Request Redirection by using Hyper links
2. Request Redirection by setting Response Headers
3. Request Redirection by using Send Redirect Mechanism

1. Request Redirection by using Hyper links:



In this mechanism, when we send a request to first application some resources will be executed and generated an hyper link at client browser as a response.

By getting hyper link at client browser we have to click on it and we have to send another request to the new web application.

By executing some resources at new web application the required response will be generated to client browser.

Drawback:

In this Request Redirection mechanism, user may or may not click the generated hyper link at client browser after sending first request. So that this mechanism won't provide guarantee to achieve Request Redirection.

2. Request Redirection by setting Response Headers:

In this mechanism, first we will send a request to first web application, where first web application will set Redirectional Status Code to Status Line field and new web application URI to Location Response Header.

When the Response Format reached to the client then client will pick up Redirectional status code value from Status Line field, with this client browser will pick up Location Response Header value i.e. new web application URL then client browser will send a new request to new web application.

By executing some resources at new web application the required response will be generated at client machine.

To represent Request Redirection HttpServletResponse has introduced the following 2 constants.

1. public static final int SC_MOVED_TEMPORARILY;
2. public static final int SC_MOVED_PERMANENTLY;

To set a particular status code value to Response Header we will use the following method.

public void setStatus(int statuscode)

To set a particular Response Header value in Response Format we have to use the following method.

public void setHeader(String header_name, String value)

Drawback:

To perform Request Redirection, if we use this approach then every time we have to set Redirectional status code and new web application URL to Location Response Header.

DURGASOFT

3. Request Redirection by using Send Redirect Mechanism:

To perform Request Redirection, If we use Send Redirect Mechanism no need to use Hyper links, not required to set status code and Response Header values to the Response Format, but We need to use the following method.

public void sendRedirect(String url)

sendredirectapp:-

web.xml:

```
<web-app>
<display-name>sendredirectapp</display-name>
<servlet>
<servlet-name>HutchServlet</servlet-name>
<servlet-class>HutchServlet</servlet-class>
</servlet>


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

HutchServlet.java:

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

publicclass HutchServlet extends HttpServlet {

	protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.sendRedirect("http://localhost:2020/vodaphone/welcome.html");
	}

}
```

vodaphoneapp:-

welcome.html:

DURGASOFT

```
<html>
<bodybgcolor="red">
        <center><b><fontsize="7"color="black">
        <br><br>
        Welcome to Vodaphone
        </font></b></center>
</body>
</html>
```

2. Web-Component Communication:

The process of providing communication between more than one web component available at server machine is called as **Web-Component Communication.**

In general, web-component communication is available in between Servlet-Servlet, Servlet-Jsp, Servlet-HTML, Jsp-Jsp, Jsp-Servlet, Jsp-HTML and so on.

In web applications, we are able to achieve web-component communication in the following 2 ways.

1. Include Mechanism
2. Forward Mechanism

If we want to perform the above mechanisms internally we must use RequestDispatcher object. So that both include and forward mechanisms are commonly called as Request Dispatching Mechanisms.

To achieve web-component communication in web applications we have to use the following 2 steps

Step 1: Get RequestDispatcher object.

Step 2: Apply either include or forward mechanism by using the respective methods.

Step 1: 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

DURGASOFT

2. gteNamedDispatcher(_) method

2. ServletRequest

1. getRequestDispatcher(_) method

Q: What is the difference between getRequestDispatcher(_) method and getNamedDispatcher(_) method from ServletContext ?

Ans: 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 caseof the servlet, logical name is a name specified along with <servlet-name> tag in web.xml file.

public RequestDispatcher getNamedDispatcher(String path)

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

Ans: 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("/").

<u>Step 2: Apply either Include mechanism or Forward mechanism to achieve Web-Component Communication:</u>

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

DURGASOFT

Q: What are the differences between Include and Forward mechanisms?

Ans: In web applications, Include Request Dispatching mechanism can be used to include the target resource response into the present resource response.

In case of Include mechanism, when we send a request to first resource then container will prepare request and response objects, by executing some part in first resource container may generate some response in response object.

When container encounter include(_,_) method then container will bypass the request and response objects to target resource along with flow of execution without refreshing response object.

By executing the target resource some response will be generated in response object, at the end of target resource container will bypass request and response objects back to the first resource to continue its further execution.

In the above context, container will execute remaining content in first resource , some response will be added to response object, at the end of first resource container will dispatch ovaerall response to client.

Therefore, In case of Include mechanism, client is able to receive all resources which are participated in the present request processing.

In web applications, the main purpose of Forward mechanism is to forward request from present resource to target resource.

In case of Forward mechanism, when we send a request to first resource then container will create request and response objects, by executing some part in first resource container may generate some response in response object.

When container encounter forward(_,_) method then container will bypass the request and response objects to the target resource along with flow of execution by refreshing response object (by eliminating previous content in response object).

By executing the target resource some response will be generated in response object, at the end of target resource container will dispatch ovaerall response to client directly without moving back to the first resource.

Therefore, In case of Forward mechanism, client is able to receive only the target resource response which has included in the present request processing.

Q: What is Servlet Chaining?

Ans: The processofincluding more than one servlet in order to process a single request is called as **Servlet Chaining**or **Servlet Collaboration.**

Q:What are the differences between Forward mechanism and send Redirect mechanism?

DURGASOFT

Ans: 1. In web applications, Forward Request Dispatching mechanism can be used to provide the communication between two resources which must be available at same server.

In web applications, Send Redirect mechanism can be used to provide the communication between two resources which may be available at same server or at two different servers.

2. In case of Forward mechanism, one request is sufficient to establish the communication between two resources.

In case of Send Redirect mechanism, we need requests to establish the communication between two web resources.

includeapp:-

addform.html:

```
<html>
<bodybgcolor="lightgreen">
<center>
<formmethod="get"action="./add">
<br><br>
<tablebgcolor="lightyellow">
<tr><tdcolspan="2"><center><b><fontsize="6"color="red"><u>Product Add
Form</u></font></b></center></td></tr>
<tr><td>Product Id</td><td><inputtype="text"name="pid"/></td>
</tr>


<tr>
<td>Product Name</td>
<td><inputtype="text"name="pname"/></td>
</tr>
<tr>
<td>Product Cost</td>
<td><inputtype="text"name="pcost"/></td>
</tr>
<tr>
<td><inputtype="submit"value="ADD"/></td>
</tr>
</table></form></center></body></html>
```

web.xml:

```
<web-app>
<display-name>includeapp</display-name>
<welcome-file-list>
<welcome-file>addform.html</welcome-file>
</welcome-file-list>
<servlet>
<description></description>
<display-name>AddServlet</display-name>
```

DURGASOFT

```
<servlet-name>AddServlet</servlet-name>
<servlet-class>AddServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>AddServlet</servlet-name>
<url-pattern>/add</url-pattern>
</servlet-mapping>
</web-app>
```

Product.java:

```
publicclassProduct {
        private String pid;
        private String pname;
        privateintpcost;
        public String getPid() {
                returnpid;
        }
        publicvoid setPid(String pid) {
                this.pid = pid;
        }
        public String getPname() {
                returnpname;
        }
        publicvoid setPname(String pname) {
                this.pname = pname;
        }


publicint getPcost() {
                returnpcost;
        }
        publicvoid setPcost(int pcost) {
                this.pcost = pcost;
        }
}
```

ProductDao.java:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.ArrayList;

publicclass ProductDao {
        Connection con;
        Statement st;
        ResultSet rs;
        ArrayList<Product>al;
        ProductDao(){
                try {
```

DURGASOFT

```
                        Class.forName("oracle.jdbc.driver.OracleDriver");
        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system",
"durga");
                        st=con.createStatement();
                        al=new ArrayList<Product>();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        publicvoid add(String pid, String pname, int pcost){
                try {
                        st.executeUpdate("insert into product
values('"+pid+"','"+pname+"','"+pcost+")");
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public ArrayList<Product> getProducts(){
                try {
                        rs=st.executeQuery("select * from product");
                        while(rs.next()){
                                Product p=new Product();
                                p.setPid(rs.getString(1));
                                p.setPname(rs.getString(2));
                                p.setPcost(rs.getInt(3));
                                al.add(p);
                        }
                } catch (Exception e) {


                e.printStackTrace();
                }
                returnal;
        }
}
```

AddServlet.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

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

publicclass AddServlet extends HttpServlet {

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


DURGASOFT

```
                        response.setContentType("text/html");
                        PrintWriter out=response.getWriter();
                        String pid=request.getParameter("pid");
                        String pname=request.getParameter("pname");
                        int pcost=Integer.parseInt(request.getParameter("pcost"));
                        ProductDao pd=new ProductDao();
                        pd.add(pid,pname,pcost);
                        ArrayList<Product> prds=pd.getProducts();
                        out.println("<html><body><center><br><br>");
                        out.println("<table border='1' bgcolor='lightyellow'>");
                out.println("<tr><td>PID</td><td>PNAME</td><td>PCOST</td></tr>");
                        for(Object o : prds){
                                Product p=(Product)o;
        out.println("<tr><td>"+p.getPid()+"</td><td>"+p.getPname()+"</td><td>"+p.g
etPcost()+"</td></tr>");
                        }
                        out.println("</table></center><br><hr><br></body></html>");

        RequestDispatcherrd=request.getRequestDispatcher("/addform.html");
                                rd.include(request, response);
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

forwardapp:-

layout.html:

```
<html>
<framesetrows="20%,65%,15%">
<framesrc="header.html"/>
<framesetcols="20%,80%">
<framesrc="menu.html"/>
<framesrc="welcome.html"name="body"/>
</frameset>
<framesrc="footer.html"/>
</frameset>
</html>
```

Header.html:

```
<html><bodybgcolor="red">
<center><b><fontsize="7"color="white">
    Durga Software Solutions
</font></b></center>
</body></html>
```


DURGASOFT

menu.html:

```
<html>
<bodybgcolor="cyan"><br><br><center><b><fontsize="6">
<ahref="./addform.html"target="body">Add</a><br><br>
<ahref="./searchform.html"target="body">Search</a> <br><br>
<ahref="./updateform.html"target="body">Update</a><br><br>
<ahref="./deleteform.html"target="body">Delete</a>
</font></b></center></body>
</html>
```

footer.html:

```
<html><bodybgcolor="blue">
<center><b><fontsize="6"color="white">
copyrights2010-2020@www.durgasoft.com
</font></b></center></body></html>
```

welcome.html:

```
<html><bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br><marquee>
Welcome to Durga Software Solutions
</marquee>
</font></b></center></body></html>
```

addform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7">
<br>
<formmethod="get"action="./add">
<pre>
            Student Id          <inputtype="text"name="sid"/>

            Student Name        <inputtype="text"name="sname"/>

            Student Marks       <inputtype="text"name="smarks"/>

            <inputtype="submit"value="Add"/>
</pre></form></font></b></body></html>
```

searchform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7"><br><br>
```

DURGASOFT

```
<formmethod="get"action="./search">
<pre>
                Student Id          <inputtype="text"name="sid"/>

                <inputtype="submit"value="Search"/>
</pre></form></font></b></body></html>
```

updateform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7"><br><br>
<formmethod="get"action="./edit">
<pre>
                Student Id          <inputtype="text"name="sid"/>

                <inputtype="submit"value="GetEditForm"/>
</pre></form></font></b></body></html>
```

deleteform.html:

```
<html>
<bodybgcolor="lightgreen">
<b><fontsize="7">
<br><br>
<formmethod="get"action="./delete">
<pre>

                Student Id          <inputtype="text"name="sid"/>


<inputtype="submit"value="Delete"/>
</pre></form></font></b></body></html>
```

success.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Success
</font></b></center></body></html>
```

failure.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Failure
</font></b></center></body></html>
```

DURGASOFT

existed.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Student Existed Already
</font></b></center></body></html>
```

notexisted.html:

```
<html>
<bodybgcolor="lightyellow">
<center><b><fontsize="7"color="red">
<br><br>
Student Not Existed
</font></b></center></body></html>
```

StudentDao.java:

```
import java.sql.ResultSet;

publicinterface StudentDao {
	public String add(String sid, String sname, int smarks);
	public ResultSet search(String sid);
	public ResultSet getStudent(String sid);
	public String update(String sid, String sname, int smarks);
	public String delete(String sid);
}
```

StudentDaoImpl:

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

public class StudentDaoImpl implements StudentDao {

	Connection con;
	Statement st;
	ResultSet rs;
	String status = "";

	public StudentDaoImpl() {
		try {
			Class.forName("oracle.jdbc.driver.OracleDriver");
			con = DriverManager.getConnection(
				"jdbc:oracle:thin:@localhost:1521:xe", "system", "durga");
			st = con.createStatement();
		} catch (Exception e) {
```


DURGASOFT

```
                e.printStackTrace();
            }
        }

        public ResultSet getStudent(String sid) {
            try {
                rs = st.executeQuery("select * from student where sid='" + sid
                            + "'");
            } catch (Exception e) {
                e.printStackTrace();
            }
            return rs;
        }

        public String add(String sid, String sname, int smarks) {
            try {
                rs = getStudent(sid);
                boolean b = rs.next();
                if (b == true) {
                    status = "existed";
                } else {
                    int rowCount = st.executeUpdate("insert into student values('"
                                + sid + "','" + sname + "'," + smarks + ")");
                    if (rowCount == 1) {
                        status = "success";
                    } else {
                        status = "failure";
                    }
                }
            } catch (Exception e) {

e.printStackTrace();
            }
            return status;
        }

        public ResultSet search(String sid) {
            return getStudent(sid);
        }

        public String update(String sid, String sname, int smarks) {
            try {
                int rowCount = st
                            .executeUpdate("update student set sname='" + sname
                                        + "',smarks=" + smarks + " where sid='"
+ sid + "'");
                if (rowCount == 1) {
                    status = "success";
                } else {
                    status = "failure";
                }
            } catch (Exception e) {
```

DURGASOFT

```
                e.printStackTrace();
            }
            return status;
        }

        public String delete(String sid) {
            try {
                rs = getStudent(sid);
                boolean b = rs.next();
                if (b == true) {
                    int rowCount = st
                    .executeUpdate("delete from student where sid='" + sid+ "'");
                    if (rowCount == 1) {
                        status = "success";
                    } else {
                        status = "failure";
                    }
                } else {
                    status = "notexisted";
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return status;
        }
}
```

AddServlet.java:

```
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;
public class AddServlet extends HttpServlet
{
        public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
            try
            {
                String sid=req.getParameter("sid");
                String sname=req.getParameter("sname");
                int smarks=Integer.parseInt(req.getParameter("smarks"));
                StudentDao sd=new StudentDaoImpl();
                String status=sd.add(sid,sname,smarks);
```


DURGASOFT

```
                        if(status.equals("existed"))
                        {
                        RequestDispatcherrd1=req.getRequestDispatcher("/existed.html");
                                rd1.forward(req,res);
                        }
                        if(status.equals("success"))
                        {
                                req.getRequestDispatcher("success.html").forward(req,res);
                        }
                        if(status.equals("failure"))
                        {
                                req.getRequestDispatcher("failure.html").forward(req,res);
                        }
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}
```

SearchServlet.java:

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

import java.sql.ResultSet;

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


import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
public class SearchServlet extends HttpServlet
{
        public void doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
        {
                try
                {
                        res.setContentType("text/html");
                        PrintWriter out=res.getWriter();
                        String sid=req.getParameter("sid");
                        StudentDao sd=new StudentDaoImpl();
                        ResultSet rs=sd.search(sid);
                        boolean b=rs.next();
                        if(b==true)
                        {
                                out.println("<html>");
                                out.println("<body bgcolor='lightyellow'>");
                                out.println("<b><font size='6'><br>");
                                out.println("<pre>");
```


DURGASOFT

```
                        out.println("          Student Id....."+rs.getString(1));
                        out.println();
                        out.println("          Student Name....."+rs.getString(2));
                        out.println();
                        out.println("          Student Marks....."+rs.getInt(3));
                        out.println("</pre></font></b></body></html>");
                }
                else
                {
                        req.getRequestDispatcher("notexisted.html").forward(req,res);
                }
        }
        catch(Exception e)
        {
                e.printStackTrace();
        }
    }
}
```

EditFormServlet:

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.ResultSet;

import javax.servlet.ServletException;

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


public class EditFormServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {
        try {
            res.setContentType("text/html");
            PrintWriter out = res.getWriter();
            String sid = req.getParameter("sid");
            StudentDao sd = new StudentDaoImpl();
            ResultSet rs = sd.getStudent(sid);
            boolean b = rs.next();
            if (b == true) {
                out.println("<html>");
                out.println("<body bgcolor='lightgreen'>");
                out.println("<b><font size='7'>");
                out.println("<br>");
                out.println("<form method='get' action='./update'>");
                out.println("<pre>");
                out.println("          Student Id          " + rs.getString(1));
                out.println("<input type='hidden' name='sid' value='" + sid
                        + "'/>");
                out.println("          Student Name                <input
```


DURGASOFT

```
type='text' name='sname' value='"
                                        + rs.getString(2) + "'/>");
                                out.println();
                                out.println("          Student marks              <input
type='text' name='smarks' value='"
                                        + rs.getInt(3) + "'/>");
                                out.println();
                                out.println("          <input type='submit' value='Update'/>");
                                out.println("</pre></form></font></b></body></html>");
                        } else {
                                req.getRequestDispatcher("notexisted.html").forward(req, res);
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

UpdateServlet.java:

```
import java.io.IOException;

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

public class UpdateServlet extends HttpServlet {
        public void doGet(HttpServletRequest req, HttpServletResponse res)
                        throws ServletException, IOException {


                try {
                        String sid = req.getParameter("sid");
                        String sname = req.getParameter("sname");
                        int smarks = Integer.parseInt(req.getParameter("smarks"));
                        StudentDao sd = new StudentDaoImpl();
                        String status = sd.update(sid, sname, smarks);
                        if (status.equals("success")) {
                                req.getRequestDispatcher("success.html").forward(req, res);
                        }
                        if (status.equals("failure")) {
                                req.getRequestDispatcher("failure.html").forward(req, res);
                        }
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

DURGASOFT

DeleteServlet.java:

```
import java.io.IOException;

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

public class DeleteServlet extends HttpServlet {
	public void doGet(HttpServletRequest req, HttpServletResponse res)
		throws ServletException, IOException {
		String sid = req.getParameter("sid");
		StudentDao sd = new StudentDaoImpl();
		String status = sd.delete(sid);
		if (status.equals("success")) {
			req.getRequestDispatcher("success.html").forward(req, res);
		}
		if (status.equals("failure")) {
			req.getRequestDispatcher("failure.html").forward(req, res);
		}
		if (status.equals("notexisted")) {
			req.getRequestDispatcher("notexisted.html").forward(req, res);
		}
	}
}
```

DURGASOFT

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);
uc.setDoOutpput(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();
```

loginapp1:-

web.xml:-

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


DURGASOFT

LoginServlet.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 LoginServlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter out=response.getWriter();
		String uname=request.getParameter("uname");
		String upwd=request.getParameter("upwd");
		if(uname.equals("durga") && upwd.equals("durga")){
			out.println("Login Success");
		}
		else{
			out.println("Login Failure");
		}
	}

}
```

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(){
```


DURGASOFT

```
                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.getTex
t());
                        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);
        }
}


LoginApplet.html:-

<applet code="LoginApplet" width="500" height="500"></applet>
```

GlassFish Server:

GlassFish Server is an Application Server, provided by Sun Micro Systems.

GlassFish server will provide almost all the middle ware services what application servers are provided in general like JNDI, JMS, Java Mail, Security, JTA and so on.

DURGASOFT

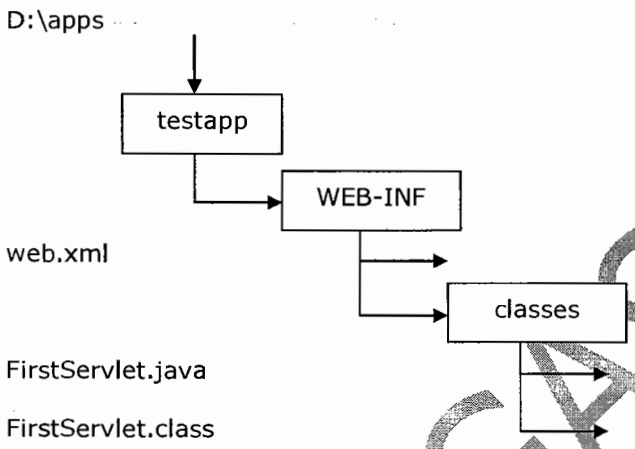
GlassFish version3 is compatible with java6 and above, it is able to provide support support for servlet3.0, jsp2.1 and so on.

To design and execute servlets in GlassFish server we are able to use Tomcat provided servlet API implementation i.e. Servlet API.jar.

If we want to design and execute web applications in GlassFish server

we have to use the following steps.

Step 1: Prepare web application and its war file separately.



To compile all the servlet files GlassFish server has provided Servlet API implementation in the form of javax.servlet.jar file.

GlassFish server has provided javax.servlet.jar file at the following location.

```
C:\glassfishv3\glassfish\modules

D:\apps\testapp\WEB-INF\classes>set classpath=C:\glassfishv3\
    glassfish\modules\ javax.servlet.jar;

D:\apps\testapp\WEB-INF\classes>javac *.java

D:\apps\testapp>jar -cvf testapp.war *.*
```

DURGASOFT

Step 2: Start the Application server.

To start Glassfish server we have execute startserv.bat file provided by Glassfish server at the following location.

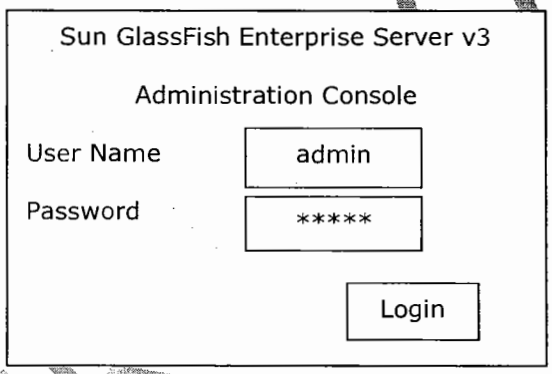
C:\glassfishv3\glassfish\bin

Step 3: Open Administration Console.

To open Administration Console in GlassFish server we have to use the following url at client browser.

http://localhost:4848

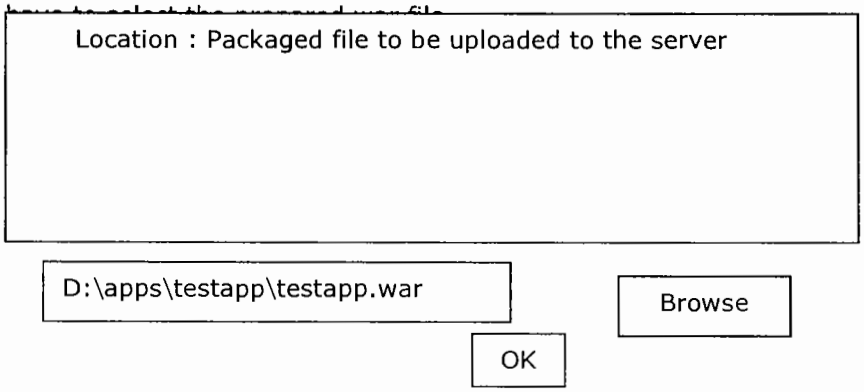
If we do the we above then Administration Console will be open, where we have to provide username and password.



Step 4: Deploy web application on GlassFish server.

If we click on Login button in Administration Console then automatically GlassFish server Home page will be open, where to deploy wed application go for Deployment section, where click on either List Deployed Application or Deploy an application.

If we click on Deploy an application automatically a browsing window will be open, where we have to select the prepared war file.



If we click on OK button, the specified war file will be uploaded to the server and the application context name will be displayed in the list of deployed applications.

NameEnabledEnginesAction

testapp √ web [Launch] [Redeploy] [Restart]

To access the web application we have to click on Launch hyper link under Action part.

If we click on Launch hyper link automatically a window will be open with the following url.

http://localhost:1111/testapp

If we write url pattern of the particular servlet then access the application

By using the following url.

http://localhost:1111/testapp/first

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.**

DURGASOFT

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.

1. req.getSession();
2. req.getSession(false);

DURGASOFT

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.

DURGASOFT

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.

DURGASOFT

<u>httpsessionapp:</u>-

<u>form1.html:</u>

```
<html>
<bodybgcolor="pink">
<center>
	<formmethod="get"action="first">
		Name : <inputtype="text"name="uname"/><br><br>
		Age : <inputtype="text"name="uage"/><br><br>
		<inputtype="submit"value="Next"/>
	</form>
</center>
</body>
</html>
```

<u>form2.html:</u>

```
<html>
<bodybgcolor="pink">
<center>
	<formmethod="get"action="second">
		Qualification : <inputtype="text"name="uqual"/><br><br>
		Designation : <inputtype="text"name="udesig"/><br><br>
		<inputtype="submit"value="Next"/>
	</form>
</center>
</body>
</html>
```

<u>form3.html:</u>

```
<html>
<bodybgcolor="pink">
<center>
	<formmethod="get"action="display">
		Email : <inputtype="text"name="email"/><br><br>
		Mobile : <inputtype="text"name="mobile"/><br><br>
		<inputtype="submit"value="Display"/>
	</form>
</center>
</body>
</html>
```

DURGASOFT

web.xml:

```
<web-app>
<display-name>httpsessionapp</display-name>
<welcome-file-list>
<welcome-file>form1.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>DisplayServlet</servlet-name>


<servlet-class>DisplayServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DisplayServlet</servlet-name>
<url-pattern>/display</url-pattern>
</servlet-mapping>
</web-app>
```

FirstServlet.java:

```
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;

publicclass FirstServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                String uname=request.getParameter("uname");
```


DURGASOFT

```
                String uage=request.getParameter("uage");
                HttpSession hs=request.getSession();
                hs.setAttribute("uname", uname);
                hs.setAttribute("uage", uage);
                RequestDispatcher rd=request.getRequestDispatcher("form2.html");
                rd.forward(request, response);
        }
}
```

SecondServlet.java:

```
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;

publicclass SecondServlet extends HttpServlet {

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

                String uqual=request.getParameter("uqual");
                String udesig=request.getParameter("udesig");
                HttpSession hs=request.getSession();
                hs.setAttribute("uqual", uqual);
                hs.setAttribute("udesig", udesig);
                RequestDispatcher rd=request.getRequestDispatcher("form3.html");
                rd.forward(request, response);
        }
}
```

DisplayServlet.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;
import javax.servlet.http.HttpSession;

publicclass DisplayServlet extends HttpServlet {

        protectedvoid doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String email=request.getParameter("email");
```


DURGASOFT

```
            String mobile=request.getParameter("mobile");
            HttpSession hs=request.getSession();
            String uname=(String)hs.getAttribute("uname");
            String uage=(String)hs.getAttribute("uage");
            String uqual=(String)hs.getAttribute("uqual");
            String udesig=(String)hs.getAttribute("udesig");
            out.println("<html>");
            out.println("<body bgcolor='lightgreen'>");
            out.println("<center><br><br>");
            out.println("<table bgcolor='lightyellow'>");
            out.println("<tr><td colspan='2'><center><b><font size='5'
color='red'><u>Registration Details</u></font></b></center></td></tr>");
            out.println("<tr><td>User Name</td><td>"+uname+"</td></tr>");
            out.println("<tr><td>User Age</td><td>"+uage+"</td></tr>");
            out.println("<tr><td>Qualification</td><td>"+uqual+"</td></tr>");
            out.println("<tr><td>Designation</td><td>"+udesig+"</td></tr>");
            out.println("<tr><td>Email</td><td>"+email+"</td></tr>");
            out.println("<tr><td>Mobile</td><td>"+mobile+"</td></tr>");
            out.println("<tr><td>Status</td><td>Success</td></tr>");
            out.println("</table></center>");
            out.println("</body>");
            out.println("</html>");
        }
}
```

2. Cookies Session Tracking Mechanism:

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 HttpSession Session Tracking Mechanism, all the clients conversations will be maintained at server machine in the form of HttpSession objects.

In HttpSession Session Tracking Mechanism, if we increase number of users then automatically number of HttpSession objects will be created at

server. So that HttpSession Session Tracking Mechanism will increase burden to server machine.

To overcome the above problem we have to use an alternative mechanism, where we have to manage all the clients conversations at the respective client machines only.

To achieve the above requirement we have to use **Cookies Session Tracking Mechanism.**

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.

DURGASOFT

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 clients previous data at the time of processing later request.

Drawbacks:

1. If we disable the Cookies at client browser then Cookies Session Tracking Mechanism will not execute its functionality.
2. 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.

cookiesapp:-

form1.html:

```
<html>
<bodybgcolor="lightgreen">
<center>
<formaction="./first">
<br><br><br>
<h1>Product Registration Form</h1><br><br>
<h2>
        Product Id <inputtype="text"name="pid"/><br><br>
        Product Name <inputtype="text"name="pname"/><br><br>
        <inputtype="submit"value="Next"/>
</h2>
</form></center></body>
</html>
```

form2.html:

```
<html>
<bodybgcolor="lightgreen">
<center>
<formaction="./second">
<br><br><br>
<h1>Product Registration Form(Cont..)</h1><br><br>
<h2>
        Product Cost <inputtype="text"name="pcost"/><br><br>
        Product Quantity <inputtype="text"name="pquantity"/><br><br>
        <inputtype="submit"value="Next"/>
</h2>
```


DURGASOFT

```
</form></center></body>
</html>
```

form3.html:

```
<html>
<bodybgcolor="lightgreen"><center>
<formaction="./reg">
<br><br><br>
<h1>Product Registration Form(Cont..)</h1><br><br>
<h2>
        Manufactured Date <inputtype="text"name="man_Date"/><br><br>
        Expiry Date <inputtype="text"name="exp_Date"/><br><br>
        <inputtype="submit"value="Registration"/>
</h2>
</form></center></body>
</html>
```

web.xml:

```
<web-app>
<display-name>cookiesapp</display-name>
<welcome-file-list>
<welcome-file>form1.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>RegistrationServlet</servlet-name>
<servlet-class>RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegistrationServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
```

DURGASOFT

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

FirstServlet.java:

```
import java.io.IOException;

import javax.servlet.RequestDispatcher;
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 doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
                String pid=request.getParameter("pid");
                String pname=request.getParameter("pname");


                Cookie c1=new Cookie("pid", pid);
                Cookie c2=new Cookie("pname", pname);
                response.addCookie(c1);
                response.addCookie(c2);
                RequestDispatcher rd=request.getRequestDispatcher("form2.html");
                rd.forward(request, response);
        }
}
```

SecondServlet.java:

```
import java.io.IOException;

import javax.servlet.RequestDispatcher;
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 doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
                try{
                        int pcost=Integer.parseInt(request.getParameter("pcost"));
                        int
pquantity=Integer.parseInt(request.getParameter("pquantity"));
                        Cookie c3=new Cookie("pcost", ""+pcost);
                        Cookie c4=new Cookie("pquantity", ""+pquantity);
```


DURGASOFT

```
                        response.addCookie(c3);
                        response.addCookie(c4);
                        RequestDispatcher
        rd=request.getRequestDispatcher("form3.html");
                        rd.forward(request, response);
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

RegistrationServlet.java:

```
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 RegistrationServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String man_Date=request.getParameter("man_Date");
                String exp_Date=request.getParameter("exp_Date");
                Cookie[] c=request.getCookies();
                String pid=c[0].getValue();
                String pname=c[1].getValue();
                int pcost=Integer.parseInt(c[2].getValue());
                int pquantity=Integer.parseInt(c[3].getValue());
                ProductBean pb=new ProductBean();
                String
status=pb.register(pid,pname,pcost,pquantity,man_Date,exp_Date);
                out.println("<html>");
                out.println("<body bgcolor='pink'>");
                out.println("<center><br><br>");
                out.println("<u>Product Registration Details</u><br><br>");
                out.println("Product Id....."+pid+"<br><br>");
                out.println("Product Name....."+pname+"<br><br>");
                out.println("Product Cost....."+pcost+"<br><br>");
                out.println("Product Quantity....."+pquantity+"<br><br>");
                out.println("Product Manufactured
Date....."+man_Date+"<br><br>");
                out.println("Product Expiry Date....."+exp_Date+"<br><br>");
                out.println("Status....."+status);
```


DURGASOFT

```
                out.println("</center>");
                out.println("</body>");
                out.println("</html>");
        }
}
```

ProductBean.java:

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

public class ProductBean {
        Connection con;
        Statement st;
        ResultSet rs;
        String status="";
        public ProductBean(){
                try{
                        Class.forName("oracle.jdbc.driver.OracleDriver");


        con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","sy
stem","durga");
                        st=con.createStatement();
                }
                catch (Exception e) {
                        e.printStackTrace();
                }
        }
        public String register(String pid,String pname,int pcost,int pquantity,String
man_Date,String exp_Date){
                try{
                        rs=st.executeQuery("select * from product where
pid='"+pid+"'");
                        boolean b=rs.next();
                        if(b==true){
                                status="Product Existed Already";
                        }
                        else{
                                int rowCount=st.executeUpdate("insert into product
values('"+pid+"','"+pname+"',"+pcost+","+pquantity+",'"+man_Date+"','"+exp_Da
te+"')");
                                if(rowCount==1){
                                        status="SUCCESS";
                                }
                                else{
                                        status="FAILURE";
                                }
                        }
                }
                catch (Exception e) {
```


DURGASOFT

```
                        status="FAILURE";
                        e.printStackTrace();
                    }
                    return status;
            }
}
```

To represent a Cookie in web application Servlet API has provided a predefined class in the form of javax.servlet.http.Cookie.

To create Cookie object with a particular name-value pair we have to use the following Cookie class constructor.

```
public 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)
```


DURGASOFT

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()

3. 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.

DURGASOFT

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.

urlrewritingapp:-

form1.html:

```
<html>
<bodybgcolor="lightgreen">
        <br><br><center><h1>Deposit Form</h1></center>
        <br><br><hr><br><br>
        <formaction="./first">
                <pre>
                        Account Number <inputtype="text"name="accNo"/>

                        Account Name   <inputtype="text"name="accName">

                        <inputtype="submit"value="Next">
                </pre>
        </form></body>
</html>
```

web.xml:

DURGASOFT

```
<web-app>
<display-name>urlrewritingapp</display-name>
<welcome-file-list>
<welcome-file>form1.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>DepositServlet</servlet-name>
<servlet-class>DepositServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>DepositServlet</servlet-name>
<url-pattern>/deposit</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;
import javax.servlet.http.HttpSession;

public class FirstServlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		PrintWriter out=response.getWriter();
		String accNo=request.getParameter("accNo");
		String accName=request.getParameter("accName");
		HttpSession session=request.getSession();
		session.setAttribute("accNo", accNo);
```


DURGASOFT

```
                session.setAttribute("accName", accName);
                out.println("<html><body bgcolor='cyan'><br><br>");
                out.println("<center><h1>Deposit
Form(Cont..)</h1></center><br><br><hr><br><br>");
                out.println("<form method='get'
action='"+response.encodeUrl("./second")+"'>");
                out.println("<pre>");
                out.println("Account Type    <input type='text' name='accType'/>");
                out.println();
                out.println("Account Branch <input type='text' name='accBranch'/>");
                out.println();
                out.println("  <input type='submit' value='Next'/>");
                out.println("</pre></form></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;
import javax.servlet.http.HttpSession;

public class SecondServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                String accType=request.getParameter("accType");
                String accBranch=request.getParameter("accBranch");
                HttpSession session=request.getSession();
                session.setAttribute("accType", accType);
                session.setAttribute("accBranch", accBranch);
                out.println("<html><body bgcolor='cyan'><br><br>");
                out.println("<center><h1>Deposit
Form(Cont..)</h1></center><br><br><hr><br><br>");
                out.println("<form method='get'
action='"+response.encodeUrl("./deposit")+"'>");
                out.println("<pre>");
                out.println("Depositor Name <input type='text' name='depName'/>");
                out.println();
                out.println("Deposit Amount <input type='text' name='depAmount'/>");
                out.println();
                out.println("  <input type='submit' value='Deposit'/>");
                out.println("</pre></form></body></html>");

        }
}
```


DURGASOFT

DepositServlet.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;
import javax.servlet.http.HttpSession;

public class DepositServlet extends HttpServlet {


	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		String depName=request.getParameter("depName");
		int depAmount=Integer.parseInt(request.getParameter("depAmount"));
		HttpSession session=request.getSession();
		String accNo=(String)session.getAttribute("accNo");
		String accName=(String)session.getAttribute("accName");
		String accType=(String)session.getAttribute("accType");
		String accBranch=(String)session.getAttribute("accBranch");
		TransactionBean tb=new TransactionBean();
		String status=tb.deposit(accNo,depAmount);
		out.println("<html><body bgcolor='lightblue'><br><br>");
		out.println("<center><table>");
		out.println("<tr><td colospan='2'><center><b><font size='6'
color='red'>");
		out.println("<u>Transaction
Details</u><font></b></center></td></tr>");
		out.println("<tr><td>Transaction Id</td><td>:t1</td></tr>");
		out.println("<tr><td>Transaction Name</td><td>:Deposit</td></tr>");
		out.println("<tr><td>Account Number</td><td>:"+accNo+"</td></tr>");
		out.println("<tr><td>Account Name</td><td>:"+accName+"</td></tr>");
		out.println("<tr><td>Account Type</td><td>:"+accType+"</td></tr>");
		out.println("<tr><td>Account
Branch</td><td>:"+accBranch+"</td></tr>");
		out.println("<tr><td>Depositor
Name</td><td>:"+depName+"</td></tr>");
		out.println("<tr><td>Deposit
Amount</td><td>:"+depAmount+"</td></tr>");
		int totalAvailableAmount=tb.getTotalAvailableAmount();
		out.println("<tr><td>Total Available
Amount</td><td>:"+totalAvailableAmount+"</td></tr>");
		out.println("<tr><td>Transaction
Status</td><td>:"+status+"</td></tr>");
		out.println("<tr><td colspan='2'><b><font size='5'>...Visit
Again...</td></tr>");
		out.println("</table></center></body></html>");
	}
```

```
}
```

TransactionBean.java:

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

public class TransactionBean {

	Connection connection;


	Statement statement;
	ResultSet resultSet;
	String status="";
	int totalAvailableAmount;

	TransactionBean(){
		try{
			Class.forName("oracle.jdbc.driver.OracleDriver");

	connection=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe",
"system", "durga");
			statement=connection.createStatement();
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}
	public String deposit(String accNo,int depAmount){
		try{
			int rowCount=statement.executeUpdate("update acc_details set
balance=balance+"+depAmount+"where accNo='"+accNo+"'");
			if(rowCount==1)
				status="SUCCESS";
			else
				status="FAILURE";
		}
		catch (Exception e) {
			status="FAILURE";
			e.printStackTrace();
		}
		return status;
	}
	public int getTotalAvailableAmount() {
		try{
			resultSet=statement.executeQuery("select * from acc_details");
			resultSet.next();
			totalAvailableAmount=resultSet.getInt(4);
```


DURGASOFT

```
            }
            catch (Exception e) {
                    e.printStackTrace();
            }
            return totalAvailableAmount;
        }
}
```

4. 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.

hiddenformfieldsapp:-

studentform.html:-

```
<html>
<head>
<bodybgcolor="lightgreen">
        <formmethod="get"action="./first">
        <center><b><br><br>
        Student Name:<inputtype="text"name="sname"/><br><br>
        Student Id:<inputtype="text"name="sid"/><br><br>
        Student Address:<inputtype="text"name="saddr"/><br><br>
        <inputtype="submit"value="Submit">
        </b></center>
        </form>
</body>
</html>
```

web.xml:-

DURGASOFT

```
<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'>");
```


DURGASOFT

```
                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;
```


DURGASOFT

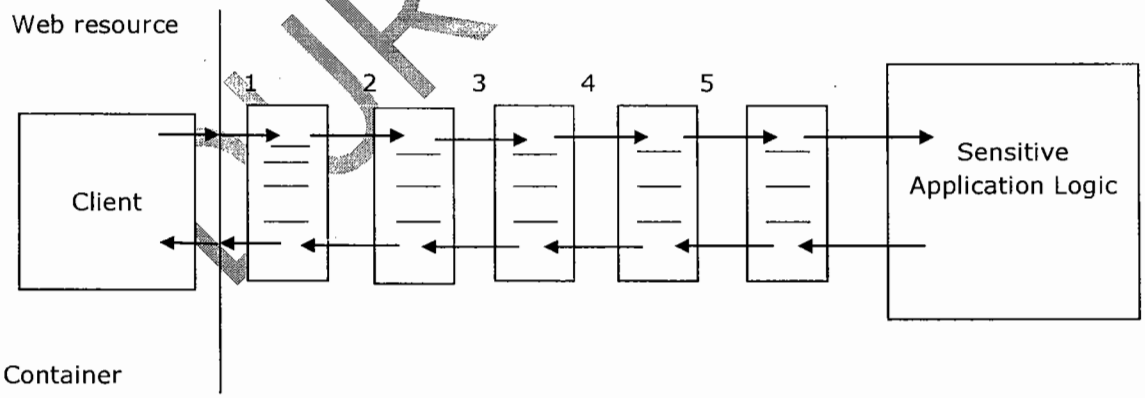
```
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

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>");
        }
}
```

Filters:

In general in web application development, we will provide the complete application logic in the form of the web resources like servlets, jsps and so on.

As part of the web application development some web resources may require the services like Authentication, Authorization, Security, Data compression and decompression and so on as preprocessing and post processing.

In the above context, to implement all the above preprocessing and post processing services Servlet API has provided a separate component called **Filter.**



1. Authentication Filter 2. Authorization Filter 3. Security Filter

4. Compression and Decompression Filter 5. Encoding and Decoding Filter

→ Preprocessing ← Post processing

DURGASOFT

From the above representation when we send a request from client to server for a particular web resource then container will pick up that request, container will check whether any Filter is associated with the respective web resource, if container identify any Filter or Filters then container will execute that Filters first.

While executing a Filter if the present request is satisfied all the Filter constraints then only container will bypass that request to next Filter or next web resource.

If the present request is not satisfied the Filter constraints then container will generate the respective response to client.

Filter is a server side component, it will be executed by the container automatically when it receives request from client for a particular web resource.

If we want to use Filters in our web applications, we have to use the following steps.

Step 1: Prepare Filter class.

Step 2: Configure Filter class in web.xml file.

Step 1: Prepare Filter class:

Filter is an object available at server machine, it must implement Filter interface either directly or indirectly.

```
public interface Filter  {

public void init(FilterConfig config)throws ServletException;

public void doFilter(ServletRequest req, ServletResponse res, FilterChain fc)throws SE, IOE

    public void destroy(); }

public class MyFilter implements Filter { -----}
```

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.

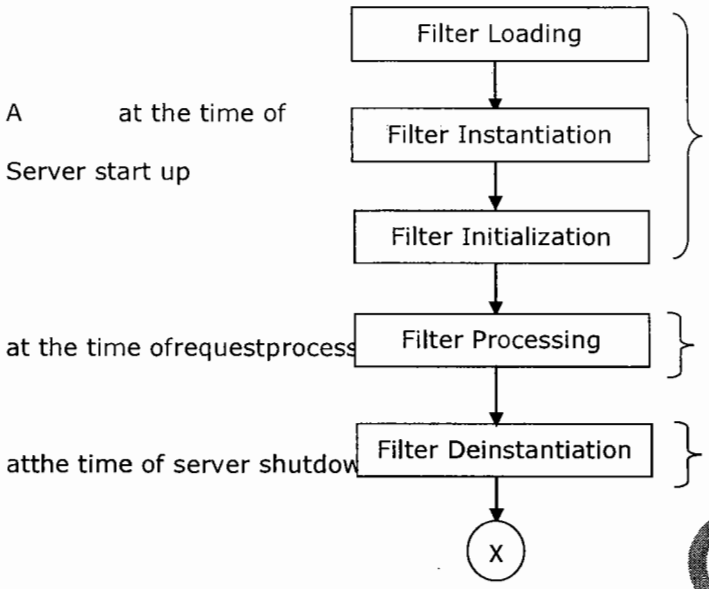
Where destroy() method can be used to perform Filter Deinstantiation.

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.

DURGASOFT



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>
```


DURGASOFT

```
</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.

filterapp:-

studentform.html:-

```
<html><bodybgcolor="lightgreen">
<center><br><br>
        <formmethod="get"action="./filter">
        <tablebgcolor="pink"border="2">
        <tr>
                <td>Student Name</td>
                <td><inputtype="text"name="sname"/></td>
        </tr>
        <tr>
                <td>Student Age</td>
                <td><inputtype="text"name="sage"/></td>
        </tr>
        <tr>
                <td>Student Address</td>
                <td><inputtype="text"name="saddr"/></td>
        </tr>
        <tr>
                <td><inputtype="submit"value=""/></td>
                <td><inputtype="submit"value="SUBMIT"/></td>
        </tr>
        </table></form></center>
</body></html>
```

DURGASOFT

web.xml:-

```
<web-app>
<display-name>filterapp</display-name>
<welcome-file-list>
<welcome-file>studentform.html</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/filter</url-pattern>
</servlet-mapping>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/filter</url-pattern>
</filter-mapping>
</web-app>
```

MyFilter.java:-

```
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 MyFilter implements Filter {

   public MyFilter() {

   }

        public void destroy() {

        }

        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
                PrintWriter out=response.getWriter();
                out.println("<body bgcolor='yellow'>");
                out.println("<center><b><br><br>");
```

DURGASOFT

```
                int age=Integer.parseInt(request.getParameter("sage"));
                if(age<18)
                {
                        out.println("U R not allowed for this Fashion Show");
                        out.println("<br><br>");
                        out.println("Bcoz, U R minor");
                        out.println("<br><br>");
                        out.println("Try after few years");
                        out.println("<br><br>");
                        out.println("<a href='/filterapp/studentform.html'>New Student
Form</a>");

                }
                else{
                        chain.doFilter(request, response);
                }
        }

        public void init(FilterConfig fConfig) throws ServletException {

        }

}
```

MyServlet.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 MyServlet extends HttpServlet {

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

                PrintWriter out=response.getWriter();
                String sname=request.getParameter("sname");
                int sage=Integer.parseInt(request.getParameter("sage"));
                String saddr=request.getParameter("saddr");
                out.println("<br><br>");
                out.println("Hello Student..... Ur details are....");
                out.println("<br><br>");
                out.println("Name....."+sname);
                out.println("<br><br>");
                out.println("Name....."+sage);
                out.println("<br><br>");
                out.println("Name....."+saddr);
```


DURGASOFT

```
                out.println("<br><br>");
                out.println("Enjoy the Fashion show");
        }

}
```

ValidationsApp

Registrationform.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Strict//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<font color="red">
<h2>Durga Software Solutions</h2>
<h3>Employee Registration Form</h3>
</font>
<form method="post" action="./reg">
<table>
<tr>
        <td>Employee Id</td>
        <td><input type="text" name="eid"/></td>
</tr>
<tr>
        <td>Employee Name</td>
        <td><input type="text" name="ename"/></td>
</tr>
<tr>
        <td>Employee Age</td>
        <td><input type="text" name="eage"/></td>
</tr>
<tr>
        <td>Employee Email</td>
        <td><input type="text" name="eemail"/></td>
</tr>
<tr>
        <td>Employee Mobile</td>
        <td><input type="text" name="emobile"/></td>
</tr>
<tr>
        <td><input type="submit" value="Registration"/></td>
</tr>
</table>
</form>
</body>
</html>
```

Web.xml

DURGASOFT

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
<display-name>filtersapp</display-name>
<welcome-file-list>
<welcome-file>registrationform.html</welcome-file>
</welcome-file-list>
<servlet>
<description></description>
<display-name>RegistrationServlet</display-name>
<servlet-name>RegistrationServlet</servlet-name>
<servlet-class>com.durgasoft.RegistrationServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>RegistrationServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
<filter>
<display-name>RegistrationFilter</display-name>
<filter-name>RegistrationFilter</filter-name>
<filter-class>com.durgasoft.RegistrationFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>RegistrationFilter</filter-name>
<url-pattern>/reg</url-pattern>
</filter-mapping>
</web-app>
```

RegistrationFilter.java

```
package com.durgasoft;
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 RegistrationFilter implements Filter {
String eid_err_msg="",
ename_err_msg="",
eage_err_msg="",
eemail_err_msg="",
emobile_err_msg="";
        boolean b=true;
        public void destroy() {
        }

        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
                try {
```


DURGASOFT

```
response.setContentType("text/html");
PrintWriter out=response.getWriter();
String eid=request.getParameter("eid");
String ename=request.getParameter("ename");
String eage=request.getParameter("eage");
String eemail=request.getParameter("eemail");
String emobile=request.getParameter("emobile");
if(eid == null || eid.equals("")){
        eid_err_msg="Employee Id is Required";
        b=false;
}else{
        if(!eid.startsWith("DSS-")){
        eid_err_msg="Employee Id Must be in DSS-ddd Format";
                b=false;
        }else{
                b=true;
        }
}
if(ename == null || ename.equals("")){
        ename_err_msg="Employee Name is Required";
        b=false;
}else{
        b=true;
}
if(eage == null || eage.equals("")){
        eage_err_msg="Employee Age is Required.";
        b=false;
}else{
        int age=Integer.parseInt(eage);
        if(age<20 || age>30){
eage_err_msg="Employee Age Must be in between 20 to 30 Years.";
                b=false;
        }else{
                b=true;
        }
}
if(eemail == null || eemail.equals("")){
        eemail_err_msg="Employee Email is Required";
        b=false;
}else{
        if(!eemail.endsWith("@durgasoft.com")){
                eemail_err_msg="Employee Email is Invalid";
                b=false;
        }else{
                b=true;
        }
}

if(emobile == null || emobile.equals("")){
        emobile_err_msg="Employee Mobile is Required";
        b=false;
}else{
        if(!emobile.startsWith("91-")){
```


DURGASOFT

```
                                emobile_err_msg="Employee Mobile Num. is Invalid";
                                b=false;
                        }else{
                                b=true;
                        }
                }
                if(b==false){
                out.println("<html>");
                out.println("<body>");
                out.println("<font color='red'>");
                out.println("<h2>Durga Software Solutions</h2>");
                out.println("<h3>Employee Registration Form</h3>");
                out.println("</font>");
                out.println("<font color='blue' size='5'>");
                out.println(eid_err_msg+"<br>");
                out.println(ename_err_msg+"<br>");
                out.println(eage_err_msg+"<br>");
                out.println(eemail_err_msg+"<br>");
                out.println(emobile_err_msg+"<br>");
                out.println("</font>");
                out.println("<form method='post' action='./reg'>");
                out.println("<table>");
                out.println("<tr><td>Employee Id</td><td><input type='text'
name='eid' value='"+eid+"'/></td><td>"+eid_err_msg+"</td></tr>");
                out.println("<tr><td>Employee Name</td><td><input type='text'
name='ename' value='"+ename+"'/></td><td>"+ename_err_msg+"</td></tr>");
                out.println("<tr><td>Employee Age</td><td><input type='text'
name='eage' value='"+eage+"'/></td><td>"+eage_err_msg+"</td></tr>");
                out.println("<tr><td>Employee Email</td><td><input type='text'
name='eemail' value='"+eemail+"'/></td><td>"+eemail_err_msg+"</td></tr>");
                out.println("<tr><td>Employee Mobile</td><td><input type='text'
name='emobile' value='"+emobile+"'/></td><td>"+emobile_err_msg+"</td></tr>");
                out.println("<tr><td><input type='submit'
value='Registration'/></td></tr>");
                out.println("</table></form></body></html>");
                eid_err_msg="";
                ename_err_msg="";
                eage_err_msg="";
                eemail_err_msg="";
                emobile_err_msg="";
                b=true;
                }else{
                        chain.doFilter(request, response);
                }
        } catch (Exception e) {
                e.printStackTrace();
        }

    }
    public void init(FilterConfig fConfig) throws ServletException {
    }

}
```


DURGASOFT

RegistrationServlet.java

```
package com.durgasoft;
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 RegistrationServlet extends HttpServlet {
	private static final long serialVersionUID = 1L;
	protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		try {
			String eid=request.getParameter("eid");
			String ename=request.getParameter("ename");
			int eage=Integer.parseInt(request.getParameter("eage"));
			String eemail=request.getParameter("eemail");
			String emobile=request.getParameter("emobile");
			out.println("<html>");
			out.println("<body>");
			out.println("<font color='red'>");
			out.println("<h2>Durga Software Solutions</h2>");
			out.println("<h3>Employee Registration Details</h3>");
			out.println("</font>");
			out.println("<table border='1'>");
			out.println("<tr><td>Employee Id</td><td>"+eid+"</td></tr>");
			out.println("<tr><td>Employee
Name</td><td>"+ename+"</td></tr>");
			out.println("<tr><td>Employee
Age</td><td>"+eage+"</td></tr>");
			out.println("<tr><td>Employee
Email</td><td>"+eemail+"</td></tr>");
			out.println("<tr><td>Employee
Mobile</td><td>"+emobile+"</td></tr>");
			out.println("</table></body></html>");
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

}
```

Servlet Wrappers:

The main purpose of Servlet Wrappers in web applications is to simplify the customization of request and response objects.

In general in web application execution when we send a request from client to server for a particular servlet then container will pick up the request, identify the requested servlet, perform servlet life cycle stages like servlet loading, servlet instantiation, servlet initialization, request processing and servlet deinstantiation.

As part of request processing container has to access service(_,_) method, for this container has to prepare servlet request and servlet response objects.

As part of application requirement we need to use our own request and response objects in the servlets instead of using container provided request and response objects.

To achieve this requirement we have to perform the customization of request and response objects.

In general Servlet is an abstraction i.e. collection of interfaces provided by Sun Micro Systems and whose implementations could be provided by all the server vendors.

Similarly if we want to customize request and response objects we have to implement ServletRequest and ServletResponse interfaces by taking implementation classes.

```
public class MyRequest implements ServletRequest
{ ---------
  --------- }
public class MyResponse implements ServletResponse
  { ---------
    --------- }
```

To perform request and response objects coustomization if we use the above approach then we must implement directly ServletRequest and ServletResponse interfaces i.e. we must provide implementation for each and every method declared in ServletRequest and ServletResponse interfaces. This approach will increase burden to the developers.

To overcome the above problem Servlet API has provided an alternative in the form of a set of predefined classes called **Wrapper classes**.

All the Servlet Wrapper classes are direct implementation classes to the respective ServletRequest and ServletResponse and so on.

Servlet Wrapper classes is an idea provided by a design pattern called as **Adapter Design Pattern.**

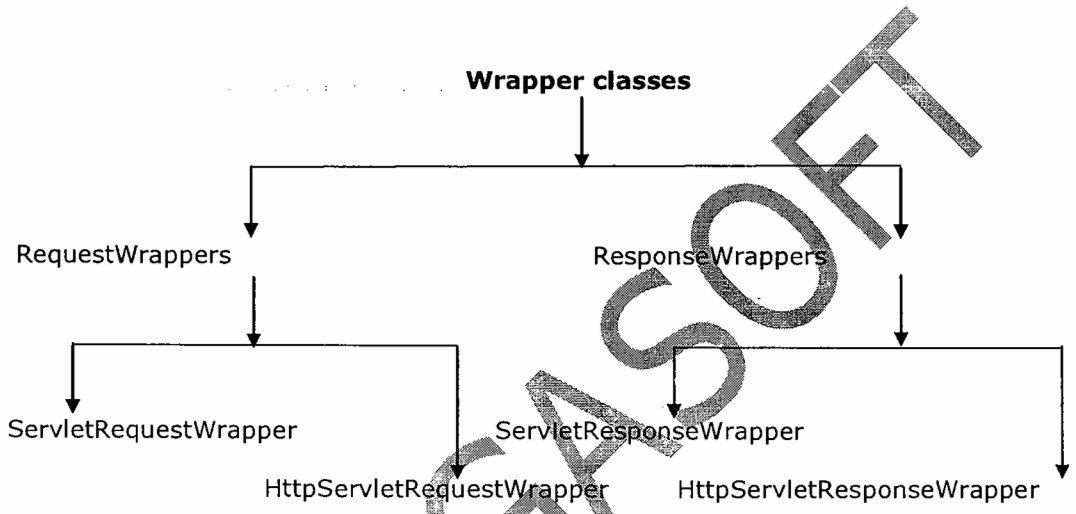
If we want to prepare our own request and response classes we have to extend the respective Servlet Wrapper class and override required method.

DURGASOFT

```
public class MyClass extends Xxxxx
{ --------
  -------- }
```

Where Xxxxx may be a particular Wrapper class.

There are 2 types of Wrapper classes in Servlet API.



To customize the request and response objects we have to use a Filter in order to prepare our own request and response objects and to pass our own request and response objects to the service(_,_) method.

requestwrapperapp:-

registrationform.html:-

```
<html>
<bodybgcolor="lightgreen">
        <center><h1><u>Registration Form</u></h1></center>
        <b><fontsize="7">
        <formmethod="get"action="./reg">
        <pre>
                Name      <inputtype="text"name="uname"/>

                Password  <inputtype="password"name="upwd"/>

                Email     <inputtype="text"name="email"/>@dss.com

                Mobile    <inputtype="text"name="mobile"/>
```


DURGASOFT

```
                <inputtype="submit"value="REGISTRATION"/>
        </pre>
        </form>
        </font></b>
</body>
</html>
```

web.xml:-

```
<web-app>
<display-name>requestwrapperapp</display-name>
<welcome-file-list>
<welcome-file>registrationform.html</welcome-file>
</welcome-file-list>

<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/reg</url-pattern>
</filter-mapping>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/reg</url-pattern>
</servlet-mapping>
</web-app>
```

MyFilter.java:-

```
import java.io.IOException;
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 MyFilter implements Filter {

   public void destroy() {

        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {

                MyRequest myRequest=new MyRequest(request);
                chain.doFilter(myRequest, response);
```


DURGASOFT

```
        }
        public void init(FilterConfig fConfig) throws ServletException {

        }

}
```

MyRequest.java:-

```
import javax.servlet.ServletRequest;
import javax.servlet.ServletRequestWrapper;

public class MyRequest extends ServletRequestWrapper {


        ServletRequest request;

        public MyRequest(ServletRequest request){
                super(request);
                this.request=request;
        }
        public String getParameter(String name){
                String value=request.getParameter(name);
                if(name.equals("email")){
                        if(!value.endsWith("@dss.com")){
                                value=value+"@dss.com";
                        }
                }
                return value;
        }
}
```

MyServlet.java:-

```
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 MyServlet extends GenericServlet {

        public void service(ServletRequest request, ServletResponse response)throws
ServletException,IOException{

                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String uname=request.getParameter("uname");
                String upwd=request.getParameter("upwd");
                String email=request.getParameter("email");
                String mobile=request.getParameter("mobile");
```

DURGASOFT

```
            out.println("<html><body bgcolor='lightyellow'>");
            out.println("<b><font size='7'><br><br>");
            out.println("Name......"+uname+"<br><br>");
            out.println("Password......"+upwd+"<br><br>");
            out.println("Email......"+email+"<br><br>");
            out.println("Mobile......"+mobile);
            out.println("</font></b></body></html>");

        }
}
```

responsewrapperapp1:-

web.xml:-

```
<web-app>
<display-name>responsewrapperapp1</display-name>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/wrapper</url-pattern>
</filter-mapping>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/wrapper</url-pattern>
</servlet-mapping>
</web-app>
```

MyFilter.java:-

```
import java.io.IOException;
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.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class MyFilter implements Filter {
```

DURGASOFT

```
public void init(FilterConfig fConfig) throws ServletException {}

public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
throws IOException, ServletException {

		HttpServletRequest httpRequest=(HttpServletRequest)request;
		HttpServletResponse httpResponse=(HttpServletResponse)response;
		MyResponse myResponse=new MyResponse(httpResponse);
		chain.doFilter(httpRequest, myResponse);
	}
	public void destroy() { }

}
```

MyResponse.java:-

```
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;

publicclass MyResponse extends HttpServletResponseWrapper {

	HttpServletResponse httpResponse;

	public MyResponse(HttpServletResponse httpResponse){
		super(httpResponse);
		this.httpResponse=httpResponse;
	}
	publicvoid setContentType(String type){
		if(!type.equals("text/html")){
			httpResponse.setContentType("text/html");
		}
	}
}
```

MyServlet.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 MyServlet extends HttpServlet {

	protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
		response.setContentType("img/jpg");
		PrintWriter out=response.getWriter();
		out.println("<h1>Hello..... from MyServlet</h1>");
```


DURGASOFT

```
        }

}
```

responsewrapperapp2:-

reverseform.html:-

```
<html>
<bodybgcolor="lightgreen">
        <formmethod="get"action="./wrapper">
                <center><b><fontsize="6"><br><br><br>
                        Enter Text : <inputtype="text"name="text"/>
                        <br><br>
                        <inputtype="submit"value="Reverse"/>
                </font></b></center>
        </form>
</body>
</html>
```

web.xml:-

```
<web-app>
<display-name>responsewrapperapp2</display-name>
<welcome-file-list>
<welcome-file>reverseform.html</welcome-file>
</welcome-file-list>
<filter>
<filter-name>MyFilter</filter-name>
<filter-class>MyFilter</filter-class>
</filter>
<filter-mapping>
<filter-name>MyFilter</filter-name>
<url-pattern>/wrapper</url-pattern>
</filter-mapping>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>MyServlet</servlet-name>
<url-pattern>/wrapper</url-pattern>
</servlet-mapping>
</web-app>
```

DURGASOFT

MyFilter.java:-

```
import java.io.IOException;
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.http.HttpServletRequest;


import javax.servlet.http.HttpServletResponse;

public class MyFilter implements Filter {

   public MyFilter() {

   }
   public void init(FilterConfig fConfig) throws ServletException {

        }
        public void destroy() {

        }
        public void doFilter(ServletRequest request, ServletResponse response, FilterChain
chain) throws IOException, ServletException {
                HttpServletRequest httpRequest=(HttpServletRequest)request;
                HttpServletResponse httpResponse=(HttpServletResponse)response;
                MyResponse myResponse=new MyResponse(httpResponse);
                chain.doFilter(httpRequest, myResponse);
        }

}
```

MyResponse.java:-

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

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;


publicclass MyResponse extendsHttpServletResponseWrapper{
        HttpServletResponse httpServletResponse;
        public MyResponse(HttpServletResponse httpServletResponse){
                super(httpServletResponse);
                this.httpServletResponse=httpServletResponse;
        }
        public PrintWriter getWriter() throws IOException{
                PrintWriter out=httpServletResponse.getWriter();
                MyWriter myWriter=new MyWriter(out);
```


DURGASOFT

```
                return myWriter;
        }
}
```

MyWriter.java:-

```
import java.io.PrintWriter;

publicclass MyWriter extends PrintWriter {
        PrintWriter out;
        public MyWriter(PrintWriter out){
                super(out);
                this.out=out;
        }
        publicvoid println(String data){
                if(!data.startsWith("<")){
                        StringBuffer sb=new StringBuffer(data);
                        out.println(sb.reverse());
                }else{
                        out.println(data);
                }
        }
}
```

MyServlet.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 MyServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                String data=request.getParameter("text");
                out.println("<html><body bgcolor='lightblue'>");
                out.println("<center><b><font size='7' color='red'>");
                out.println("<br><br>");
                out.println(data);
                out.println("</font></b></center></body></html>");
        }

}
```

DURGASOFT

Servlet Listeners:

In GUI applications, when we click on a button, when we select an item in checkboxes, when we select an item in the list, choice boxes automatically the respective GUI components may raise the respective events.

In GUI applications, all the GUI components are capable of raising the events only, they are not capable of handling the events.

In the above context, to handle the events all the GUI components will bypass the generated events to a separate implicit component called as **Listener.**

The main role of Listener in GUI applications is to listen an event from the respective GUI component, handle it by executing Listener methods and send back the response to GUI application.

The above process in GUI applications is treated as **Event Handling** or **Event Delegation Model.**

In GUI applications, all the events are represented by some predefined classes and all the Listeners are represented in the form of interfaces.

Similarly in web application execution, container may generate events at the time of creating request object, adding an attribute, replacing an attribute, removing an attribute and destroying request object.

In this context, to listen the events generated by the container and to handle that events Servlet API has provided a set of interfaces called as Servlet Listeners.

Therefore, the main purpose of **Servlet Listeners** is to read all the life cycle stages of request object, ServletContext object and HttpSession objects.

In web applications, to perform event handling Servlet API has provided the following predefined library.

Where ServletRequestListener, ServletContextListener and HttpSessionListener can be used to read the life cycle stages like creation, destruction of ServletRequest object, ServletContext object and HttpSession object.

Where ServletRequestAttributeListener, ServletContextAttributeListener and HttpSessionAttributeListener can be used to read the life cycle stages of request object, context object and session object like adding a attribute, replacing an attribute and removing an attribute.

Where HttpSessionBindingListener can be used to read the life cycle stages of HttpSession object like adding HttpSessionBindingListener implementation class object

DURGASOFT

reference to HttpSession object and eliminating HttpSessionBindingListener implementation class object reference from HttpSession object.

Where HttpSessionActivationListener can be used to read the life cycle stages of HttpSession object like participating HttpSession object in Serialization process of Marshalling and participating HttpSession object in Deserialization process of Unmarshalling in RMI applications i.e. Distributed applications.

If we want to use Listeners in our web applications then we have to use the following steps.

Step 1: Prepare Listener class.

Here take one user defined class, it must be an implementation class to Listener interface.

Ex: public class MyListener implements Listener {-----}

Step 2: Configure Listener classin web.xml file.

To configure Listener class in web.xml file we have to use the following xml tags.

```
<web-app>
            -------------
<listener>
<listener-class>Fully Qualified name of Listener class</listener-class>
</listener>
            -------------
</web-app>
```

The above Listener configuration is required for all Listeners except HttpSessionBindingListener and HttpSessionActivationListener.

listenerapp:-

web.xml:-

```
<web-app>
<display-name>listenerapp</display-name>
<listener>
<listener-class>HitCountListener</listener-class>
```


DURGASOFT

```
</listener>
<servlet>
<servlet-name>MyServlet</servlet-name>
<servlet-class>MyServlet</servlet-class>
</servlet>



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

HitCountListener.java:-

```
import javax.servlet.ServletContext;
import javax.servlet.ServletRequestEvent;
import javax.servlet.ServletRequestListener;

public class HitCountListener implements ServletRequestListener {
        int count=0;
         public void requestInitialized(ServletRequestEvent e) {
              System.out.println("Request Object Created");
        }
public void requestDestroyed(ServletRequestEvent e) {
count=count+1;
ServletContext context=e.getServletContext();
context.setAttribute("count", count);
System.out.println("Request Object Destroyed");
}

}
```

MyServlet.java:-

```
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;

public class MyServlet extends HttpServlet {

        protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                ServletContext context=getServletConfig().getServletContext();
                out.println("<center><h1>Hit Count is....."
```


DURGASOFT

```
+context.getAttribute("count")+"</h1></center>");
        }

}
```

SERVLETS INTERVIEW QUESTIONS:

1.In How many ways we can develop a servlet?

2.What is the difference between CGI & Servlet?

3.What is the difference between ServletConfig & ServletContext?

4.How to define a Servlet?

5.Specify parallel technologies to the servlet?

6.What is the difference between web server & web container?

7.What is the difference between web server & application server?

8.What are the various types of web containers?

9.By using which 2 Packages we can implement Servlet?

10.What is the purpose of RequestDispatcher?

11.What methods we can call on RequesrDispatcher Object?

12.Explain about SingleThreadModeal?

13.By using which interfaces we can implements Filter concept?

14.What are the various listeners are available in Servlet specification?

15.What are the various Event classes present in Servlet Specification?

16.By using which object we can send text data as response from the Servlet?

17.By using which object we can read binary data send by the client?

18.By using which object we can send binary data send as response from the Servlet?

19.What are the (various) lifecycle methods of the Servlet?

DURGASOFT

20.Explain the purpose of the init() method & How many times it will be excuted? When it will be executed?

21.If init() method throws any Exception i.e; if init() method fails to execute then what will be happen?

22.Is it possible to write constructor with in the Servlet?

23.Why init() is required to perform initialization activities instead of constructor?

24.Is it possible to perform Initialization activities with in the constructor?

25.What is the purpose of service() & How Many times it will be Executed?

26.Explain about destroy?

27.Is it Possible to call destroy() method explicitly from service?

28. With in the <servlet-mapping> how many url pattrerns taken at a time?

29.Explain LifeCycle of the Servlet?

30.What is the purpose of <load-on-startup>?

31.What is the significance of the number for <load-on-startup>?

32.If two servlets having same <load-on-startup>value then which wii be loaded first?

33.Explain about GenericServlet?

34.Which interfaces are implemented by GenericServlet?

35.What is the necessity of having 2 init() methods with in the Servlet?

36.Explain best way of overriding init()?

37.What are various possible status code of response?

38.Explain the difference between GET&POST?

39.What are various HttpRequest methods?

40.What is the difference between HEAD&GET?

41.What is the difference between PUT&GPOST?

42.Which Http methods are non-idempotent?

43.Which Http methods are idempotent?

44.What is the default method for the form?

DURGASOFT

45.How many service() methods available in HttpServlet?

46.Is it recommended to override service() in Http based Servlet?

47.If you are sending Get request but our Servlet contains doGet() & service() Methods then which method will be executed?

48.If you are sending Get request but our Servlet doesn't contain doGet() what happen?

49.Even though HttpServlet doesn't contain any abstract method why it is declared as abstract class?

50.What are the various methods to retrieve from parameters?

51.What is the purpose of request Headers?

52.How we can retrieve headers associated with the ServletRequest?

53. To what value is a variable of the String type automatically initialized?How we can retrieve cookies from the request?

54.By using which method we can get client &server information from the request?

55.How we can add response headers to the ServletResponse?

56.How we can set ContentType to the response?

57.What is the MIME type?

58.Is it possible to send multiple content type as response from the same servlet?

59.Write Servlet code to send movie file as response?

60.Is it possible to get PrintWriter & ServletOutputStream objects simultaneously?

61.How we can implement Redirection mechanism?

62.Explain, difference between sendRedirect&forword?

63.How many times we can call sendRedirect() method with in the same Servlet?

64.How we can add cookies to the response?

65.Explain the directory structure of a web application?

66.In which location we have to place static content?

67.Is it possible to access resources present in the context root directly?

68.Is WEB-INF folder mandatory for web application?

DURGASOFT

69.Explain about web.xml?

70.Where we have to place 3rd party jar files?

71.If the required class file available in classes folder and lib folder jar file, then which one will get preference?

72.Is there any alternate location to place servlet .class file Other than classes folder?

73.Is it possible to access web.xml directory?

74.Where we have to place tag libraries inside web application?

75.Is it important the order of tags in the web.xml?

76.Can you specify any 10 tags of web.xml?

77.With in the <web-app> which tags are mandatory?

78.What is the purpose of <servlet-name>

79.How many names are possible for a servlet in web-app?

80.Is it possible to configure jsp's in web.xml?

81.When we have to configure jsp's in web.xml?

82.What is the purpose of Servlet initialization parameters and explain how to configure in web.xml?

83.With in the servlet how we can access logical name of the servlet?

84.With in the servlet how we can access Servlet initialization parameter?

85.What is the ServletConfig object and explain the methods available in ServletConfig interface?

86.What is the purpose of <load-on-startup>explain its advantages & disadvantages?

87.If two Servlets having same<load-on-startup> values what will be happen?

88.How many types of url patterns are possible according to Servlet specification?

89.With in the <servlet-mapping>how many url pattrern tags we can take?

90.What is the difference between url, uri & urn?

91.How we can get Contextpath and quereyString directly with in the Servlet?

92.What is the necessity of welcome-file and explain How to configure welcome files in web.xml?

93.What is the default welcome-file?

DURGASOFT

94. Is it possible to configure welcome file in folder wise?

95.What is the necessity of error page and explain the process of Configuration in web.xml?

96.How to send ErrorCode programmatically?

97.What is the purpose of <mime-mapping>?

98.Explain about of war file & the process of creation?

99.What is the purpose of META-INF folder?

100.Explain about MANIFEST.MF?

101.Explain about <context-param>tag &<init-param>?

102.What are the differences between Servlet initialization parameters& Context initialization parameters?

103.How we can access context parameters and servlet parameters with in the Servlet?

104.How we can declare context & Servlet <init-parameters>in web.xml?

105.What is the scope of <context-param>&<init-param>?

106.What are the difference between parameters & attributes?

107.What is the purpose of an attribute?

108.What are various scopes available for Servlets?

109.Explain the cases where we should go for each scope?

110.explain the methods to perform following activities?

a) Adding an attribute?

b) Get the name of an attribute?

c) Remove an attribute?

d) Modify the value of an attribute?

e) To display all attribute names present in a specified scope?

111.If we store information in application scope by default it is available everywhere with in the web application, then what is the need of session and Request scopes?

112.What is the purpose of RequestDispatcher?

DURGASOFT

113.How many possible ways we can get the RequestDispatcher?

114.What is the difference between obtaining RequestDispatcher from ServletRequest and ServletContext object?

115.What is the difference between forward() &include()?

116.What is the difference between forward() &sendRedirect()?

117.What is the foreign RequestDispatcher & explain the process how we can get it?

118.What are various attributes added by web container while forwarding & including? What is the purpose of these attributes?

119.What is the purpose of filter? Explain the cases where exactly required?

120.By using which interfaces we can implement filter concepts?

121. What is the purpose of FilterChain?

122.How we can configure filter in web.xml?

123.In how many ways we can map a filter?

124.In filter chain in which order the filters will be executed?

125.What is the difference between Filter interface doFilter() & FilterChain doFilter()?

126.What is the purpose of wrapper?

127.Explain the types of wrappers?

128.Explain the cases where you used filters & wrappers in your previous project?

129.What is the need of Session Management? Explain cases where Session Management is required with example?

130.What are the various Session Management techniques are available?

131.Explain the process of creating Session object?

132.What is the difference between getSession() & getSession(boolean b)?

133.Explain with an example where getSession(false) is required?

134.How we can invalidate a Session?

135.Define Session management?

136.How we can configure SessionTimeout in web.xml?

DURGASOFT

137.What is the difference between <Session-timeout>and <SetMaxInactiveInterval>?

138.How to know SessionCreation time & lastaccesed time?

139.Explain the Session Management Mechanism by SessionAPI?

140.What is the default session timeout?

141.Explain Session Management by using Cookies?

142.How the SessionId is exchanging between Client & Server?

143.What is the difference between Session API & Cookie which approach is recommended?

144.What is the difference between persistent and non persistent cookies?

145.What is URL Rewriting?

146.By using which Methods we can implement URLRewriting?

147.BY using which methods we can identify under laying Session Management technique?

148.Explain advantages & disadvantages of

cookies

URL Writing

Session API

149.Explain the purpose of Listener?

150.What are the various listeners are available according to Servlet specification?

151.What is the difference between ServletRequestListener and ServletRequestAttributeListener?

152.How to configure listener in web.xml?

153.To print hit count of the web application which listener is required to use?

154.To print the no of active session objects at server side which listener is responsible?

155.What is the difference between HSAL & HSBL?

156.What are various methods present in binding listeners?

157.Explain about HttpSessionActivationListener?

158.At the time of application deployment to perform certain activities which listener is responsible?

DURGASOFT

159.What are various listeners which are not required to configure in web.xml?

160.What are various event classes available in ServletSpecification?

161.Is it possible to configure more than one listener of same type?

162.If you are configures more than one listener of the same type then in which order listener will be executed?

163.Who is responsible to perform instantiation of listener?

164.At what time listener classes will be instantiated?

165.What is the difference between declarative Security & programmatic security?

166.By using which methods we can implement programmatic Security?

167.What is the purpose of <security-role-ref>?

168.How we can configure users in Tomcat?

169.In your previous project how you implement security?

170.In your previous project what type of authentication used?

DURGASOFT

JSP(JAVA SERVER PAGES)

The main purpose of the web applications in Enterprise applications area is to generate dynamic response from server machine.

To design web applications at server side we may use Web Technologies like CGI, Servlets, JSP and so on.

CGI is basically a Process based technology because it was designed on the basis of C technology.

If we deploy any CGI application at server then for every request CGI container will generate a separate process.

In the above context, if we send multiple number of requests to the same CGI application then CGI container has to generate multiple number of processes at server machine.

To handle multiple number of processes at a time server machine has to consume more number of system resources, as a result the performance of the server side application will be reduced.

To overcome the above problem we have to use Thread based technology at server side like servlets.

In web application development, servlets are very good at the time of pick up the request and process the request but servlets are not good at the time of generating dynamic response to client.

Servlet is a Thread based technology, if we deploy it at server then container will create a separate thread instead of the process for every request from the client.

Due to this Thread based technology at server side server side application performance will be increased.

In case of the servlet, we are unable to separate both presentation logic and business logic.

If we perform any modifications on servlets then we must perform recompilation and reloading.

If we want to design web applications by using servlets then we must require very good knowledge on Java technology.

JSP is a server side technology provided by Sun Microsystems to design web applications in order to generate dynamic response.

The main intention to introduce Jsp technology is to reduce java code as much as possible in web applications.

Jsp technology is a server side technology, it was designed on the basis of Servlet API and Java API.

In web application development, we will utilize Jsp technology to prepare view part or presentation part.

DURGASOFT

Jsp technology is very good at the time of generating dynamic response to client with very good look and feel.

If we want to design any web application with Jsp technology then it is not required to have java knowledge.

In case of Jsp technology, we are able to separate presentation logic and business logic because to prepare presentation logic we will use html tags and to prepare business logic we will use Jsp tags separately.

If we perform any modifications on Jsp pages then it is not required to perform recompilation and reloading because Jsp pages are auto-compiled and auto-loaded.

Jsp Deployment:

In web application development, it is possible to deploy the Jsp pages at any location of the web application directory structure, but it is suggestible to deploy the Jsp pages under application folder.

If we deploy the Jsp pages under application folder i.e. public area thenwe are able to access that Jsp page from client by using its name directly in the url.

If we deploy the Jsp pages under private area(WEB-INF, classes) then we must define url pattern for the Jsp page in web.xml file and we are able to access that Jsp page by specifying url pattern in client url.

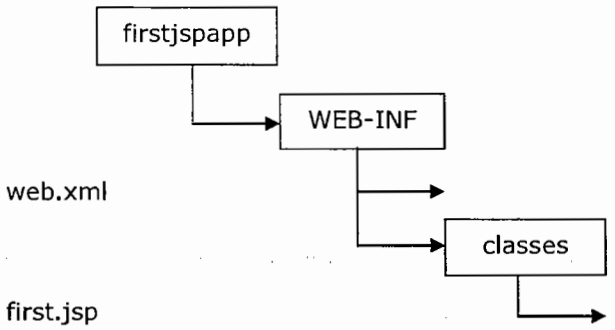
To configure Jsp pages in web.xml file we have to use the following xml tags.

```
<web-app>
        ----------
<servlet>
<servlet-name>logical_name</servlet-name>
<Jsp-file>context relative path of Jsp page</Jsp-file >
</servlet>
<servlet-mapping>
<servlet-name>logical_name</servlet-name>
<url-pattern>pattern_name</url-pattern>
</servlet-mapping>
        ----------
</web-app>
```

DURGASOFT

Application:

Directory Structure:



first.jsp:

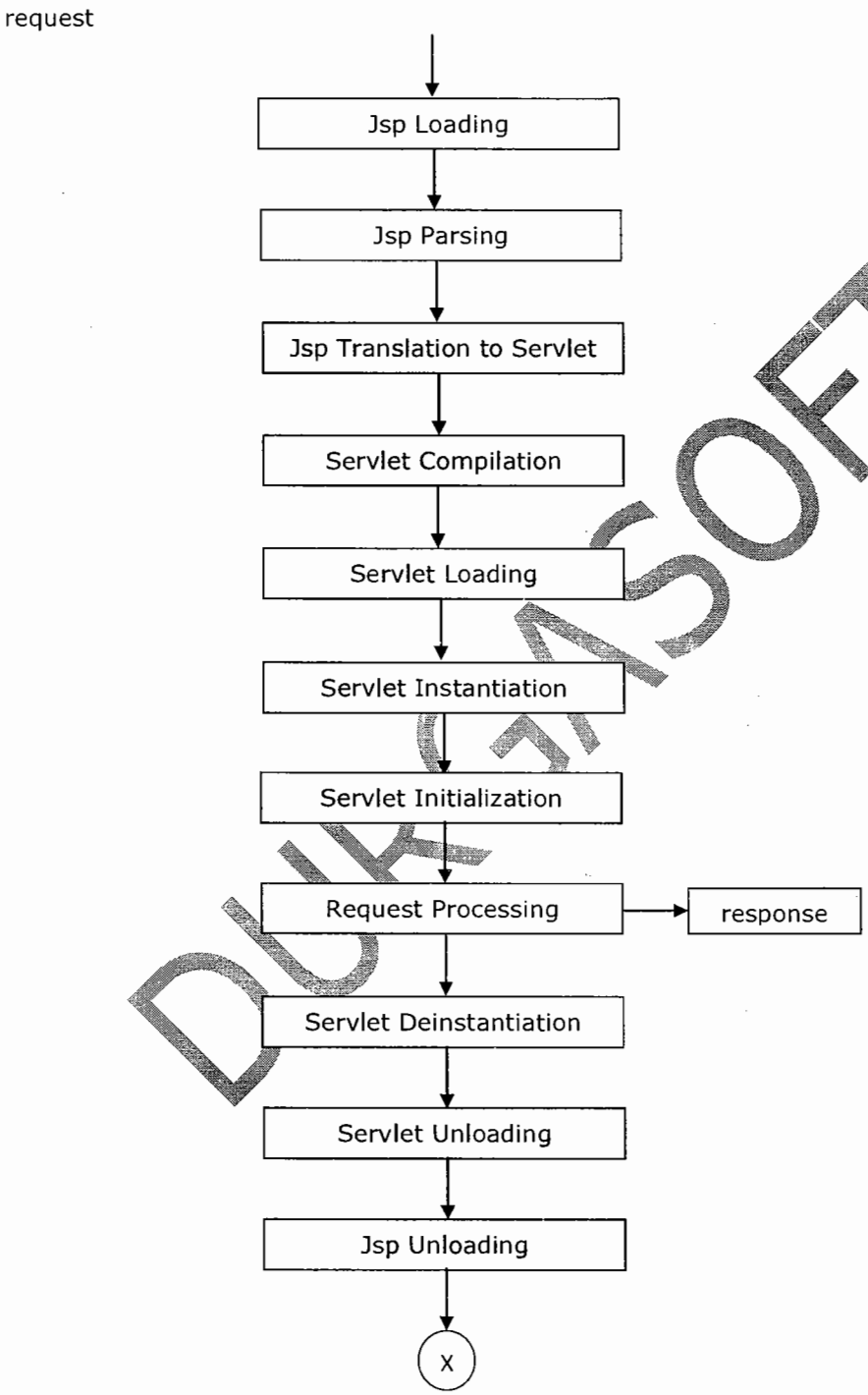
```
<html>
<body bgcolor="lightgreen">
<center><b><font size="7" color="red">
<br><br>
   First Jsp Application Deployed in Classes
</font></b></center>
</body>
</html>
```

Web.xml:

```
<web-app>
<servlet>
<servlet-name>fj</servlet-name>
<jsp-file>/WEB-INF/classes/first.jsp</jsp-file>
</servlet>
<servlet-mapping>
<servlet-name>fj</servlet-name>
<url-pattern>/jsp</url-pattern>
</servlet-mapping>
</web-app>
```

DURGASOFT

Jsp Life Cycle:



DURGASOFT

When we send request from client to server for a particular Jsp page then container will pick up the request, identify the requested Jsp pages and perform the following life cycle actions.

1. Jsp Loading:

Here container will load Jsp file to the memory from web application directory structure.

2. Jsp Parsing:

Here container will check whether all the tags available in Jsp page are in well-formed format or not.

3. Jsp Translation to Servlet:

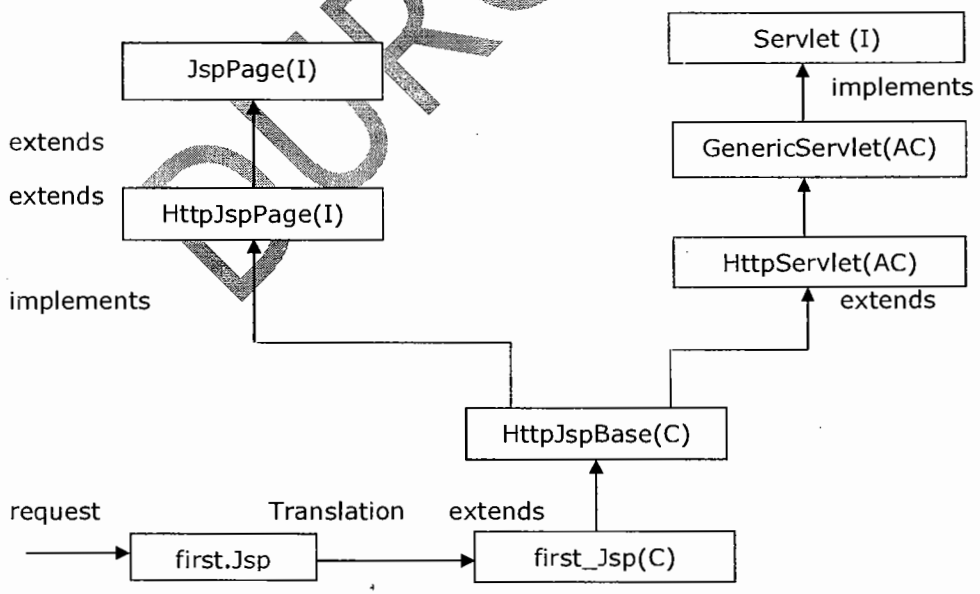
After the Jsp parsing container will translate the loaded Jsp page into a particular servlet.

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. By default all the translated servlets provided by Tomcat container are final.

The default super class for translated servlet is **HttpJspBase**.



DURGASOFT

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 req, HttpServletResponse res)

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.

4. Servlet Compilation:

After getting the translated servlet container will compile servlet.java file and generates the respective .class file.

5. Servlet Loading:

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

6. Servlet Instantiation:

Here container will create object for the loaded servlet.

7. Servlet Initialization:

Here container will access _JspInit() method to initialize the servlet.

8. 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.

9. 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.

DURGASOFT

10. 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.

11. 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.

12. 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.

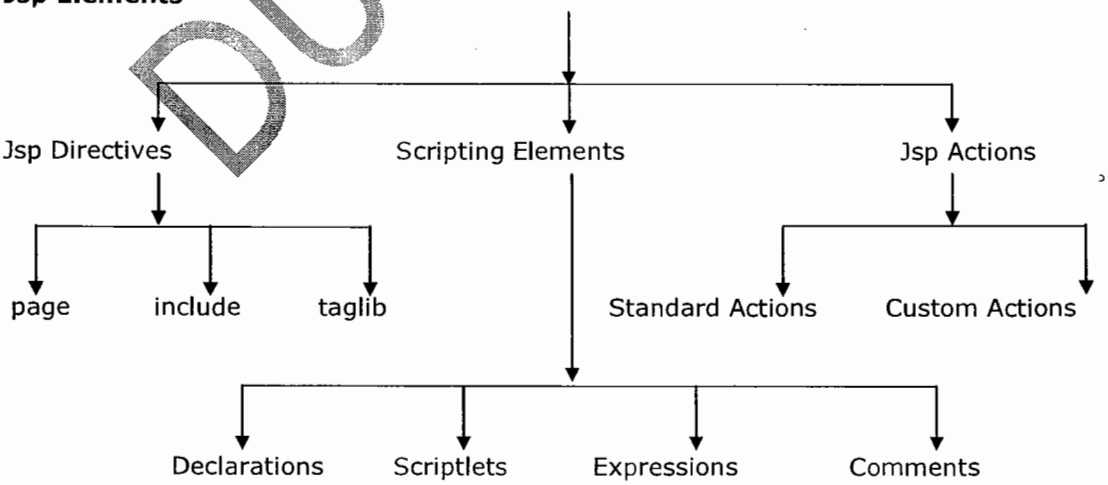
13. Servlet Unloading and Jsp Unloading:

After the servlet deinstantiation container will eliminate the translated servlet byte code and Jsp code from memory.

Jsp Elements:

In web applications to design Jsp pages we have to use the following elements.

Jsp Elements



DURGASOFT

Q: What are the differences between Jsp Directives and Scripting Elements?

Ans: 1. In web applications, Jsp Directives can be used to define present Jsp page characteristics, to include the target resource content into the present Jsp page and to make availableuser defined tag library into the present Jsp page.

In web applications, Jsp Scripting Elements can be used to provide code in Jsp pages.

2. All the Jsp Directives will be resolved at the time of translating Jsp page to servlet.

All the Jsp Scripting Elements will be resolved at the time of request processing.

3. Majority of the Jsp Directives will not give direct effect to response generation, but majority of Scripting Elements will give direct effect to response generation.

Q: To design Jsp pages we have already Jsp Scripting Elements then what is requirement to go for Jsp Actions?

Ans: In Jsp applications, Scripting Elements can be used to allow java code inside Jsp pages but the main theme of Jsp technology is not to allow java code inside the Jsp pages.

In the above context, to preserve the theme ofJsp technology we have to eliminate scripting elements from Jsp pages, for this we have to provide an alternative i.e. Jsp Actions provided by Jsp technology.

In case of Jsp Actions, we will define scripting tag in place of java code, in Jsp pages and we will provide the respective java code inside the classes folder.

In this context, when Jsp container encounter the scripting tag then container will execute the respective java code and perform a particular action called as **Jsp Action.**

DURGASOFT

1. Jsp Directives:

To provide Jsp Directives in Jsp pages we have to use the following syntaxes.

1. Jsp-Based Syntax:

<%@Directive_name [attribute-list]%>

Ex: <%@page import="java.io.*"%>

2. XML-Based Syntax:

<jsp:directive.directiveName[attribute-list]%/>

Ex: <jsp:directive.page import="java.io.*"/>

There are 3 types of Directives in Jsp technology.

1. Page Directive
2. Include Directive
3. Taglib Directive

1. Page Directive:

In Jsp technology, **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.

Syntax 1:<%@page [attribute-list]%>

Syntax 2:<jsp:directive.page [attribute-list]/>

Where attribute-list in Jsp page directive may include the following list.

1. language
2. contentType
3. import
4. extends
5. info
6. buffer
7. autoFlush
8. errorPage
9. isErrorPage
10. session
11. isThreadSafe
12. isELIgnored

1. language:

This attribute can be used to specify a particular scripting language to use scripting elements.

The default value of this attribute is java.

Ex: <%@page language="java"%>

2. 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.

DURGASOFT

Ex: <%@page contentType="text/html"%>

3. import:

This attribute can be used to import a particular package/packages into the present Jsp pages.

Ex:<%@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.

Approach 1:

Specify multiple number of packages with comma(,) as separator to a single import attribute.

Ex: <%@page import="java.io.*,java.util.*,java.sql.*"%>

Approach 2:

Provide multiple number of import attributes for the list of packages.

Ex: <%@page import="java.io.*" import="java.util.*" import="java.sql.*"%>

Note: Among all the Jsp page attributes only import attribute is repeatable attribute, no other attribute is repeatable.

The default values of this attribute are java.lang, javax.servlet, javax.servlet.http, javax.servlet.jsp.

4. extends:

This attribute will take a particular class name, it will be available to the translated servlet as super class.

Ex: <%@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.

5. info:

This attribute can be used to specify some metadata about the present Jsp page.

DURGASOFT

Ex: <%@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.

6. 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.

Ex: <%@page buffer="52kb"%>

The default value of this attribute is 8kb.

7. 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.

Ex:
```
<%@page buffer="52kb" autoFlush="true"%>
<html>
<body bgcolor="lightgreen">
<center><b><font size="7" color="red"><br><br>
<%
    for(int i=0; i<10000000; i++) {
        out.println("RAMA");
    }
    %>
</font></b></center></body>
</html>
```

In the above piece of code, if we provide autoFlush attribute value false then container will raise an exception like

org.apache.jasper.JasperException:An exception occured processing JSP page/first.jsp at line:9

DURGASOFT

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 likeorg.apache.jasper.JasperException:/first.jsp(1,2) jsp.error.page.badCombo

The default value of this attribute is true.

8. errorPage:

This attribute can be used to specify an error page to execute when we have an exception in the present Jsp page.

Ex: <%@page errorPage="error.jsp"%>

9. 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.

Ex: <%@page isErrorPage="true"%>

first.jsp:

```
<%@page errorPage="error.jsp"%>
<%
java.util.Date d=null;
out.println(d.toString());
%>
```

error.jsp:

```
<%@page isErrorPage="true"%>
<html>
<body bgcolor="lightgreen">
```


DURGASOFT

```
<center><b><font size="" color=""><br><br>
<%=expression%>
</font></b></center></body>
</html>
```

10. session:

It is a boolean attribute, it is give an 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.

Ex: <%@page session="true"%>

11. 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.

Ex: <%@page isThreadSafe="true"%>

12. 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.

Ex: <%@page isELIgnored="true"%>

DURGASOFT

2. Include Directive:

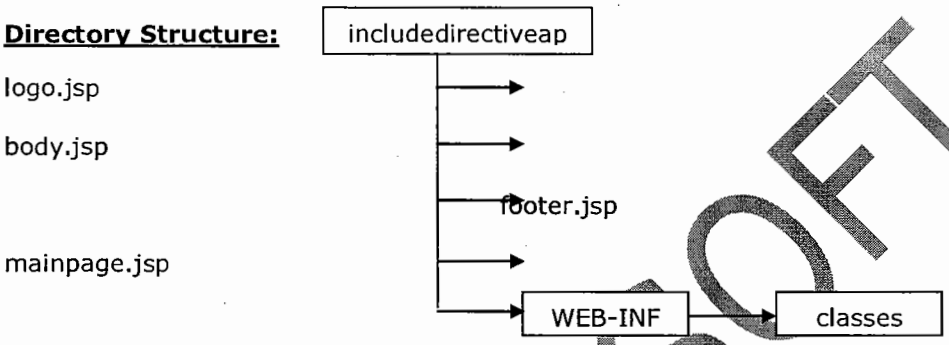
Include Directive can be used to include the content of the target resource into the present Jsp page.

Syntax:<%@include file="--"%>

Where file attribute can be used to specify the name and location of the target resource.

Application:

Directory Structure:



logo.jsp:

```
<html>
<body><center>
<table width="100%" height="20%" bgcolor="red">
<tr><td colspan="2"><center><b><font size="7" color="white">
Durga 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@durgasoftwaresolutions
</font></b></center></td></tr>
</table></center></body>
</html>
```

body.jsp:

DURGASOFT

```
<html>
<body bgcolor="lightyellow">
<center><b><font size="7">
<p><br>
   Durga 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"%>
```

3. Taglib Directive:

The main purpose of **Taglib Directive** is to make available user defined tag library into the present Jsp pages.

Syntax:<%@taglib uri="_ _" prefix="_ _"%>

Where uri attribute can be used to specify the name and location of user defined tag library.

Where prefix attribute can be used to define prefix names for the custom tags.

Ex: <%@taglib uri="/WEB-INF/db.tld" prefix="connect"%>

2. Scripting Elements:

The main purpose of **Jsp Scripting Elements** is to allow java code directly into the present Jsp pages.

There are 3 types of Scripting Elements.

1. Declarations
2. Scriptlets
3. Expressions

1. Declarations:

It can be used to provide all the java declarations like variable declarations, method definitions, classes declaration and so on.

DURGASOFT

Syntax:<%!

```
                --------
                --------   Java Declarations
              --------
          %>
```

If we provide any java declarations by using declaration scripting element then that all java declarations will be available to the translated servlet as class level declarations.

2. Scriptlets:

This Scripting Element can be used to provide a block of java code.

Syntax:<%

```
                --------
                --------   Block of Java code
              --------
          %>
```

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);

first.jsp:

```
<%@page import="java.util.*"%>
<%!
Date d=null;
String date=null;
%>
<%
d=new Date();
date=d.toString();
%>
<html>
<body bgcolor="lightyellow">
<center><b><font size="6" color="red"><br><br>
Today Date : <%=date%>
</font></b></center></body>
</html>
```

DURGASOFT

Translated Servlet:

```
-------------
-------------
import java.uitl.*;
public final class first_jsp extends HttpJspBase implements JspSourceDependent
{
   Date d=null;
   String date=null;
public void _jspInit()throws ServletException
   {  }
public void _jspDestroy()
   {  }
public void _jspService(HttpServletRequest req, HttpServletResponse res)throws SE, IOE
   {
      d=new Date();
      date=d.toString();
      out.write("<html>");
      out.write("<body bgcolor='lightyellow'>");
      out.write("<center><b><font size='6' color='red'><br><br>");
      out.write("Today Date.......");
      out.write(date);
      out.write("</font></b></center></body></html>");
}
}
```

4. Jsp Comments:

In Jsp pages, we are able to use the following 3 types of comments.

1. XML-Based Comments
2. Jsp-Based Comments
3. Java-Based Comments inside Scripting Elements

1. XML-Based Comments:

```
<!--
      --------
      --------  Description
      --------
   -->
```

2. Jsp-Based Comments:

DURGASOFT

```
<%--
      --------
      --------  Description
      --------
   --%>
```

3. Java-Based Comments inside Scripting Elements:

1. Single Line Comment:

```
//--------Description---------
```

2. Multiline Comment:

```
/*
      --------
      --------  Description
      --------
   */
```

3. Documentation Comment:

```
/**
      --------
      --------  Description
      --------
*/
```

Jsp Implicit Objects:

In J2SE applications, for every java developer it is common requirement to display data on command prompt.

To perform this operation every time we have to prepare PrintStream object with command prompt location as target location

```
PrintStream ps=new PrintStream("C:\program Files\.....\cmd.exe") ;

ps.println("Hello");
```

In java applications, PrintStream object is frequent requirement so that java technology has provided that PrintStream object as predefined object in the form of out variable in System class.

```
public static final PrintStream out;
```

Similarly in web applications, all the web developers may require some objects like request, response, config, context, session and so on are frequent requirement, to get these objects we have to write some piece of java code.

DURGASOFT

Once the above specified objects are as frequent requirement in web applications, Jsp technology has provided them as predefined support in the form of Jsp Implicit Objects in order to reduce burden on the developers.'

Jsp technology has provided the following list of implicit objects with their respective types.

1. out ------→ javax.servlet.jsp.JspWriter
2. request ------→ javax.servlet.HttpServletRequest
3. response ------→ javax.servlet.HttpServletResponse
4. config ------→ javax.servlet.ServletConfig
5. application ------→ javax.servlet.ServletContext
6. session ------→ javax.servlet.http.HttpSession
7. exception ------→ java.lang.Throwable
8. page ------→ java.lang.Object
9. pageContext ------→ javax.servlet.jsp.PageContext

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: JspWriterout=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

DURGASOFT

```
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;
```

Ex:
```
<%
    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.

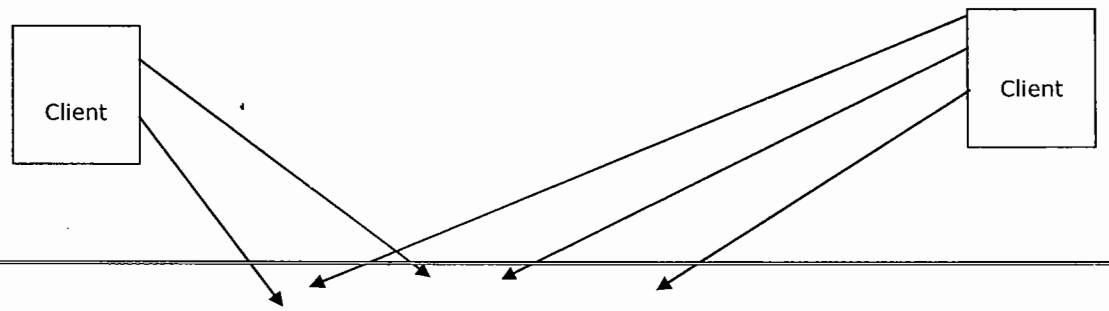
Public Object findAttribute(String name)

Ex: String name=pageContext.findAttribute("uname");

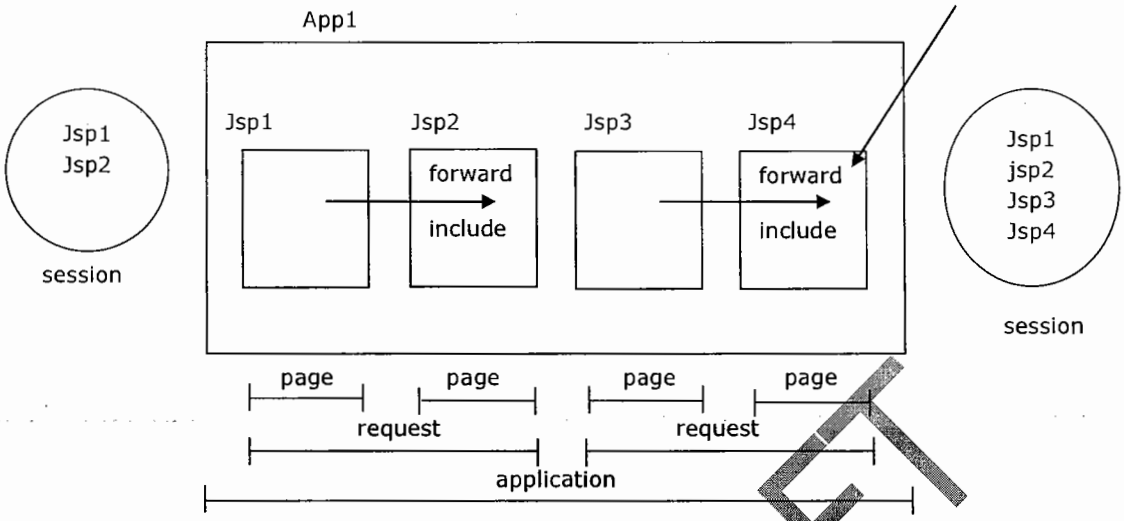
Jsp Scopes:

In J2SE applications, to define data availability i.e. scope for we have to use access specifiers public, protected, default and private.

Similarly to make available data to number of resources Jsp technology has provided the following 4 types of scopes with the access modifiers.



DURGASOFT



1. Page Scope:

If we declare the data in page scope by using pageContext object then that data should have the scope upto the present Jsp page.

2. Request Scope:

If we declare the data in request object then that data should have the scope up to the number of resources which are visited by the present request object.

3. Session Scope:

If we declare the data in HttpSession object then that data should have the scope up to the number of resources which are visited by the present client.

4. Application Scope:

If we declare the data in ServletContext object then that data should have the scope up to the number of resources which are available in the present web application.

----------Application1----------

<u>App1:</u>

<u>employeedetails.html:</u>

DURGASOFT

```
<html>
        <body bgcolor="lightblue"><br><br><br><br>
        <center><h1>Employee Details Form</h1></center>
        <pre><h2>
<form method="get" action="display.jsp">
                Employee Id :<input type="text" name="eid"/>
                Employee Name : <input type="text" name="ename"/>
                Employee Salary : <input type="text" name="esal"/>
                <input type="submit" value="Display"/>
        </h2></pre>
        </body>
</html>
```

display.jsp:

```
<%!
        int eid;
        String ename;
        float esal;
%>
<%
        try {
                eid=Integer.parseInt(request.getParameter("eid"));
                String ename=request.getParameter("ename");
                float esal=Float.parseFloat(request.getParameter("esal"));
        }
        catch(Exception e){
                e.printStackTrace();
        }
 %>
<html>
        <body>
                <center><h1>Employee Details</h1></center>
                <center>
                        Employee Id : <%=eid %><br><br>
                        Employee Name : <%=ename %><br><br>
                        Employee Salary : <%=esal %><br><br>
                </center>
        </body>
</html>
```

DURGASOFT

3. 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.

DURGASOFT

```
Ex:  public class Employee implements Serializable {
        private String eno;
        private String ename;
        private float esal;
public void setEno(String eno) {
            this.eno=eno;
        }
        public void setEname(String ename) {
this.ename=ename;
        }
public void setEsal(String esal) {
this.esal=esal;
        }
public String getEno() {
            return eno;
        }
public String getEname() {
            return ename;
        }
public float getEsal() {
            return esal;
        }
      }
```

1. <jsp:useBean>:

The main purpose of <jsp:useBean> tag is to interact with bean object from a particular Jsp page.

Syntax:<jsp:useBean id="--" class="--" type="--" scope="--"/>

Where id attribute will take a variable to manage generated Bean object reference.

Where class attribute will take the fully qualified name of Bean class.

Where type attribute will take the fully qualified name of Bean class to define the type of variable in order to manage Bean object reference.

Where scope attribute will take either of the Jsp scopes to Bean object.

Note: In <jsp:useBean> tag, always it is suggestible to provide either application or session scope to the scope attribute value.

Ex: <jsp:useBean id="e" class="Employee" type="Employee" scope="session"/>

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.

DURGASOFT

After getting Bean object reference container will store Bean object in a scope specified as value to scope attribute.

2. <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.

----------Application2----------

DURGASOFT

usebeanapp:

empform.html:

```
<html>
	<body bgcolor="lightblue"><br><br><br><br>
	<center><h1>Employee Details Form</h1></center>
	<form method="get" action="display.jsp">
	<pre><h2>
		Employee Id :<input type="text" name="eid"/>
		Employee Name : <input type="text" name="ename"/>
		Employee Salary : <input type="text" name="esal"/>
		<input type="submit" value="Display"/>
	</h2></pre>
	</form>
	</body>
</html>
```

Employee.java:

```
package comm.dss;
publicclass Employee implements java.io.Serializable {
	privateint eid;
	private String ename;
	privatefloat esal;
	publicint getEid() {
		return eid;
	}
	publicvoid setEid(int eid) {
		this.eid = eid;
	}
	public String getEname() {
		return ename;
	}
	publicvoid setEname(String ename) {
		this.ename = ename;
	}
	publicfloat getEsal() {
		return esal;
	}
	publicvoid setEsal(float esal) {
		this.esal = esal;
	}
}
```

display.jsp:

```
<%!
	int eid;
	String ename;
	float esal;
%>
```

DURGASOFT

```
<%
        try {
                eid=Integer.parseInt(request.getParameter("eid"));
                ename=request.getParameter("ename");
                esal=Float.parseFloat(request.getParameter("esal"));
        }
        catch(Exception e){
                e.printStackTrace();
        }
 %>
<jsp:useBean id="e" class="com.dss.EmployeeBean" type="com.dss.EmployeeBean"
scope="session">
<jsp:setProperty name="e" property="eid" value='<%=eid %>'/>
<jsp:setProperty name="e" property="ename" value='<%=ename %>'/>
<jsp:setProperty name="e" property="esal" value='<%=esal %>'/>
<html>
        <body>
                <center><h1>Employee Details</h1></center>
                <center>
                        Employee Id : <jsp:getProperty name="e"
property="eid"/><br><br>
                        Employee Name : <jsp:getProperty name="e"
property="ename"/><br><br>
                        Employee Salary : <jsp:getProperty name="e"
property="esal"/><br><br>
                </center>
        </body>
</html>
</jsp:useBean>
```

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.

4. <jsp:include>:

Q: What are the differences between include directive and <jsp:include> action tag?

DURGASOFT

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.

----------Application3----------

includeapp:

addform.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"/>
```


DURGASOFT

```
                </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",
"durga");
                        st=con.createStatement();
                }
                catch(Exception e){
                        e.printStackTrace();
                }
        }
%>
<%
        try{
                pid=request.getParameter("pid");
                pname=request.getParameter("pname");
                pcost=Integer.parseInt(request.getParameter("pcost"));
                st.executeUpdate("insert into product
values('"+pid+"','"+pname+"','"+pcost+")");
                rs=st.executeQuery("select * from product");
                rsmd=rs.getMetaData();
                int count=rsmd.getColumnCount();
%>
        <html><body><center>
        <table border="1" bgcolor="lightyellow">
        <tr>
<%
        for (int i=1;i<=count;i++){
%>
        <td><b><font size="6" color="red">
        <center><%=rsmd.getColumnName(i) %></center>
        </font></b></td>
<%
        }
%>
        </tr>
<%
```

DURGASOFT

```
        while(rs.next()){
%>
        <tr>
<%
        for(int i=1;i<=count;i++){
%>
        <td><b><font size="6">
<%=rs.getString(i) %>
        </font></b></td>
<%
        }
%>
        </tr>
<%
        }
%>
        </table></center></body></html>
<%
        }
        catch(Exception e){
                e.printStackTrace();
        }
%>
        <hr>
        <jsp:include page="addform.html" flush="true"/>
```

DURGASOFT

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.

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>
```


DURGASOFT

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",
"durga");
			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
			{
```


DURGASOFT

```
%>
        <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>
```


DURGASOFT

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");

----------Application5----------

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="durga 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>

DURGASOFT

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>

Ex:

```
<%@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>
```

DURGASOFT

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.

DURGASOFT

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>
```


DURGASOFT

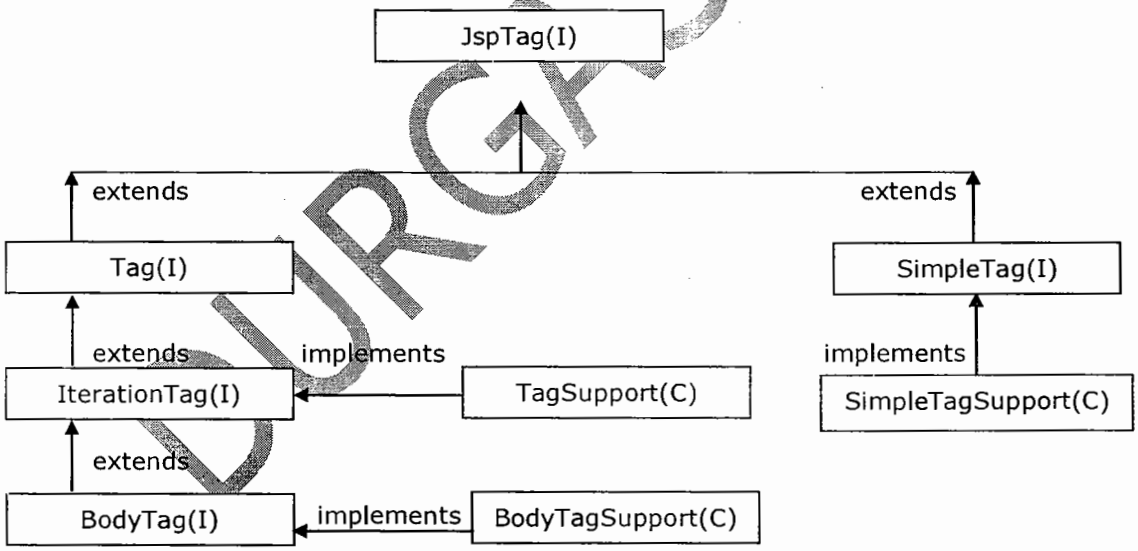
```
<tag>
<name>custom tag name</name>
<tag-class>fully qualified name of TagHandler class</tag-class>
<body-content>jsp or empty</body-content>
<short-name>custom tag short name</short-name>
<description>description about custom tags</description>
</tag>
        ---------
</taglib>
```

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

2. 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.

```
public interface Tag extends JspTag
{
    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 void setPageContext(PageContext pageContext);
    public void setParent(Tag t);
    public Tag getParent();
    public int doStartTag()throws JspException;
    public int doEndTag()throws JspException;
    public void release();
}
public class MyHandler implements Tag
{
    --------
    --------
}
```

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.

DURGASOFT

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.

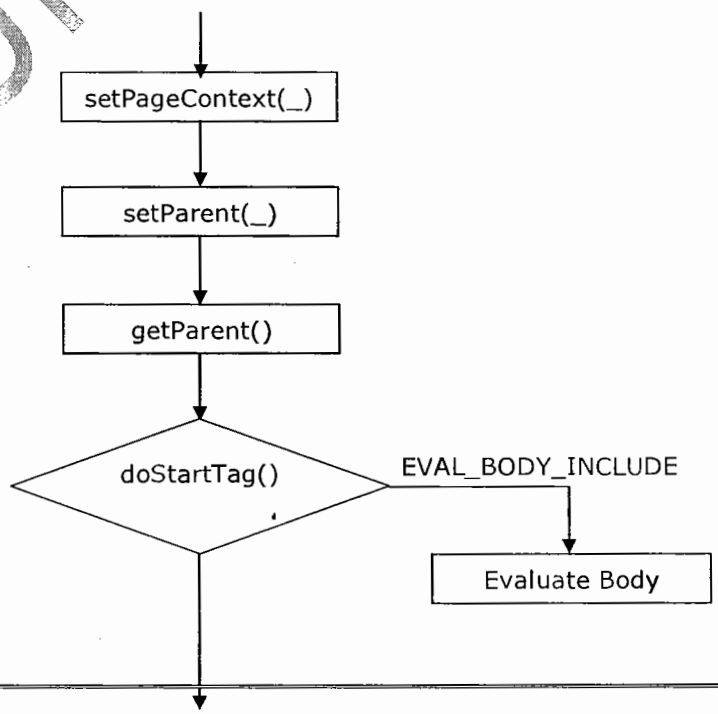
1. EVAL_PAGE
2. SKIP_PAGE

If doEndTag() method returns EVAL_PAGE constant then container will evaluate the remaining Jsp page.

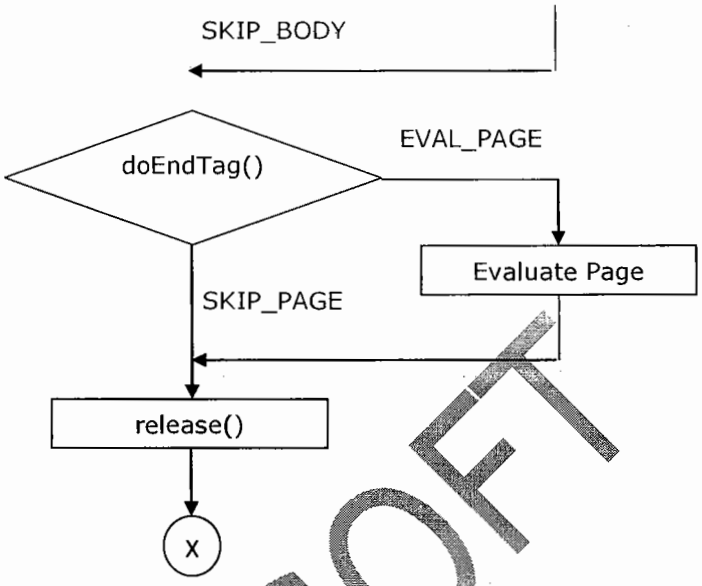
If doEndTag() method returns SKIP_PAGE constant then container will not evaluate the remaining Jsp page.

Where release() method can be used to perform TagHandler class deinstantiation.

Life Cycle of Tag interface:



DURGASOFT



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.

----------Application6----------

custapp1:

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.dss.HelloHandler</tag-class>
                <body-content>jsp</body-content>
        </tag>
</taglib>
```

HelloHandler.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class HelloHandler implements Tag
{
	PageContext pageContext;
	public void setPageContext(PageContext pageContext)
	{
		this.pageContext=pageContext;
		System.out.println("setPageContext()");
	}
	public void setParent(Tag t)
	{
		System.out.println("setParent()");
	}
	public Tag getParent()
	{
		System.out.println("getParent()");
		return null;
	}
	public int doStartTag() throws JspException
	{
		try
		{
			System.out.println("doStartTag()");
			JspWriter out=pageContext.getOut();
			out.println("<h1>Hello..... First Custom Tag Application</h1>");
		}
		catch (Exception e)
		{
			e.printStackTrace();
		}
		return SKIP_BODY;
	}
	public int doEndTag()throws JspException
	{
		System.out.println("doEndTag()");
		return SKIP_PAGE;
	}
	public void release()
	{}
}
```

Note: To compile above code we need to set the classpath environment variable to the location of jsp-api.jar file.

Observations:

DURGASOFT

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.

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:

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.

Ex:<mytags:hello name="Durga"/>

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;
    }
```


DURGASOFT

```
    ----------
  }
```

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.

```
public interface IterationTag extends Tag {
    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 void setPageContext(PageContext pageContext);

    public void setParent(Tag t);

    public Tag getParent();

    public int doStartTag()throws JspException;

    public int doAfterBody()throws JspException;

    public int doEndTag()throws JspException;

    public void release();
}


public class MyHandler implements IterationTag
{ ----- }
```

In general there are 2 possible return values from doStartTag() method.

1. EVAL_BODY_INCLUDE
2. SKIP_BODY

If we return SKIP_BODY constant from doStartTag() method then container will skip the custom tag body.

If we return EVAL_BODY_INCLUDE constant from doStartTag() method then container will execute the custom tag body.

DURGASOFT

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.

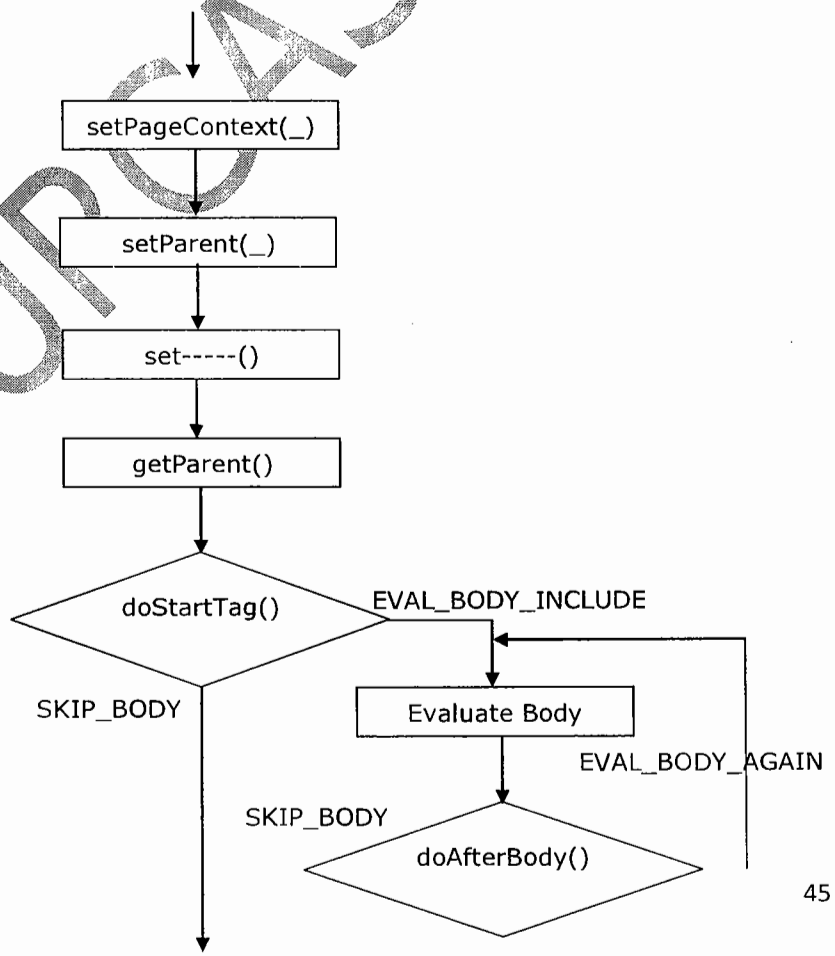
In the above context, evaluating the custom tag body again or not is completely depending on the return value which we are going to return from doAfterBody() method.

1. EVAL_BODY_AGAIN
2. SKIP_BODY

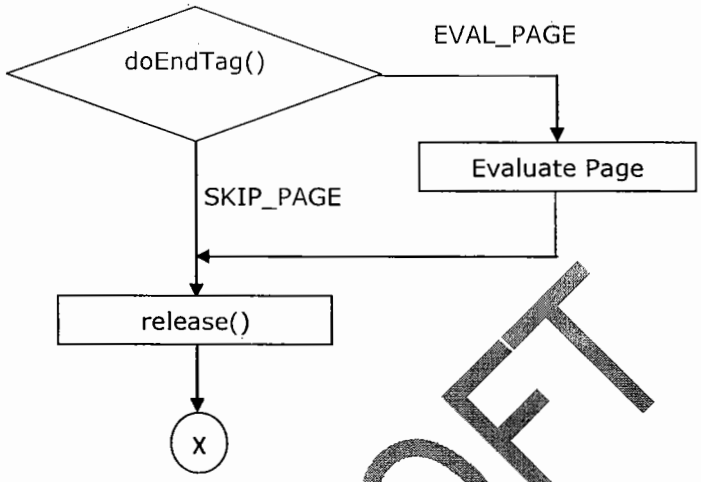
If we return EVAL_BODY_AGAIN constant from doAfterBody() method then container will execute the custom tag body again.

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:



DURGASOFT



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.

```
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;
    }
```


DURGASOFT

```
    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() { }
}
  public class MyHandler implements TagSupport
{ ----- }
```

----------Application7----------

custapp2:

iterate.jsp:

```
<%@taglib uri="/WEB-INF/iterate.tld" prefix="mytags"%>
<mytags:iterate times="10"><br>
        Durga Software Solutions
</mytags:iterate>
```

iterate.tld:

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>iterate</name>
                <tag-class>com.dss.Iterate</tag-class>
                <body-content>jsp</body-content>
                <attribute>
                        <name>times</name>
                        <required>true</required>
                        <rtexprvalue>true</rtexprvalue>
                </attribute>
        </tag>
</taglib>
```

Iteration.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class Iterate extends TagSupport
{
        int count=1;
        private int times;
        public void setTimes(int times)
        {
```

DURGASOFT

```
                this.times=times;
        }
        public int doStartTag() throws JspException
        {
                return EVAL_BODY_INCLUDE;
        }
        public int doAfterBody() throws JspException
        {
                if(count<times)
                {
                        count++;
                        return EVAL_BODY_AGAIN;
                }
                else
                        return SKIP_BODY;
        }
}
```

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.

----------Application8----------

custapp3:

nested.jsp:

```
<%@taglib uri="/WEB-INF/nested.tld" prefix="mytags"%>
<h1><center>
<mytags:if condition='<%=10>20%>'>
        <mytags:true>condition is true</mytags:true>
        <mytags:false>condition is false</mytags:false>
</mytags:if>
</center></h1>
```

nested.tld:

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
```


DURGASOFT

```
        <tag>
                <name>if</name>
                <tag-class>com.dss.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.dss.True</tag-class>
                <body-content>jsp</body-content>
        </tag>
        <tag>
                <name>false</name>
                <tag-class>com.dss.False</tag-class>
                <body-content>jsp</body-content>
        </tag>
</taglib>
```

If.java:

```
package com.dss;

import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class If extends TagSupport
{
        private boolean condition;
        public void setCondition(boolean condition)
        {
                this.condition=condition;
        }
        public boolean getCondition()
        {
                return condition;
        }
        public int doStartTag() throws JspException
        {
                return EVAL_BODY_INCLUDE;
        }
}
```

True.java:

```
package com.dss;
```

DURGASOFT

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class True extends TagSupport
{
	public int doStartTag() throws JspException
	{
		If i=(If)getParent();
		boolean condition=i.getCondition();
		if(condition == true)
			return EVAL_BODY_INCLUDE;
		else
			return SKIP_BODY;
	}
}
```

False.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class False extends TagSupport
{
	public int doStartTag() throws JspException
	{
		If i=(If)getParent();
		boolean condition=i.getCondition();
		if(condition == true)
			return SKIP_BODY;
		else
			return EVAL_BODY_INCLUDE;
	}
}
```

----------Application9----------

custapp4:

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>
```


DURGASOFT

```
            <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","
durga");
                        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++)
                                {
```


DURGASOFT

```
                                        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.

```
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();
}
public class MyHandler implements BodyTag
{ ----- }
```

In case of body tags, there are 3 possible return values from doStartTag() method.

1. EVAL_BODY INCLUDE

DURGASOFT

2. SKIP_BODY
3. EVAL_BODY_BUFFERED

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.

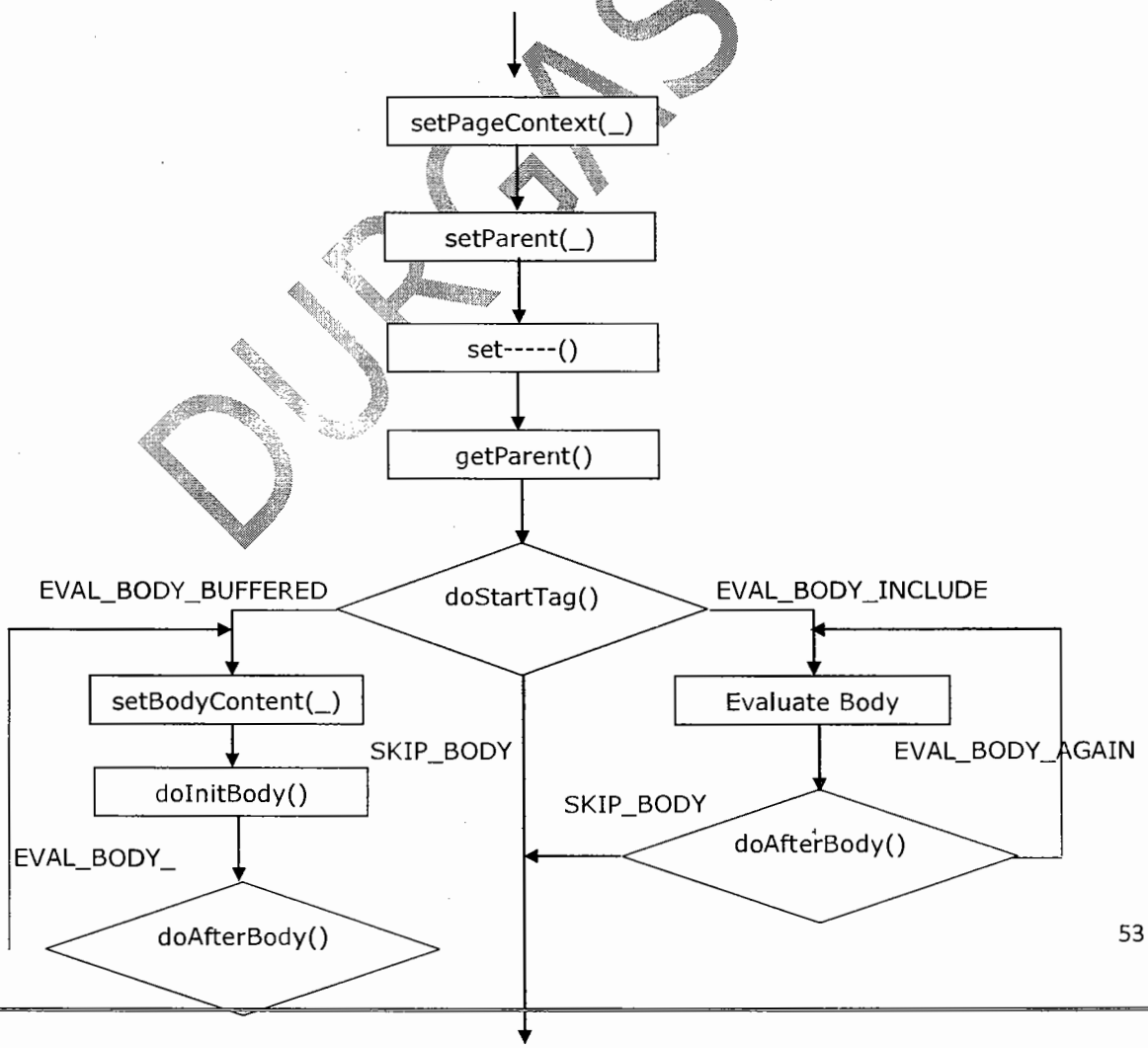
If we return SKIP_BODY constant from doStartTag() method then container will skip the custom tag body.

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.

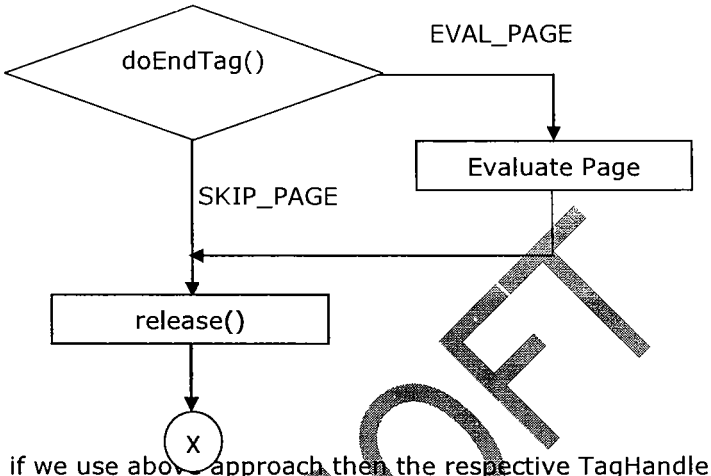
After executing setBodyContent(_) method container will access doInitBody() method in order to prepare BodyContent object for allow modifications.

Life Cycle of BodyTag interface:



DURGASOFT

INCLUDE SKIP_BODY



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 classBodyTagSupport 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() {
```


DURGASOFT

```
        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()
    {  }
}

  public class MyHandler extendsBodyTagSupport
{ ----- }
```

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()
```

----------Application10----------

custapp5:

reverse.jsp:

```
<%@taglib uri="/WEB-INF/reverse.tld" prefix="mytags"%>
<mytags:reverse>
        Durga Software Solutions
</mytags:reverse>
```

reverse.tld:

```
<taglib>
        <jsp-version>2.1</jsp-version>
        <tlib-version>1.0</tlib-version>
        <tag>
                <name>reverse</name>
                <tag-class>com.dss.Reverse</tag-class>
                <body-content>jsp</body-content>
```


DURGASOFT

```
        </tag>
</taglib>
```

Reverse.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
public class Reverse extends BodyTagSupport
{
        public int doEndTag() throws JspException
        {
                try
                {
                        String data=bodyContent.getString();
                        StringBuffer sb=new StringBuffer(data);
                        sb.reverse();
                        JspWriter out=bodyContent.getEnclosingWriter();
                        out.println("<html>");
                        out.println("<body bgcolor='lightyellow'>");
                        out.println("<center><b><font size='7' color='red'>");
                        out.println("<br><br>");
                        out.println(sb);
                        out.println("</font></b></center></body><html>");
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
                return EVAL_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>
```


DURGASOFT

```
</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","
durga");
                        st=con.createStatement();
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                }
        }
```

DURGASOFT

```
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;
}
```


DURGASOFT

```
}
```

2. Simple Tags:

Q: What are the differences between Classic tags and Simple tags?

Ans: 1. Classic tags are more API independent, but Simple tags are less API independent.

2. 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.

3. 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.

4. 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.

```
public interface SimpleTag extends JspTag {
    public void setJspContext(JspContext jspContext);
    public void setParent(JspTag t);
    public JspTag getParent();
    public void setJspBody(JspFragment jspFragment);
    public void doTag()throws JspException, IOException;
}
public class MyHandler implements SimpleTag
{ ------- }
```

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.

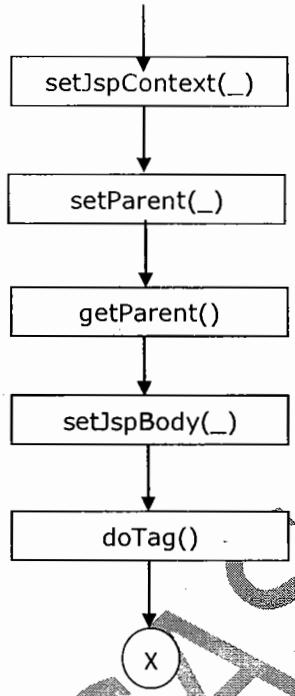
Where getParent() method can be used to get parent tags TagHandler class object.

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.

DURGASOFT

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.

```
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;
```


DURGASOFT

```
    }
    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
    {  }
}
public class MyHandler extends SimpleTagSupport
{ ------- }
```

----------Application12----------

custapp7:

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.dss.Hello</tag-class>
                <body-content>empty</body-content>
        </tag>
</taglib>
```

Hello.java:

```
package com.dss;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import java.io.*;
public class Hello extends SimpleTagSupport
{
        public void doTag() throws JspException,IOException
        {
                getJspContext().getOut().println("<h1><center>Hello SimpleTag
Application</center></h1>");
        }
}
```

DURGASOFT

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

DURGASOFT

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----->

DURGASOFT

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:

DURGASOFT

```
<%@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:

DURGASOFT

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>
```

DURGASOFT

```
        <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.

DURGASOFT

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>
```


DURGASOFT

```
            <center><b><font size="7">
                <c:forTokens var="token" items="Durga 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----------

DURGASOFT

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>
```

DURGASOFT

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----------

DURGASOFT

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="durga"/>
			<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="durga"/>
			<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"%>
```


DURGASOFT

```
<%@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="durga"/>
                        <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="durga"/>
                        <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"%>
```


DURGASOFT

```
<%@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="durga"/>
			<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.

----------Application28----------

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="durga"/>
			<sql:query var="result" sql="select * from emp"/>
			<table border="1" bgcolor="lightyellow">
			<tr>
```


DURGASOFT

```
                        <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="--"/>

DURGASOFT

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>
```


DURGASOFT

```
                    <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="Durga Software Solutions"/>

${fn.length(a)}

${fn.concat(a, "Hyderabad")}

${fn.toLowerCase(a)}

${fn.toUpperCase (a)}

${fn.contains(a, "Software")}

${fn.startsWith(a, "Durga")}

${fn.endsWith(a, "Solutions")}

${fn.substring(a, 4, 20)}

DURGASOFT

Expression Language:

In JSP technology,scripting elements will allow java code inside the JSP pages,which is against to JSP principles.

To eliminate java code and scripting elements from JSP pages,we have to use standard actions,but which are in limited number with bounded functionality.

We will use custom actions,but it should require lot of java code internally inorder to implement simple programming constructs like if,switch,for and so on.

To overcome the above problems,we are able to use JSTL tag library,but still it is not sufficient to eliminate java code completely from JSP pages.

In the above context,to eliminate java code completely from JSP pages,we have to use Expression language syntax along with standard actions,custom actions and JSTL tag library.

Eg:

To get a particular request parameter from request object,we have to use java code before expression language.

```
<%
uname=request.getParameter("uname");
%>
```

In the above context,we are able to retrieve a particular request parameter without using java code, by using expression language syntax.

Ex:

```
${param.uname}
```

To eliminate java code completely from JSP pages,by using the following expression language elements.

1)EL operators

2)EL implicit objects.

3)EL functions.

DURGASOFT

1)EL operators:

To prepare expressions in expression language,we have to use the following operators.

-->**General purpose operators:**

.,(),{},[]

-->**Arithmetic Operators:**

+ - * / % and so on

-->**Comparison Operators:**

== or eq

!= or ne

< or lt

> or gt

<= or le

>= or ge

-->**Ternary Operators:**

expr1?expr2:expr3;

-->**Logical Operators:**

&& ||

DURGASOFT

2)EL implicit objects:

-->**Scope related:**

PageScope

requestScope

sessionScope

applicationScope

param

paramValues

initParam --->to get context parameters

cookie

header --->to get particular header value

headerValues

pageContext

In JSP technology,we will use JSP implicit objects in scripting elements to reduce java codeinside the scripting elements.

Similarly,Expression Language has provided the following list of implicit objects to eliminate Java code from JSP pages.

pageScope,requestScope,sessionScope,applicationScope,param,paramValues, initParam,cookie,header,headerValues,pageContext.

To access the attributes data from the page scope,request scope,session scope and application scope,we will use the above scope related implicit objects like pageScope,requestScope,sessionScope and applicationScope implicit objects respectively.

Ex:

```
<%
request.setAttribute("uname","Laddu");
%>
<html>
<body>
${requestScope.uname}
</body>
</html>
```

DURGASOFT

param:

This can be used to access a particular request parameter.

paramValues:

This implicit object can be used to access more than one parameter values associated with a particular request parameter.

Ex:

form.html:

```
<html>
<body>
<center><b><font size="6">
<form method="get" action="first.jsp">
<br><br>
Name <input type="text" name="uname"/><br><br>
Food items <select size="3" multiple="true" name="food">
<option value="Dosa">Dosa</option>
<option value=="Vada">Vada</option>
<option value="Idly">Idly</option>
</select><input type="submit" value="Display"/>
</form></font></b></center>
</body>
</html>
```

DURGASOFT

first.jsp:

```
<html>
<body>
<center><b><font size="7"><br><br>
User Name.....${param.uname}
<br><br>
Food Items...<br>
${paramValues.food[0]}<br>
${paramValues.food[1]}<br>
${paramValues.food[2]}<br>
</font></b></center>
</body>
</html>
```

initParam

This implicit object can be used to access context parameters from ServletContex object.

Ex:

Web.xml:

```
<web-app>
<context-param>
<param-name>a</param-name>
<param-name>AAA</param-name>
</context-param>
        <context-param>
<param-name>b/param-name>
```


DURGASOFT

```
<param-name>BBB</param-name>
</context-param>
</web-app>
```

first.jsp:

```
<html>
<body>
<center><b><font size="7">
a--------->${initParam.a}<br>
b--------->${initParam.b}
</font></b></center>
</body>
</html>
```

cookie:

This implicit object can be used to get session id cookie name and session id cookie value.

Ex:

```
<html>
<body>
${cookie.JSESSIONID.name}<br>
${cookie.JSESSIONID.value}
</body>
</html>
```

DURGASOFT

header:

This implicit object can be used to access a particular request header value

Ex:

${header.cookie}

headerValues:

This implicit object can be used to access more than one value associated with a single request header.

Ex:

${headerValues.accept[0]}

pageContext:

To get all the implicit objects and to get the values from either of the scopes like pageScope,requestScope,sessionScope,applicationScope we will use pageContext implicit object.

Ex:

${pageContext.servletContext.servletInfo}

Expression Language Functions:

The main purpose of language functions is to perform a particular action in web applications.

In web applications we will utilise expression language functions as an alternative to JSP custom actions.

To prepare functions in expression language,we have to use the following steps:

1)Define a function by taking Java class under classes floder:

In expression language functions,we have to define a java method as a static method with the required implementation.

DURGASOFT

Ex:

```
public class Hello
{
public static String sayHello(String name)
{
return "Good Morning..."+name;
}
}
```

2)Configure expression language function in TLD file:

```
<taglib>
------
------
<function>
<name>function_name</name>
<function_class>Class_name</function_class>
<function_signature>method_signature</function_signature>
</function>
</taglib>
```

3)Access the function from JSP page:

To access the function in JSP page,we have to use the following syntax:

```
${fn:function_name([param_list])}
```

Ex:

```
${fn:sayHello("Nag")}
```


DURGASOFT

Ex:

```
firstapp
|
|---------first.jsp
|
|---------WEB-INF
|                    |
|                    |----hello.tld
|                    |----classes
|                    |              |---com.dss.Hello.class
|                    |              |
|
```

Hello.class

```
package com.dss;
public class Hello
{
public static String sayHello(String name)
{
return name;
}
}
```


DURGASOFT

hello.tld:

```
<taglib>
<jsp-version>2.1</jsp-version>
<tlib-version>1.0</tlib-version>
<function>
<name>sayHello</name>
<function-class>com.dss.Hello</function-class>
<function-signature>java.lang.String sayHello(java.lang.String)</function-signature>
</function>
</taglib>
```

first.jsp:

```
<%taglib uri="/WEB-INF/hello.tld" prefix="fn"%>
<html>
<body>
<b><fint size="7"><br><br>
${fn:sayHello("durga")}
</font></b>
</body>
</html>
```

DURGASOFT

JSP INTERVIEW QUESTIONS:

1. What is JSP ? Describe its concept.

2 . Explain the benefits of JSP?

3. Is JSP technology extensible?

4 .Can we implement an interface in a JSP?

5 What are the advantages of JSP over Servlet?

6. Differences between Servlets and JSP?

7 . Explain the differences between ASP and JSP?

8 . Can I stop JSP execution while in the midst of processing a request?

9. How to Protect JSPs from direct access ?

10. Explain JSP API ?

11. What are the lifecycle phases of a JSP?

12. Explain the life-cycle mehtods in JSP?

13. Difference between _jspService() and other life cycle methods.

14 What is the jspInit() method?

15. What is the _jspService() method?

16. What is the jspDestroy() method?

17. What JSP lifecycle methods can I override?

18. How can I override the jspInit() and jspDestroy() methods within a JSP page?

19 . Explain about translation and execution of Java Server pages?

20 . Why is _jspService() method starting with an '_' while other life cycle methods do not?

21. How to pre-compile JSP?

22. The benefits of pre-compiling a JSP page?

23.How many JSP scripting elements and explain them?

DURGASOFT

24. What is a Scriptlet?

25. What is a JSP declarative?

26. How can I declare methods within my JSP page?

27. What is the difference b/w variable declared inside a declaration and variable declared in scriplet ?

28.What are the three kinds of comments in JSP and what's the difference between them?

29. What is output comment?

30. What is a Hidden Comment?

31. How is scripting disabled?

32. What are the JSP implicit objects?

33. How does JSP handle run-time exceptions?

34. How can I implement a thread-safe JSP page? What are the advantages and Disadvantages of using it?

35. What is the difference between ServletContext and PageContext?

36 . Is there a way to reference the "this" variable within a JSP page?

37 . Can you make use of a ServletOutputStream object from within a JSP page?

38 .What is the page directive is used to prevent a JSP page from automatically creating a session?

39. What's a better approach for enabling thread-safe servlets and JSPs? SingleThreadModel Interface or Synchronization?

40. What are various attributes Of Page Directive ?

41 . Explain about autoflush?

42. How do you restrict page errors display in the JSP page?

43. What are the different scopes available fos JSPs ?

44. when do use application scope?

45. What are the different scope valiues for the ?

46. How do I use a scriptlet to initialize a newly instantiated bean?

47 . Can a JSP page instantiate a serialized bean?

DURGASOFT

48.How do we include static files within a jsp page ?

49.In JSPs how many ways are possible to perform inclusion?

50.In which situation we can use static include and dynamic include in JSPs ?

51.Differences between static include directive and include action ?

DURGASOFT