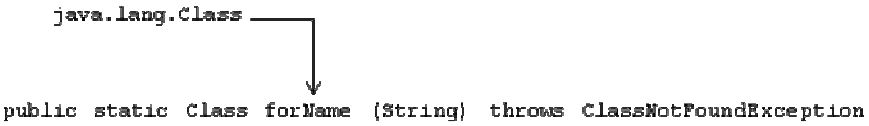
Day - 4:

4) We know the class name at runtime and we have to obtain the runtime information about the class.
To perform the above we must use the method *java.lang.Class* and whose prototype is given below:



When we use the *forName* as a part of java program it performs the following operations:

- It can create an object of the class which we pass at runtime.
- It returns runtime information about the object which is created.

For example:

```
try
{
        Class c=Class.forName ("java.awt.Button");
}
catch (ClassNotFoundException cnfe)
{
        System.out.println ("CLASS DOES NOT EXIST...");
}
```

forName is taking String as an argument. If the class is not found *forName* method throws an exception called *ClassNotFoundException*.

Here, *forName* method is a **factory method** (a *factory method* is one which return type is similar to name of the class where it presents).

Every factory method must be static and public. The class which contains a factory method is known as **Singleton** class (a java class is said to be *Singleton* class through which we can create single object per JVM).

For example:

java.lang.Class is called Singleton class

Write a java program **to find name of the class** and **its super class name** by passing the class name at runtime?

Answer:

```
class ref1
{
        public static void main (String [] args)
        {
                if (args.length==0)
                {
                        System.out.println ("PLEASE PASS THE CLASS NAME..!");
                }
                else
                {
                        try
                        {
                                Class c=Class.forName (args [0]);
                                printSuperclass (c);
```

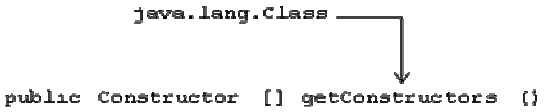
```
            }
        }
};
```

Output:

```
java Hierarchy java.awt.TextField
java.awt.TextField
java.awt.TextComponent
java.awt.Component
java.lang.Object
```

Obtaining information about CONSTRUCTORS which are present in a class:

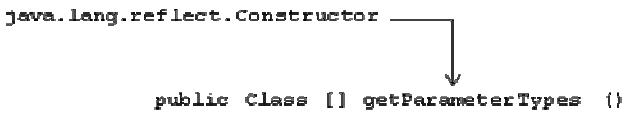
In order to get the constructor of the current class we must use the following method:



For example:

```
Constructor cons []=c.getConstructors ();
System.out.println ("NUMBER OF CONSTRUCTORS = "+cons.length);
```

In order to get the parameters of the constructor we must use the following method:



For example:

```
Class ptype []=cons [0].getParameterTypes ();
```

Day - 5:

Write a java program to obtain constructors of a class?

Answer:

```
class ConsInfo
{
      public static void main (String [] args)
      {
            if (args.length==0)
            {
                  System.out.println ("PLEASE PASS THE CLASS NAME..!");
            }
            else
            {
                  try
                  {
                        Class c=Class.forName (args [0]);
                        printConsts (c);
                  }
                  catch (ClassNotFoundException cnfe)
                  {
                        System.out.println (args [0]+" DOES NOT EXISTS...");
                  }
            }
      }
      static void printConsts (Class c)
```

Create an oracle procedure which takes two input numbers and it must return sum of two numbers, multiplication and subtraction?

Answer:

```
create or replace procedure proc2 (a in number, b number, n out number, n2 out
number, n3 out number)
as
begin
	n1:=a+b;
	n2:=a*b;
	n3:=a-b;
end;
/
```

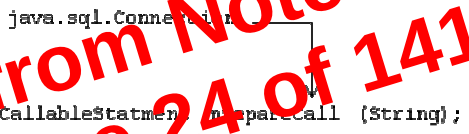
- A *function* is one which contains n number of block of statements to perform some operation and it returns a single value only.

Syntax for creating a function:

```
create or replace function (a in number, b in number) return <return type>
as
	n1 out number;
begin
	n1:=a+b;
	return (n1);
end;
/
```

In order to execute the stored procedures from *jdbc* we must follow the following steps:

1. Create an object of *CallableStatement* by using the following method:



Here, String represents a call for calling a stored procedure from database environment.

2. Prepare a call either for a function or for a procedure which is residing in database.

Syntax for calling a function:

```
"{? = call <name of the function> (?,?,?….)}"
```

For example:

```
CallableStatement cs=con.prepareCall ("{? = call fun1 (?,?)}");
```

The positional parameters numbering will always from left to right starting from 1. In the above example the positional parameter-1 represents **out** parameter and the positional parameter-2 and parameter-3 represents **in** parameters.

Syntax for calling a procedure:

```
"{call <name of the procedure> (?,?,?….)}"
```

For example:

```
CallableStatement cs=con.prepareCall ("{call fun1 (?,?,?,?,?)}");
```

3. Specify which input parameters are by using the following generalized method:

```
Public void setXXX (int, XXX);
```

For example:

```
cs.setInt (2, 10);
```

```
cs.setInt (3, 20);
```

4. Specify which output parameters are by using the following generalized method:

In *jdbc* we have a predefined class called java.sql.Types which contains various data types of *jdbc* which are equivalent to database data types.

Java	*Jdbc*	Database
int	*INT*EGER	number
String	VARCHAR	varchar2
Short	TINY *INT*EGER	number
Byte	SMALL *INT*EGER	number

All the data members which are available in Types class are belongs to **public static final data members**.

For example:

```
cs.registerOutParameter (1, Types.INTEGER);
```

5. Execute the stored procedure by using the following method:

For example:

```
cs.execute ();
```


6. Get the values of out parameters by using the following method:

public XXX getXXX (*int*);

Here, *int* represents position of out parameter. XXX represents fundamental data type or string or date.

For example:

int x=cs.get*Int* (1);

System.out.println (x);

Day - 14:

Write a java program which illustrates the concept of function?

Answer:

StuFun:

```
create or replace function StuFun
(a in number, b in number, n1 out number) return number
as
```

```
                con.close ();
            }
            catch (Exception e)
            {
                System.out.println (e);
            }
        }// main
};// ScrollResultSet
```

Day - 18:

Updatable *ResultSet*:

Whenever we create a *ResultSet* object which never allows us to update the database through *ResultSet* object and it allows retrieving the data only in forward direction. Such type of *ResultSet* is known as non-updatable and non-scrollable *ResultSet*.

In order to make the *ResultSet* object as updatable and scrollable we must use the following constants which are present in *ResultSet int*erface.

***int* Type**	***int* Mode**
TYPE_SCROLL_SENSITIVE	CONCUR_UPDATABLE

The above two constants must be specified while we are creating Statement object by using the following method:

For example:

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

On *ResultSet* we can perform the following three operations, they are **inserting** a record, **deleting** a record and **updating** a record.

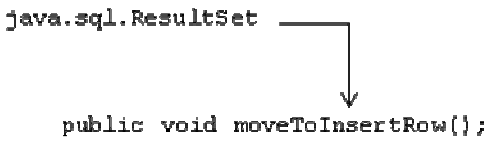
Steps for INSERTING a record through *ResultSet* object:

1. Decide at which position we are inserting a record by calling absolute method.

For example:

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

2. Since we are inserting a record we must use the following method to make the *ResultSet* object to hold the record.



For example:

```
rs.moveToInsertRow ();
```

Servlet Hierarchy:

- In the above hierarchy chart Servlet is an *int*erface which contains three life cycle methods without def*init*ion.
- GenericServlet is an abstract class which implements Servlet *int*erface for defining life cycle methods i.e., life cycle methods are defined in GenericServlet with null body.
- Using GenericServlet class we can develop protocol independent applications.
- HttpServlet is also an abstract class which extends GenericServlet and by using this class we can develop **protocol dependent applications**.
- To develop our own servlet we must choose a class that must extends either GenericServlet or HttpServlet.

LIFE CYCLE METHODS of servlets:

In servlets we have three life cycle methods, they are

```
public void init ();
public void service (ServletRequest req, ServletResponse res);
public void destroy ();
```

public void *init* ():

Whenever client makes a request to a servlet, the server will receive the request and it automatically calls *init* () method i.e., *init* () method will be called only one time by the server whenever we make first request.

In this method, we write the block of statements which will perform one time operations, such as, opening the file, database connection, *in*itialization of parameters, etc.

public void service (ServletRequest, ServletResponse):

After calling *init* () method, service () method will be called when we make first request from second request to further subsequent requests, server will call only service method. Therefore, service () method will be called each and every time as and when we make a request.

In service () method we write the block of statements which will perform repeated operations such as retrieving data from database, retrieving data from file, modifications of parameters data, etc. Hence, in service () method we always write business logic.

Whenever control comes to service () method the server will create two objects of **ServletRequest** and **ServletResponse** *int*erfaces.

Object of ServletRequest contains the data which is passed by client. After processing client data, the resultant data must be kept in an object of ServletResponse.

An object of ServletRequest and ServletResponse must be used always within the scope of service () method only i.e., we cannot use in *init* () method and destroy () method.

Once the service () method is completed an object of ServletRequest and an object of ServletResponse will be destroyed.

public void destroy ():

The destroy () method will be called by the server in two situations; they are **when the server is closed** and **when the servlet is removed from server context**. In this method we write the block of statements which are obtained in *init* () method.

```
            pw.println ("<h1> WELCOME TO SERVLETS <h1>");
            pw.println ("<h2> CURRENT DATE & TIME IS : "+s+"<h2>");
      }
      public void destroy ()
      {
            System.out.println ("I AM FROM destroy METHOD...");
      }
};
```

web.xml:

```
<web-app>
      <servlet>
            <servlet-name>kalyan</servlet-name>
            <servlet-class>ds.DateServ</servlet-class>
      </servlet>
      <servlet-mapping>
            <servlet-name>kalyan</servlet-name>
            <url-pattern>/suman</url-pattern>
      </servlet-mapping>
</web-app>
```

Day - 24:

HttpServlet:

- HttpServlet is the sub-class of GenericServlet.
- HttpServlet contains all the life cycle methods of GenericServlet and the service () method of GenericServlet is further divided *into* the following two methods, they are
  public void doGet (*HttpServletRequest*, HttpServletResponse) throws ServletException, IOException
  public void doPost (*HttpServletRequest*, HttpServletResponse) throws ServletException, IOException

- Whenever client makes a request the **servlet container** (server) will call service () method, the service () method depends on type of the method used by the client application.
- If client method is *get* then service () method will call doGet () method and doGet () method *int*ernally creates the objects of *HttpServletRequest* and HttpServletResponse. Once doGet () method is completed its execution, the above two objects will be destroyed.

LIMITATIONS of *get* method:

1. Whatever data we sent from client by using *get* method, the client data will be populated or appended as a part of URL.
   For example:
   http://localhost:7001/servlet/DDservlet?uname=scott&pwd=tiger
2. Large amount of data cannot be transmitted from client side to server side.

- When we use *post* method to send client data, that data will be send as a part of method body and *int*ernally the service () method will called doPost () method by creating the objects of *HttpServletRequest* and HttpServletResponse.

ADVANTAGES of *post* method:

1. Security is achieved for client data.
2. We can send large amount of data from client to server.

```
        public doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
        {
                ……
                ……
                ServletContext ctx=this.getServletContext ();
                ……
                ……
        }
};
```

By using *ServletConfig* interface:

In *ServletConfig* interface we have the following method which gives an object of *ServletContext*.

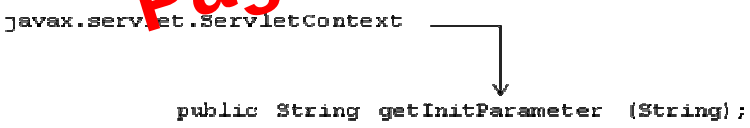
In order to call the above method first we must obtain an object of *ServletConfig* interface and later with that object we can call get*ServletContext* () method.

For example:

```
public class Serv2 extends HttpServlet
{
        public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
        {
                ……
                ……
                ServletConfig config=this.getServletConfig ();
                ServletContext ctx=config.getServletContext ();
                …….
                …….
        }
};
```

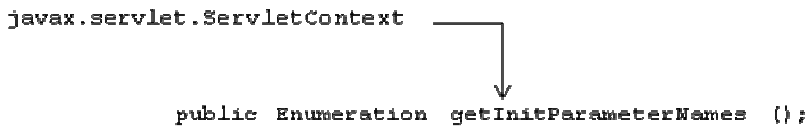

Number of ways to RETRIEVE THE DATA from an OBJECT of *ServletContext*:

In *ServletContext* interface we have the following methods to retrieve the value of context parameter by passing context parameter name.



For example:

```
ServletContext ctx=getServletContext ();
String val1=ctx.getInitParameter (“v1”);
String val2=ctx.getInitParameter (“v2”);
```



For example:

```
ServletContext ctx=getServletContext ();
Enumeration en=ctx.getInitParameterNames ();
While (en.hasMoreElements ())
{
      String cpn= (String) en.nextElement ();
      String cpv=ctx.getInitParameter (cpn);
      pw.println (cpv+” is the value of ”+cpn);
}
```

```
float smarks=0;
if ((sno1==null)||(sno1.equals ("")))
{
	al.add ("PROVIDE STUDENT NUMBER...");
}
else
{
	try
	{
		sno=Integer.parseInt ("sno1");
	}
	catch (NumberFormatException nfe)
	{
		al.add ("PROVIDE int DATA IN STUDENT NUMBER...");
	}
}
if ((sname==null)||(sname.equals ("")))
{
	al.add ("PROVIDE STUDENT NAME...");
}
if ((smarks1==null)||(smarks1.equals ("")))
{
	al.add ("PROVIDE STUDENT MARKS...");
}
else
{
	try
	{
		smarks=Float.parseFloat ("smarks1");
	}
	catch (NumberFormatException nfe)
	{
		al.add ("PROVIDE float DATA IN STUDENT MARKS...");
	}
}
if (al.size ()!=0)
{
	pw.println (al);
}
else
{
	try
	{
		Class.forName ("oracle.jdbc.driver.OracleDriver");
		Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
Hanuman","scott","tiger");

		PreparedStatement ps=con.prepareStatement ("insert into Student values (?,?,?)");
		ps.setInt (1, sno);
		ps.setString (2, sname);
		ps.setFloat (3, smarks);
		int i=ps.executeUpdate ();
		if (i>0)
		{
			pw.println ("RECORD INSERTED...");
		}
		else
```

What is the difference between getRequestDispatcher (String) and getNamedRequestDispatcher (String)?

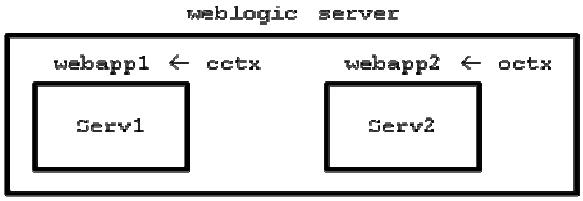
Answer:

getRequestDispatcher (String) method takes **url-pattern** or **public-url** of *web.xml* where as *getNamedRequestDispatcher (String)* method takes **name of the servlet** or **deployer name** of *web.xml*

Forwarding or Including request and response of one web-app to another web-app:

In order to achieve forwarding or including the request and response objects of one web application to another web application, we must ensure that both the web applications must run in the same servlet container.

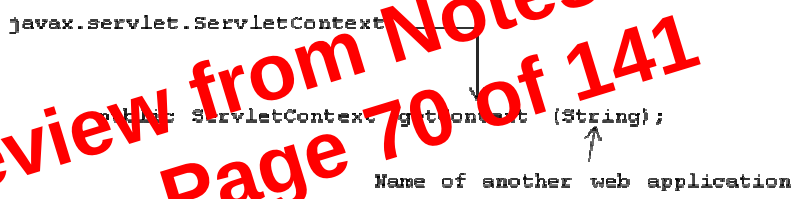
1. Obtain ServletContext object for the current web application.



For example:

```
ServletContext cctx=getServletContext ();
```

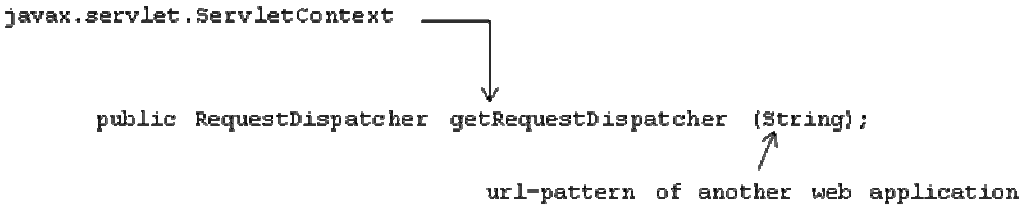
2. Obtain SerletContext object of another web application by using the following method which is present in ServletContext interface.



For example:

```
ServletContext octx=cctx.getContext ("./webapp2");
```

3. Obtain RequestDispatcher object by using ServletContext object of another web application.



For example:

```
RequestDispatcher rd=octx.getRequestDispatcher ("/s2");
```

4. Use either include of forward to pass request and response objects of current web application.

For example:

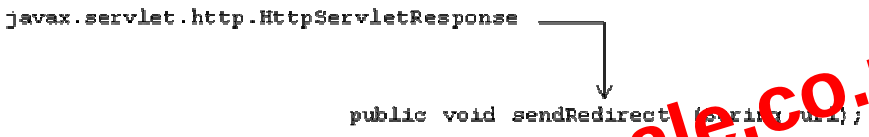
```
rd. include (req, res);
rd.forward (req, res);
```

```
public class Serv2 extends HttpServlet
{
        public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
        {
                res.setContentType ("text/html");
                PrintWriter pw=res.getWriter ();
                pw.println ("<h6>I AM FROM Serv2 OF webapp2</h6>");
        }
        public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
        {
                doGet (req, res);
        }
};
```

Forwarding request and response objects of one web application to another web application and both the web applications are running in different servlet containers:

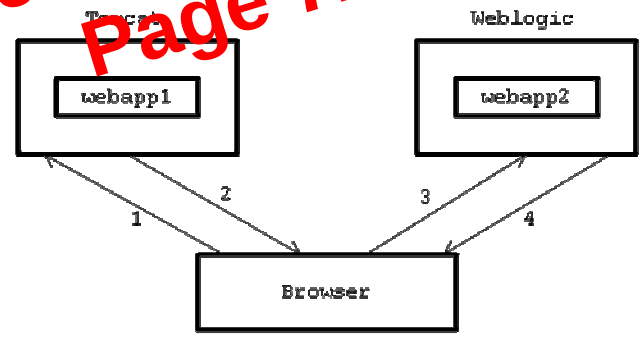
In order to send the request and response object of one web application which is running in on servlet container to another web application which is running in another servlet container, we cannot use forward and include methods.

To achieve the above we must use a method called sendRedirect (String url) method whose prototype is



The above method is present in HttpServletResponse interface, the parameter String url represents url of another web application which is running in some other servlet container.

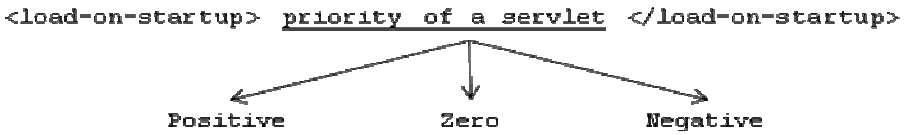
The following diagram will illustrate the concept of sendRedirect method:



1. Make a request to a servlet or JSP which is running in a web application of one container http://localhost:2008/webapp1/s1 context path or document root of one web application.
2. Servlet of web application of Tomcat will redirect the request of client to another web application of Weblogic by using the following statement:
   `Res.sendRedirect ("http://localhost:7001/webapp2/s2");` must be written in Serv1 of webapp1
3. Browser will send the request to another web application of another servlet container.
   For example:
   http://localhost:7001/webapp2/s2 which is redirected by Serv1 of webapp1.
4. Webapp1 gives the response of Serv2 only but not Serv1 servlet.

Load-on-startup:

Load-on-startup is basically used for giving equal response for all the clients who are accessing a particular web application. By default after making request the ServletContext object will be created by servlet container. Because of this first response takes more amount of time and further responses will take minimum amount of time. Therefore to avoid the discrepancy in response time we use a concept of load-on-startup. <load-on-startup> tag will be used as a part of <servlet> tag since it is specific to the servlet.



If the priority value is positive for a group of servlets then whose objects will be created based on ascending order of the priorities. If the priority value is zero then that servlet object will be created at the end. If the priority value of a servlet is negative then that servlet object will not be created i.e., neglected.

When we use load-on-startup as a part of web.xml the container will create an object of a servlet before first request is made.

web.xml:

```
<web-app>
	<servlet>
		<servlet-name>abc</servlet-name>
		<servlet-class>DdServ</servlet-class>
		<load-on-startup>10</load-on-startup>
	</servlet>
	<servlet-mapping>
		<servlet-name>abc</servlet-name>
		<url-pattern>/ddurl</url-pattern>
	</servlet-mapping>
</web-app>
```


DdServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DdServ extends HttpServlet
{
	public DdServ ()
	{
		System.out.println ("SERVLET OBJECT IS CREATED");
	}
	public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
	{
		res.setContentType ("text/html");
		PrintWriter pw=res.getWriter ();
		System.out.println ("I AM FROM doGet ()");
		pw.println ("<h3>I AM FROM doGet ()</h3>");
	}
};
```

```
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>s2</servlet-name>
		<url-pattern>/test2</url-pattern>
	</servlet-mapping>
	<servlet-mapping>
		<servlet-name>s3</servlet-name>
		<url-pattern>/test3</url-pattern>
	</servlet-mapping>
</web-app>
```

personal.html:

```
<html>
  <title>Complete example</title>
  <body>
	<center>
	<form name="ex" action="./test1" method="post">
	  <table border="1">
		<tr>
		  <th align="left">Enter ur name : </th>
		  <td><input type="text" name="ex_name" value=""></td>
		</tr>
		<tr>
		  <th align="left">Enter ur address : </th>
		  <td><textarea name="ex_add" value=""></textarea></td>
		</tr>
		<tr>
		  <th align="left">Enter ur age : </th>
		  <td><input type="text" name="ex_age" value=""></td>
		</tr>
		<tr>
		  <table>
			<tr>
			  <td align="center"><input type="submit" name="ex_continue" value="Continue"></td>
			</tr>
		  </table>
		</tr>
	  </table>
	</form>
	</center>
  </body>
</html>
```


Database table (info):

```
create table info
(
	name varchar2 (13),
	addr varchar2 (33),
	age number (2),
	exp number (2),
	skills varchar2 (13),
	sal number (7,2),
	loc varchar2 (17)
);
/
```

```
                pw.println ("</td></tr><table><tr><td align=\"center\"><input type=\"submit\"
name=\"second_submit\" value=\"Submit\">");
                pw.println ("</td></tr></table></table></form></center></body></html>");
        }
        public void doPost (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
        {
                doGet (req, res);
        }
};
```

FinalServ.java:

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
import java.util.*;
public class FinalServ extends HttpServlet
{
        public void doGet (HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException
        {
                res.setContentType ("text/html");
                PrintWriter pw=res.getWriter ();
                String sal1=req.getParameter ("second_sal");
                float salary=Float.parseFloat (sal1);
                String location=req.getParameter ("second_loc");

                HttpSession hs=req.getSession (false);

                String name=(String) hs.getAttribute ("namehs");
                String address=(String) hs.getAttribute ("addresshs");
                String age1=(String) hs.getAttribute ("agehs");
                int age=Integer.parseInt (age1);
                String exp=(String) hs.getAttribute ("exphs");
                int experiance=Integer.parseInt (exp);
                String skills=(String) hs.getAttribute ("skillshs");
                try
                {
                        Class.forName ("oracle.jdbc.driver.OracleDriver");
                        Connection con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:
Hanuman","scott","tiger");
                        PreparedStatement ps=con.prepareStatement ("insert into info values (?,?,?,?,?,?,?)");
                        ps.setString (1, name);
                        ps.setString (2, address);
                        ps.setInt (3, age);
                        ps.setInt (4, experiance);
                        ps.setString (5, skills);
                        ps.setFloat (6, salary);
                        ps.setString (7, location);
                        int j=ps.executeUpdate ();
                        if (j>0)
                        {
                                pw.println ("<html><body bgcolor=\"lightblue\"><center><h1><font color=\"red\">
Successfully ");
                                pw.println ("Inserted</font></h1></center><a href=\"personal.html\">Home</a>
</body></html>");
```

will be placed automatically in `out.println ()` method and this method is available as a part of service method.

NOTE: Expressions in the expression tag should not be terminated by semi-colon (;) .

For example-1:

```
<%! int a=10, b=20 %>
<%= a+b %>
```

The equivalent servlet code for the above expression tag is `out.println (a+b);` out is implicit object of *JSPWriter* class.

For example-2:

```
<%= new java.util.Date () %> ➔ out.println (new java.util.Date ());
```

3. **Scriplet tag:**

Scriplets are basically used to write a pure java code. Whatever the java code we write as a part of scriplet, that code will be available as a part of `service ()` method of servlet.

Syntax:

```
<% pure java code %>
```

Write a scriplet for generating current system date?

Answer:

web.xml:

```
<web-apps>
</web-apps>
```

DateTime.java:

```
<html>
	<title>Current Date & Time</title>
	<head><h4>Current date & time</h4></head>
	<body>
		<%
			Date d=new Date ();
			String s=d.toString ();
			out.println (s);
		%>
	</body>
</html>
```

[or]

```
<html>
	<title>Current Date & Time</title>
	<head><h4>Current date & time</h4></head>
	<body>
		<%=new Date ()%>
	</body>
</html>
```

Write a JSP page to print 1 to 10 numbers? [For *web.xml* refer page no: 102]

Answer:

One2TenNumbers.jsp:

```
<html>
	<title>Print Numbers 1-10</title>
	<head>Numbers 1-10</head><br>
```

```
                java.util.Date d=new java.util.Date ();
            %>
            <h4>Current date & time</h4>
            <h3><%= d %></h3>
      </body>
</html>
```

Write a JSP page which will display number of times a request is made [write a JSP for hit counter]? [For *web.xml* refer page no: 102]

Answer:

ReloadPageCount.jsp:

```
<html>
      <title>Number of Reloads</title>
      <head>Number of visitings to a browser</head><br>
      <body>
            <%! int ctr=0; %>
            <%!
                  int count ()
                  {
                        return (++ctr);
                  }
            %>
            <h3><%= count () %></h3>
      </body>
</html>
```

NOTE:

Within servlet we use to write html code to generate presentation logic whereas in JSP environment within html program we are making use of JSP tags.

Write a JSP page which will retrieve the data from database? [For *web.xml* refer page no: 102]

Answer:

```
<%@ page import="java.sql.*,java.io.*" %>
<html>
  <title>Data From Database</title>
  <head>Retrieve data from Datebase</head>
  <body>
      <%!
        Connection con=null;
        Statement st=null;
        public void jspInit ()
        {
            try
            {
              Class.forName ("oracle.jdbc.driver.OracleDriver");
              con=DriverManager.getConnection ("jdbc:oracle:thin:@localhost:1521:Hanuman","scott",
"tiger");
              st=con.createStatement ();
            }
            catch (Exception e)
            {
              out.println (e);
            }
        }
```

Login.html:

```
<html>
  <head><center><h3>Login Page</h3></center></head>
  <body>
  <center>
       <h4>Forward/Include test</h4>
       <form name="login" action="Login.jsp" method="post">
         <p>Enter username : <input type="text" name="login_uname" value=""><br>
         Enter password : <input type="password" name="login_pwd" value=""><br>
         <input type="submit" value="Login">
       </form>
  </center>
  </body>
</html>
```

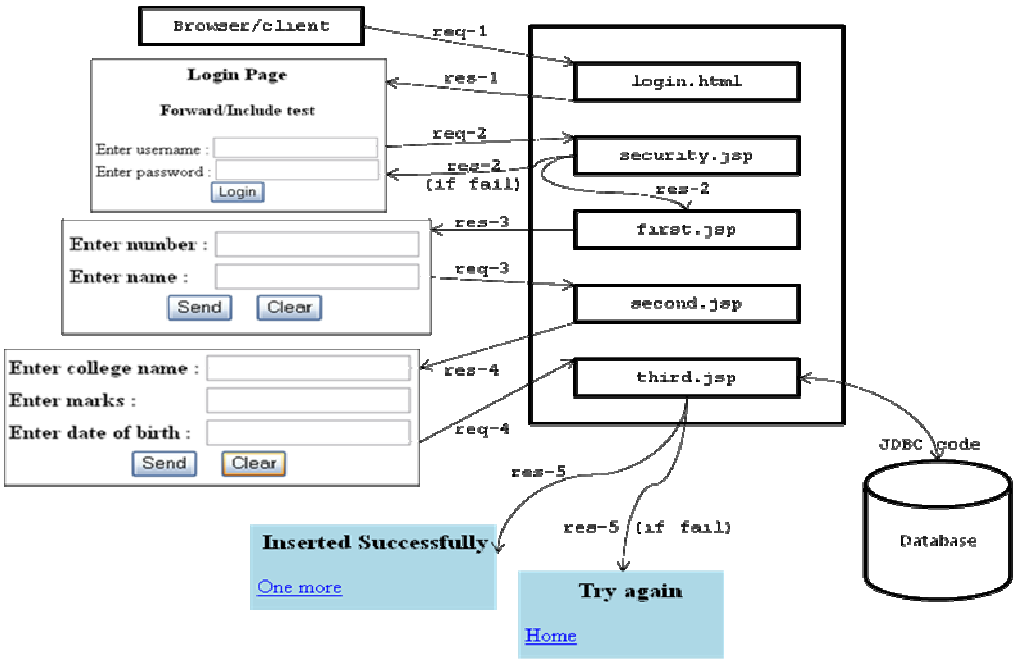
Login.jsp:

```
<%     String s1=request.getParameter ("login_uname");
       String s2=request.getParameter ("login_pwd");
       if (s1.equals ("kalpana") && s2.equals ("test"))
       {      %>
       <JSP:forward page="Success.jsp"/>
<%     }
       else
       {      %>
       <h5>Login failed</h5>
       <JSP:include page="Login.html"/>
<%     }      %>
```

Success.jsp:

```
<h5>Login Successful</h5>
Welcome to : 
<h4><%= request.getParameter ("login_name") %></h4>
```


Develop the following form?



```
	{
		out.println (e);
	}	%>
</html>
```

JavaBeans in JSP

A JavaBeans class is a software reusable component. Every JavaBeans class must belong to a package. Since, it is reusable. Every JavaBeans class modifier must be public. Every JavaBeans class must contain set of data members (known as properties).

For each and every data member of JavaBeans class we must have set of set methods whose general representation is:

```
public void setXxx (datatype FormalVariableName)
{
	……..;
	……..;
	……..;
}
```

For each and every data member of JavaBeans class we must have set of get methods whose general representation is:

```
public datatype getXxx ()
{
	……..;
	……..;
	……..;
}
```

The set of set methods are used for setting the values to JavaBeans class object whereas set of get methods are used for getting the values from JavaBeans class object.

Properties or characteristics of JavaBeans:

Every JavaBeans class contains **simple property, boolean property** and **indexed properties**.

- A **simple property** is one in which a method takes and returns elementary or single value (set of set and get methods are known as *simple properties* of a JavaBeans class.)
- A **boolean property** is one in which a method takes or return boolean value.
- An **indexed property** is one in which a method takes or return array of values.

Develop a JavaBeans class which will check whether the username and password correct or not?

Answer:

Test.java:

```
package abc;
public class Test
{
	String uname;
	String pwd;
	public void setUname (String uname)
	{
		this.uname=uname;
	}
	public void setPwd (String pwd)
```

- **request** - represents a JavaBeans class object can be accessed in those JSP pages which are participating in processing a single request.
- **session** - represents a JavaBeans class object can be accessed in all the JSP pages which are participating in a session and it is not possible to access in those JSP pages which are not participating in session.
- **application** - represents a JavaBeans class object can be accessed in all the JSP pages which belongs to the same web application but it is not possible to access in those JSP pages which are belongs to other web applications.

The **type** attribute represents specification of base interface or class name of a JavaBeans class.

For example:

```
<JSP:useBean id="eo"
             class="ep.Emp"
             scope="session"
             type="ep.GenEmp" />
```

When the above statement is executed the container creates an object eo is created in the following way:

```
ep.GenEmp eo=new ep.Emp ();
```

If we are not specifying the value for type attribute then the object eo is created in the following way:

```
ep.Emp eo=new ep.Emp ();
```

NOTE:

In the above `<JSP:useBean/>` tag if we use a tag called `<JSP:setProperty/>` then that tag becomes body tag of `<JSP:useBean/>` tag.

5. `<JSP:setProperty/>`:

This tag is used for setting the values to the JavaBeans class object created with respect to `<JSP:useBean/>` tag.

Syntax-1:

```
<JSP:setProperty      name="object name of a JavaBeans class"
                      Property="property name of JavaBeans class"
                      Value="value for property" />
```

For example:

```
<JSP:useBean id="eo" class="ep.Emp">
      <JSP:setProperty   name="eo"
                         Property="empno"
                         Value="123" />
</JSP:useBean>
```

When the above statement is executed by the container, the following statement will be taken place.

```
ep.Emp eo=new ep.Emp ();
eo.setEmpno ("123");
```

```
                        }
                        while (rs.next ())
                        {
                %>
                <tr>
                <%
                                j=1;
                                while (j<=ColCount)
                                {
                %>
                <td>
                <%              out.print (rs.getString (j));          %>
                </td>
                <%
                                        j++;
                                }
                %>
                </tr>
                <%
                                rs.close ();
                                con.close ();
                        }
                        catch (Exception e)
                        {
                                e.printStackTrace ();
                        }
                %>
        </body>
</html>
```


Day - 58:

SWINGS

- In the earlier days SUN micro system; we have a concept called awt.
- awt is used for creating GUI components.
- All awt components are written in 'C' language and those components appearance is changing from one operating system to another operating system. Since, 'C' language is the platform dependent language.
- In later stages SUN micro system has developed a concept called swings.
- Swings are used for developing GUI components and all swing components are developed in java language.
- Swing components never change their appearance from one operating system to another operating system. Since, they have developed in platform independent language.

Differences between awt and swings:

Awt	Swings
1. awt components are developed in 'C' language.	1. Swing components are developed in java language.
2. All awt components are platform dependent.	2. All swing components are platform independent.

3. All awt components are heavy weight components, since whose processing time and main memory space is more.	3. All swing components are light weight components, since its processing time and main memory space is less.

NOTE: All swing components in java are preceded with a letter 'J'.

For example:

```
JButton JB=new JButton ("ok");
```

All components of swings are treated as classes and they are belongs to a package called javax.swing.*

In swings, we have two types of components. They are **auxiliary components** and **logical components**.

- *Auxiliary components* are those which we can touch and feel. For example, mouse, keyboard, etc.
- *Logical components* are those which we can feel only.

Logical components are divided into two types. They are **passive or inactive components** and **active or interactive components**.

- *Passive components* are those where there is no interaction from user. For example, JLabel.
- *Active components* are those where there is user interaction. For example, JButton, JCheckbox, JRadioButton, etc.

✓ In order to provide functionality or behavior to swing GUI active components one must import a package called java.awt.event.*

✓ This package contains various classes and interfaces which provides functionality to active components.

✓ EDM is one which always provides the functionality to GUI active components.

Steps in EDM:

1. Every GUI active component can be processed in two ways. They are **based on name or label of the component** and **based on reference of the component**. Whenever we interact with any GUI component whose reference and label will be stored in one of the predefined class object whose general notation is xxxEvent class.

   For example:

   ```
   JButton → ActionEvent
   JCheckbox → ItemEvent
   ```

2. In order to provide behavior of the GUI component we must write some statements in methods only. And these methods are given by SUN micro system without definition. Such type of methods is known as abstract methods. In general, all abstract methods present in interfaces and those interfaces in swings known as Listenters. Hence, each and every interactive component must have the appropriate Listener whose general notation is xxxListener.

   For example:

   ```
   JButton → ActionListener
   JCheckbox → ItemListener
   ```

```
		<filter-name>abc</filter-name>
		<filter-class>FilterEx</filter-class>
	</filter>
	<servlet>
		<servlet-name>pqr</servlet-name>
		<servlet-class>ServletEx</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>pqr</servlet-name>
		<url-pattern>/Serv</url-pattern>
	</servlet-mapping>
	<filter-mapping>
		<filter-name>abc</filter-name>
		<url-pattern>/Filt</url-pattern>
	</filter-mapping>
</web-app>
```

Day - 61:

Develop a java program which will illustrate the concept of Filters?

Answer:

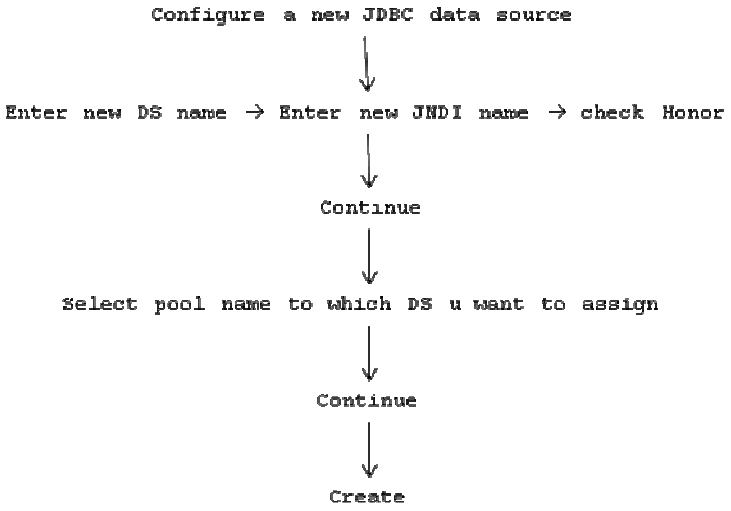
web.xml:

```
<web-app>
	<filter>
		<filter-name>abc</filter-name>
		<filter-class>FilterEx</filter-class>
	</filter>
	<servlet>
		<servlet-name>pqr</servlet-name>
		<servlet-class>ServletEx</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>pqr</servlet-name>
		<url-pattern>/Serv</url-pattern>
	</servlet-mapping>
	<filter-mapping>
		<filter-name>abc</filter-name>
		<url-pattern>/Filt</url-pattern>
	</filter-mapping>
</web-app>
```


FilterEx.java:

```
import javax.servlet.*;
import java.io.*;
public class FilterEx implements Filter
{
	public FilterEx ()
	{
```

Steps for creating data sources:



NOTE:

In a Servlet program we should always use JNDI name which in turns pointing to appropriate data source name and it points to one of named connection in connection pool.

For example:

web.xml:

```
<web-app>
	<servlet>
		<servlet-name>abc</servlet-name>
		<servlet-class>FirstConPoolServ</servlet-class>
	</servlet>
	<servlet-mapping>
		<servlet-name>abc</servlet-name>
		<url-pattern>/ServPool</url-pattern>
	</servlet-mapping>
</web-app>
```

index.html:

```
<html>
	<body bgcolor=lightblue>
		<center>
		<form action="./ServPool">
		Enter table name: <input type="text" name="table" value=""><br>
		<input type="submit" value="Bring data">
		</center>
	</body>
</html>
```

FirstConPoolServ.java:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import javax.sql.*;
import javax.naming.InitialContext;
public class FirstConPoolServ extends HttpServlet
```