Servlets

By Suryanarayana

ActiveNET

Course Content:

i) Introduction to Web application & Web site projects
ii) Java Web application directory structure
iii) Develop, deploy and run first web application in java
iv) Introduction to web server, web container, web application, web document, web component, SevletContext, Servlet, ServletConfig, ServletRequest, ServletResponse, HttpSession etc.
v) Difference b/w request parameters, initialization parameters and context parameters and how to pass, receive them in servlet and process / use them.
vi) One form processing example with all the parameters
vii) Difference b/w stateless and stateful servlets
viii) Different types of servlets named GenericServlet and HttpServlet
ix) Achieving Session management and client state persistence in HttpServlet and one example
x) Different types of session management
1. Cookies, Hidden fields, URL rewriting
xi) Non idem-potency problem and its prevention using token concept
xii) Client state persistence on server using HttpSession object and client state persistence on client system using Cookies.
xiii) xiii) Servlet-Servlet communication/Inter-servlet communication with the web application, across web application using RequestDispatcher class forward() and include() methods, across web servers using Socket programming and other one is using response.sendRedirect()
xiv) xiv) Developing Filters in Servlets
xv) xv) Importance and usage of Listeners
xvi) xvi) Securing Servlets
xvii) xvii) Pagination in Servlets using Scrollable ResultSet and Pager tags
xviii) xviii) JDBC Connection Pooling in Servlets
xix) xix) File upload processing in servlets
xx) xx) InternationalizatioN in web applications using Locale and ResourceBundle classes
xxi) xxi) Sending email from standalone/servlets using SMTP API

xxii) xxii) Receiving emails using POP3 API

What is the difference between ***web site projects*** & ***web application projects***?
A web project which uses only static resources such as .html, .js, .css, .jpg, .gif, .au, .pdf, .swf, .mpg etc are called as ***web site projects***/***static web sites***.

A web project which uses dynamic resources along with static resources is called as "***web application projects***". Dynamic resources are Servlets, JSP, ASP, PHP, ColdFusion CFM, Ruby on Rails, Python Djongo. These dynamic resources always exist and run on server system but only produce dynamic web page output to browser. Static resources such as .html, css, .js, .jpg etc directly downloads onto browser and runs.

The intention of dynamic resources is whenever user submits a form data from the browser, we need a server side resource to read the form data, validate, process and store the data to and from DB. In java we will use JDBC/Hibernate/any ORM framework API to implement servlet/jsp to interact with databasse on server system.

In web site projects static html files are placed in web site root directory, images are placed in \images folder, javascript and css are placed in \script and \style folders and so on.

But whereas in web application projects for each technology the web application directory structure differs. For instance in **java web applications the directory structure is:**

```
hello_1
  index.html
  \script\.js
  \style\.css
  \images\.jpg
  \jsp\.jsp
  \WEB-INF
    web.xml - to configure servlets, filters, listeners etc
    \classes
      all compiled servlet .class files must be placed in this folder
          other than servlets, like filters or listeners or beans or factory must be placed in packages under this folder
    \lib
      3rd party jar files
```

In PHP based web application projects the html files must be placed in public_html folder.

```
css
        style.css
images
includes
        header.php
        footer.php
        db_connect.php
```

functions.php
js
validation.js
lib
plugins
uploads
index.php

In ASP.net web application the project directory contains as follows:

- BIN
- App_Code
- App_GlobalResources
- App_LocalResources
- App_WebReferences
- App_Data
- App_Browsers
- App_Themes

Both web site projects and as well web application projects are deployed on **local or www server**. Their URL is accessed from client's web browser. Static html pages are downloaded, once they are filled and submitted by client the server side resources like servlets and jsp, accept the request, process and responds to clients.

To run server side resources we need both compiler and interpreter because when JSPs requested for the first time JSP container translates them into servlet, compiled by jsp container using javac and then servlet container runs them using java interpreter. Whereas servlets are pre-compiled and placed in web server to run them java interpreter is sufficient.

The classes placed in web application projects must be placed in appropriate namespaces. As we discussed earlier other than servlet class must be placed in package in java web applications.

Develop, deploy and run first web application in java

To DDR web application along with JDK we need one Java Web Server also.

WebServers:

i) Apache – Tomcat
ii) Mortbay - Jetty
iii) Caucho - Resin
iv) Cheyenne Secure Web Server

Obsolote web servers:

i) ServletRunner - Sun
ii) JWS (Java Web Server) - Sun
iii) Atlanta Corp ServletExec
iv) iPlanet - iPlanet Web Server

v) Macromedia - Cold Fusion Server
vi) Allaire Corp - JRun Web Server

Application Servers:
i) IBM - WebSphere AS
ii) BEA/Oracle - WebLogic AS
iii) RedHat - JBoss AS
iv) Pramati - Pramati AS
v) Fujitsu - AS
vi) HP - AS
vii) Fiorana - Fiorana AS
viii) Progress - AS
ix) Oracle - OC4J / BC4J
Oracle Container for Java / Business Container for Java
x) Caucho - Resin AS
xi) Apache - Geronimo
xii) Sun Microsystems - Sun One/GlassFish

Tomcat Installation:

To install Tomcat we need to download the tomcat zip file or installer file.

If installer file is installed, during installation the path will be prompted C:\Program Files\apache-tomcat-6.0.37-windows-x86\apache-tomcat-6.0.37, followed by prompting the password of admin user, port number and JDK installation path.

If ZIP file is extracted then we need to include JAVA_HOME, JRE_HOME and CATALINA_HOME environment variables that tomcat startup process uses.

JAVA_HOME
D:\Program Files\Java\jdk1.6.0_35

CATALINA_HOME
C:\Program Files\apache-tomcat-6.0.37-windows-x86\apache-tomcat-6.0.37

tomcat
bin
startup.bat, shutdown.bat
conf
server.xml, tomcat-users.xml
lib
servlet-api.jar, jsp-api.jar; catalina.jar
only servlet-api.jar file must be included in CLASSPATH
logs
catalina.log, host-manager.log, localhost.log, manager.log
webapps
java web applications like first.war file must be deployed
first.war

E:\active\adv\am36\Servlets\hello_1

index.html

WEB-INF - *directory*

web.xml

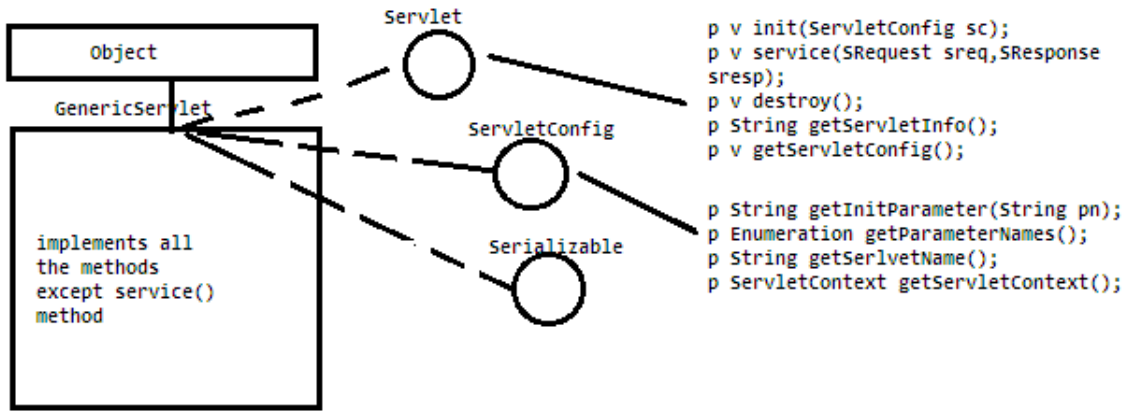
classes - *directory*

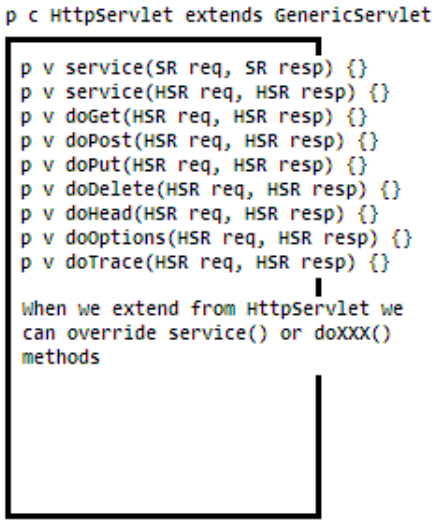
HelloServlet.java, HelloServlet.class

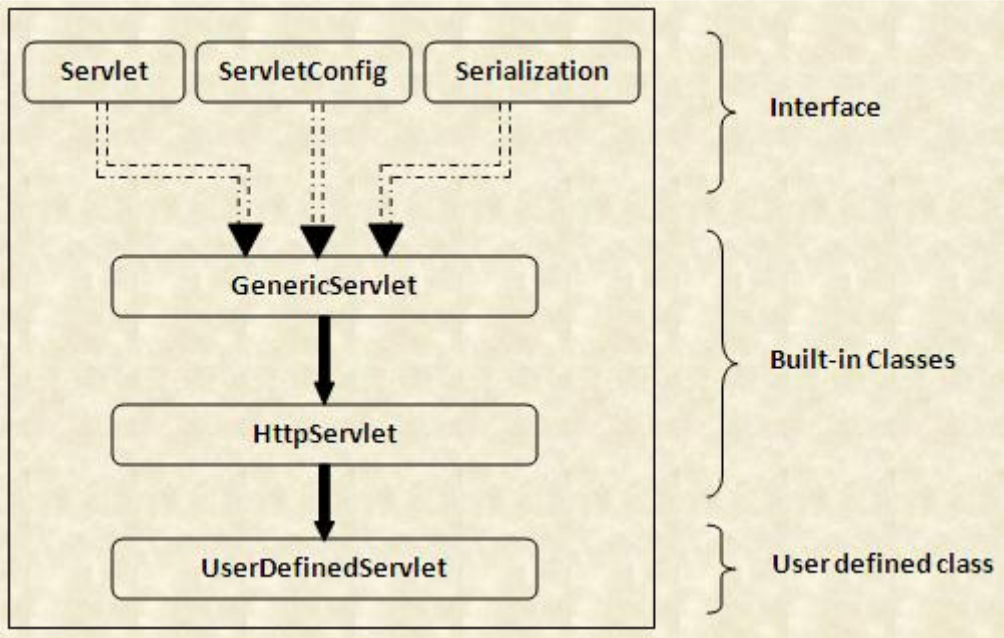
index.html

```
<HTML>
 <BODY>
  <FORM action="./helloServlet">
   Name <INPUT type="text" name="name">
   <INPUT type="submit" name="submit" value="Say Hello">
  </FORM>
 </BODY>
</HTML>
```

To write and compile servlet servlet-api.jar file must be included in CLASSPATH. Servlet API exists in ***javax.servlet*** package. In that one interface is given named ***Servlet***. This interface is having 5 abstract methods that every servlet class must implement.







Instead of implementing from ***Servlet*** interface and implement all the 5 abstract methods, to Servlet interface one adapter class is given named ***GenericServlet*** in which all 4 abstract methods are implemented and one method is left abstract, the abstract method is

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

First Java Web Application

// HelloServlet.java

```
import javax.servlet.*;
import java.io.*;
public class HelloServlet extends GenericServlet {
public void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException {
   String name=req.getParameter("name").trim();
   PrintWriter out=resp.getWriter();
   out.println("Hello "+name);
   out.flush();
   out.close();
 }
}
```

web.xml

```
<web-app>
 <servlet>
  <servlet-name>helloServlet</servlet-name>
  <servlet-class>HelloServlet</servlet-class>
 </servlet>
 <servlet-mapping>
```

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

compile servlet
E:\active\adv\am36\Servlets\first\WEB-INF\classes
javac *.java

make web application or .war (**W**eb **AR**chieve) file
cd..
cd..
jar -cvf am37first.war *.html WEB-INF
.war file is created. War file is also called as java web application.

Jar command options

- c - create a jar
- u - update a jar
- x - extract data from jar
- l - display list of content
- v - verbose output on
- f - specifying file name

deploy am36first.war into webapps directory of tomcat web server.
and click on <tomcat>**\bin\starup.bat**

open browser and type
http://localhost:8082/am37first/index.html

As soon as we deploy first web application and start tomcat web server, tomcat extracts the .war file (Web ARchieve) into the same war named folder. Web applications can be deployed as folders also directly into tomcat webapps directory. But if we want to deploy web application on remote server system, the ftp upload will take time to deploy. Hence if we make .war and deploy, the compressed files will reduce the folder size to more than half of the size of original folder, deployment time reduces.

During tomcat server startup, tomcat starts 3 processes called

- HTTPEngine (Coyote)
- Servlet container (Catalina)
- and JSP Container (Jasper)

When client requests for a static resource (other than servlet & jsp), the web server redirects the call to HTTP engine, HTTP engine reads the web application name, into the corresponding folder the client requested HTML document will be lookuped. If the file is found using input & outputstreams the HTML file content will be sent to browser. Otherwise 404 file not found error is sent to client browser.

Client submits the HTML form data to web server. If the request comes to a servlet, Servlet container runs the servlet. Before running a servlet, at the time of web application deployment or during server startup, servlet container goes to each web application reads, web.xml file and creates one new instance of ServletContext object. ServletContext object contains servlet configuration data. url-pattern mapped to servlet-name, servlet-name mapped to servlet class. Along with this ServletContext also contains context-params passed from web.xml file. Such all web applications ServletContext objects are stored with their respective web application name into one more hashtable.

Servlet instance is created by servlet container in two occasions. If servlet load-on-startup property is mentioned as 1 or 2 or 3. In the load-on-startup property order the servlet instances are created. If not mentioned servlet instance is created during first client request to that servlet. As soon as servlet is instantiated, on the servlet init(ServletConfig sc) method will be called, ServletConfig object is passed as argument into it. ServletConfig contains all the init-params passed into that servlet through web.xml. It also contains getServletContext() method from which we can retrieve context parameters.

Using init and context parameters we can initialize servlet class instance variables such as Connection, Statement, PreparedStatement, Logger, Connection Pool object object DataSource etc.

****

Context parameters are passed by web developer, passed from web.xml file, in servlet init() method using servletConfig.getServletContext() we can obtain ServletContext object reference. In service() method the object must be obtained directly by calling getServletContext() method. Context parameters are passed into servlet only once through init() method via ServletConfig object. Context parameters are retrieved using sctxt.getInitParameter(String paramName).

Init Parameters are also passed by web developer, from web.xml file, passed as ServletConfig object into init() method. init parameters are also retrieved only once in init() method. The parameters can be retrieved using getInitParameter() method.

Request parameters are the form data filled and submitted by client from HTML form. ServletContainer/Web Server reads HTTP request, reads request parameters, stores into ServletRequest/HttpServletRequest object and passed as argument into service() method. Every client passed his/her form data is passed as request parameters. For every client request new ServletRequest and ServletResponse objects are created and passed as argument. Once service() method execution is completed. req and resp objects are destroyed.
****

Including from first client onwards servlet container creates new thread for each client. Passes servlet instance as argument into new thread starts the thread. The Thread class run method will invoke service() method on servlet instance. For each client one new thread is created and servlet instance is executed from multiple threads. Such execution is called as Singleton-MultiThread model.

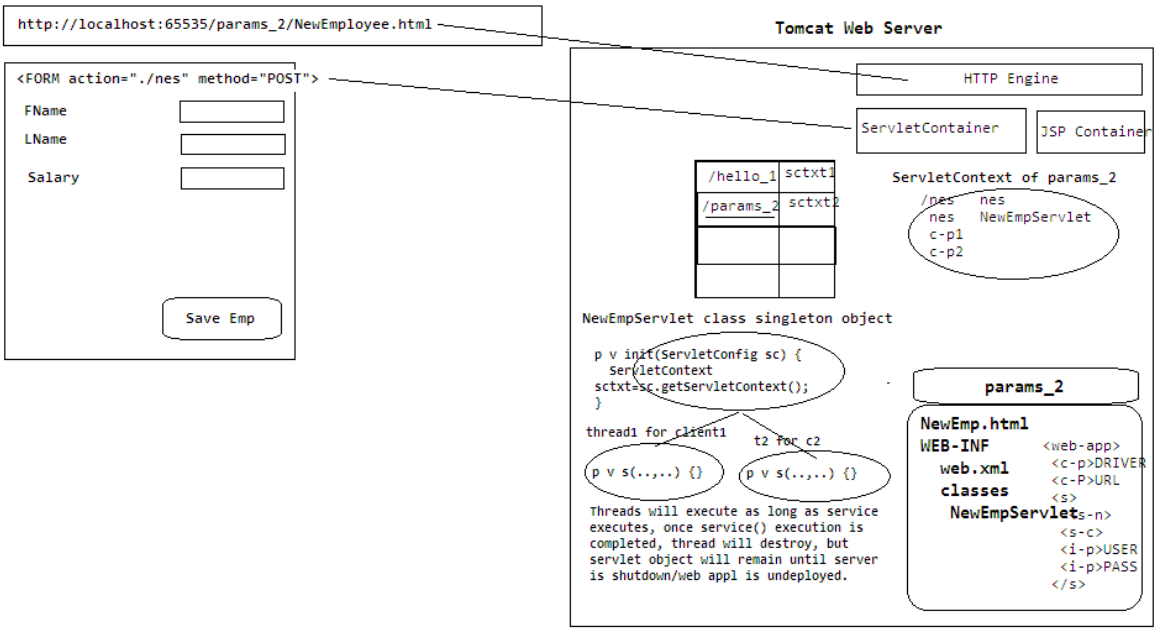
As soon the code written in the service() method execution is completed, the client corresponding thread object is destroyed, but servlet instance remains alive.

Servlet container removes servlet instance in two occasions:

i) During web application undeployment
ii) During web server shutdown

While destroying servlet instance, servlet container will call destroy() method. In the destroy() method we will uninitialize servlet class instance variables.

Second Java Web Application



E:\active\adv\am36\Servlets\params_2

NewEmp.html
NewEmpServlet.java
web.xml
log4j.properties
build.xml

NewEmp.html

```
<HTML>
 <BODY>
  <FORM method="post" action="./newEmpServlet" onSubmit="return validateForm()">
   <TABLE width="50%" align="center">
    <TR>
     <TH>First Name</TH>
     <TD><INPUT type="text" name="first_name" id="first_name"  ></TD>
    </TR>
    <TR>
     <TH>Last Name</TH>
     <TD><INPUT type="text" name="last_name" id="last_name"></TD>
    </TR>
```

```
   <TR>
    <TH>Salary</TH>
    <TD><INPUT type="text" name="salary" id="salary" ></TD>
   </TR>
   <TR>
    <TH>Hire Date <span style="font-size:10px;font-
style:italic;color:red;">(dd/mm/yyyy)</span></TH>
    <TD><INPUT type="text" name="hire_date" id="hire_date" ></TD>
   </TR>
   <TR>
    <TH>Job ID</TH>
    <TD><INPUT type="text" name="job_id" id="job_id" ></TD>
   </TR>
   <TR>
    <TH>Mgr ID</TH>
    <TD><INPUT type="text" name="mgr_id" id="mgr_id" ></TD>
   </TR>
   <TR>
    <TH>Deptno</TH>
    <TD><INPUT type="text" name="deptno" id="deptno" ></TD>
   </TR>
   <TR>
    <TH><INPUT type="submit" name="submit" value="Save Emp"></TH>
    <TD><INPUT type="reset" name="reset" value="Clear Form"></TD>
   </TR>
  </TABLE>
 </FORM>
</BODY>

<HEAD>
<style>
body{
        background-color:"red";

}
</style>
 <SCRIPT>

  var err=" ";

  function validateForm() {
   // window.alert('in validateForm');
   err=validateFirstName();
   err+=validateLastName();
   err+=validateSalary();
```

```
  err+=validateHireDate();
  err+=validateJobId();
  err+=validateMgrId();
  err+=validateDeptNo();

          // window.alert('before if');

          if(err.length>0) {
       window.alert(err);
                    // window.alert('false');
       return false;
      } else {
                    // window.alert('true');
       return true;
      }
}

function validateFirstName() {
  var f_n=window.document.getElementById("first_name");
  var value=f_n.value;
  if(value.length==0)
   return "First Name cannot be null \n";
}
function validateLastName() {
  var f_n=window.document.getElementById("last_name");
  var value=f_n.value;
  if(value.length==0)
   return "Last Name cannot be null \n";
}

function validateSalary() {
  var f_n=window.document.getElementById("salary");
  var value=f_n.value;
  if(value.length==0)
   return "Salary cannot be null \n";
}
function validateHireDate() {
  var f_n=window.document.getElementById("hire_date");
  var value=f_n.value;
  if(value.length==0)
   return "Hire Date cannot be null \n";
}

function validateJobId() {
  var f_n=window.document.getElementById("job_id");
```

```
    var value=f_n.value;
    if(value.length==0)
     return "Job Id cannot be null \n";
   }
   function validateMgrId() {
    var f_n=window.document.getElementById("mgr_id");
    var value=f_n.value;
    if(value.length==0)
     return "Mgr ID cannot be null \n";
   }

   function validateDeptNo() {
    var f_n=window.document.getElementById("deptno");
    var value=f_n.value;
    if(value.length==0)
     return "Deptno cannot be null \n";
   }

  </SCRIPT>
 </HEAD>

</HTML>

<!--
    // window.alert(isNaN(value));
    if(isNaN(value)==true)
     return "Salary is not a double\n";
    /* try {
      parseFloat(value);
    }catch(err) {
     return err+"\n";
    }*/
-->
```

// NewEmpServlet.java

```
import javax.servlet.*;
import java.io.*;
import java.sql.*;
import org.apache.log4j.*;
import org.apache.commons.validator.routines.*;
public class NewEmpServlet extends GenericServlet {
  Connection con;
  Statement stmt;
  PreparedStatement pstmt;
  ResultSet rs;
```

```
 Logger logger;

 public NewEmpServlet() {
  System.out.println("NewEmpServlet() called");
 }

 // first lifecycle method
 public void init(ServletConfig sc) throws ServletException {
  logger=Logger.getLogger(this.getClass());
  ServletContext sctxt=sc.getServletContext();
  logger.debug("sctxt obtained");
  try {
   Class.forName(sctxt.getInitParameter("DRIVER"));
   logger.debug("driver class loaded");
   con=DriverManager.getConnection(sctxt.getInitParameter("URL"),   sc.getInitParameter("USER"),
sc.getInitParameter("PASS"));
   logger.debug("con established");
   stmt=con.createStatement();
   logger.debug("stmt created");
   pstmt=con.prepareStatement("INSERT INTO employee VALUES (?,?,?,?,?,?,?,?)");
   logger.debug("pstmt created");
  }
  catch(Exception e) {
   e.printStackTrace();
 logger.error(e.toString());
  }// catch()
 }//init()

 // second lifecycle method
 public void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException
{

  PrintWriter out=resp.getWriter();

  // read request parameters
  String first_name=req.getParameter("first_name").trim();
  String last_name=req.getParameter("last_name").trim();
  double salary=Double.parseDouble(req.getParameter("salary").trim());
  String hire_date=req.getParameter("hire_date").trim();
  String job_id=req.getParameter("job_id").trim();
  int mgr_id=Integer.parseInt(req.getParameter("mgr_id").trim());
  int deptno=Integer.parseInt(req.getParameter("deptno").trim());
  logger.debug("Request parameter read");

  // server side validations
```

```
String err="";
DoubleValidator dv=new DoubleValidator();
if(dv.validate((salary+"")==null)
 err+="Salary is not a double value<br>";

DateValidator dav=new DateValidator();
if(dav.validate(hire_date)==null)
 err+="HireDate is invalid date<br>";

IntegerValidator iv=new IntegerValidator();
if(iv.validate(mgr_id+"")==null)
 err+="Mgr ID is not a valid int<br>";

if(iv.validate(deptno+"")==null)
 err+="Deptno is not a valid int<br>";

if(err.length()>0) {
  out.println(err);
  out.flush();
  logger.error(err);
} else {
  // DB operations
  // insert record into employee table
  logger.info("validations are successful");
  try {
    // generate PK
    int eid=0;
    rs=stmt.executeQuery("SELECT max(eid) FROM employee");
    if(rs.next()) {
      eid=rs.getInt(1);
    }// if()
    eid++;

    // record insertion
    pstmt.setInt(1, eid);
    pstmt.setString(2, first_name);
    pstmt.setString(3, last_name);
    pstmt.setDouble(4, salary);

    java.util.StringTokenizer st=new java.util.StringTokenizer(hire_date, "/");
    String date=st.nextToken();
    String month=st.nextToken();
    String year=st.nextToken();
    java.sql.Date  sdate=new  java.sql.Date((Integer.parseInt(year)-1900),  (Integer.parseInt(month)-
1), Integer.parseInt(date));
```

```
      pstmt.setDate(5, sdate);
      pstmt.setString(6, job_id);
      pstmt.setInt(7, mgr_id);
      pstmt.setInt(8, deptno);
      pstmt.executeUpdate();
      out.println("employee record inserted, eid is "+eid);
      logger.info("employee record inserted, eid is "+eid);
    }// try
    catch(Exception e) {
      e.printStackTrace();
      logger.error(e);
    }// catch()
  }// else
 }// service()

 // last lifecycle method
 public void destroy() {
   try {
    rs.close(); pstmt.close(); stmt.close(); con.close();
   }
   catch(Exception e) {
     e.printStackTrace();
   }// catch()
 }// destroy()

}// class
```

web.xml

```
<web-app>
 <context-param>
  <param-name>DRIVER</param-name>
  <param-value>oracle.jdbc.driver.OracleDriver</param-value>
 </context-param>
 <context-param>
  <param-name>URL</param-name>
  <param-value>jdbc:oracle:thin:@localhost:1521:XE</param-value>
 </context-param>

 <servlet>
  <servlet-name>nes</servlet-name>
  <servlet-class>NewEmpServlet</servlet-class>
  <init-param>
   <param-name>USER</param-name>
   <param-value>am36</param-value>
  </init-param>
```

```
 <init-param>
  <param-name>PASS</param-name>
  <param-value>am36</param-value>
 </init-param>
 <load-on-startup>1</load-on-startup>
</servlet>

<servlet-mapping>
 <servlet-name>nes</servlet-name>
 <url-pattern>/newEmpServlet</url-pattern>
</servlet-mapping>

<welcome-file-list>
 <welcome-file>index.html</welcome-file>
 <welcome-file>NewEmp.html</welcome-file>
</welcome-file-list>
</web-app>
```

log4j.properties

```
# log4j.rootLogger=debug,stdout,datefile,pattern,html,xml
log4j.rootLogger=debug,stdout,datefile,html

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.TTCCLayout

log4j.appender.datefile=org.apache.log4j.FileAppender
log4j.appender.datefile.File=MyApp.log
log4j.appender.datefile.layout=org.apache.log4j.PatternLayout
log4j.appender.datefile.layout.conversionPattern=%c %C %d %F %l %L %p %r %t - %m%n

log4j.appender.html=org.apache.log4j.FileAppender
log4j.appender.html.File=MyApp.html
log4j.appender.html.layout=org.apache.log4j.HTMLLayout
```

build.bat

```
md WEB-INF
md WEB-INF\classes
md WEB-INF\lib

javac -d .\WEB-INF\classes NewEmpServlet.java
copy logj.properties \WEB\classes
copy ojdbc14.jar \WEB\lib

jar -cvf second_form_params.war *.html *.css *.js *.png WEB-INF
```

copy second_form_params.war %tomcat%\webapps

windows dependent build .bat/.cmd extension file is. we want to platform independent build processing file. That is ANT.

build.xml

```
<project name="second_web" default="all">
 <target name="init">
  <property name="build" value="build"/>
  <property name="webinf" value="${build}\WEB-INF"/>
  <property name="classes" value="${webinf}\classes"/>
  <property name="lib" value="${webinf}\lib"/>
  <property name="oraclelib"
value="D:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar"/>
  <property name="log4jlib" value="D:\apache-log4j-1.2.17\log4j-1.2.17.jar"/>
  <property name="commonsvalidatorlib" value="E:\active\adv\am36\commons-validator-1.4.0-
bin\commons-validator-1.4.0\commons-validator-1.4.0.jar"/>
  <property name="tomcatdeploy" value="C:\Program Files\apache-tomcat-6.0.37-windows-
x86\apache-tomcat-6.0.37\webapps"/>
 </target>
 <target name="md" depends="init">
  <mkdir dir="${build}"/>
  <mkdir dir="${webinf}"/>
  <mkdir dir="${classes}"/>
  <mkdir dir="${lib}"/>
 </target>
 <target name="copy" depends="md">
  <copy file="NewEmp.html" todir="${build}"/>
  <copy todir="${build}">
   <fileset dir="." includes="*.js,*.css,*.gif,*.png"/>
  </copy>
  <copy file="web.xml" todir="${webinf}"/>
  <copy file="log4j.properties" todir="${classes}"/>
  <copy file="${oraclelib}" todir="${lib}"/>
  <copy file="${log4jlib}" todir="${lib}"/>
  <copy file="${commonsvalidatorlib}" todir="${lib}"/>
 </target>
 <target name="compile" depends="copy">
  <javac srcdir="." destdir="${classes}"  classpath="${commonsvalidatorlib}"/>
 </target>
 <target name="war" depends="compile">
  <jar basedir="${build}" destfile="second_form_params.war"/>
 </target>
 <target name="deploy" depends="war">
  <copy file="second_form_params.war" todir="${tomcatdeploy}"/>
```

```
</target>
<target name="delay" depends="deploy">
 <sleep milliseconds="1000"/>
</target>
<target name="home" depends="delay">
 <exec command="C:\Program Files\Internet Explorer\IEXPLORE.exe
http://localhost:8081/second_form_params/NewEmp.html"/>
</target>
<target name="all" depends="home">
</target>
</project>
```

In the above servlet we declared con, stmt, pstmt, rs variables as class instance variables. Class instance variables in servlet are not thread-safe because servlet runs in a singleton-multithread model. If con, stmt are declared as class instance variables, initialized in init() method and used in service(). The same con & stmt objects are used by service() method in multiple threads, if on a stmt object one resultset object is opened and is under use, in another thread process on the same stmt object if one more rs is opened, the first resultset object implicitly closes. The another problem is in the above servlet, we retrieved request parameters, validated fields, generated PK value and stored in DB using pstmt object.

In this process during first client request the service() method execution starts in one thread, the thread executed upto PK generation, the first thread allocated time is completed, so that second thread starts its execution, in the second thread also the service() method executed upto PK generation only. The problem is whichever PK value first thread has retrieved, the same value service() method in second thread also retrieves and increments for PK. Because of round-robin fashion of threads execution control comes first thread. First thread successful inserts record into DB, the first thread process destroys. Second thread execution starts, in second thread the service() method tries to insert a record with that ID/PK one record is already inserted in the DB in the previous thread operation.

Apply TRANSACTION_SERIALIZABLE isolation level to prevent this problem, but still declaring con, stmt & pstmt declared as class instance are not safe from servlet running in multiple threads.

Hence declaring such variables in service() method is recommended. But if declared service() method performance will be effected and load on DB increases.

The solution is every web server and application server in-turn supports JDBC connection pooling. Connection pool is also a one of the service runs in web and application servers. In contrast with JDBC program, connection pooling service don't perform any DB operations instead it will maintain pool of DB connections. When number of clients requests for more than its initial capacity of connections, CPMs will increase number of connection two-two more like that up-to maximum number of connections. If clients request more than DB connections capacity, such client requests are kept in queue until the existing connection comes back to pool. JDBC applications who completes their DB operations such JDBC programs releases connection back to pool.

Every web and application server vendor are having their own copy of connection pool implementation classes. Such all class implement from javax.sql.DataSource interface. java.sql is a

JDBC Core API and javax.sql is an JDBC extension API. This interface is having getConnection() method that returns Connection object. Web & App server vendors must implement their class from this interface, implements connection pool implementation on their own, but provides getConnection() method to get Connection object.

- Apache Software Foundation (ASF) - org.apache.commons.dbcp.BasicDataSource class
- Spring - org.springframework.jdbc.datasource.DriverManagerDataSource, DelegatingDataSource

Applying connection pooling in the above application:

To know how to configure connection pool in tomcat, we need to start tomcat web server first, open browser and type http://localhost:8081/

Select Tomcat Documentation

Select JDBC DataSource topic no 9

open tomcat\conf\server.xml

```
        <Context path="/second_form_params" docBase="second_form_params" unpackWars="true">
                <Resource name="jdbc/OP" auth="Container" type="javax.sql.DataSource" maxActive="10" maxIdle="2" maxWait="10000" username="am36" password="am36" driverClassName="oracle.jdbc.driver.OracleDriver" url="jdbc:oracle:thin:@localhost:1521:XE"/>
        </Context>

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

open web.xml file of our application

```
 <resource-ref>
  <res-ref-name>jdbc/OP</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
 </resource-ref>
```

open NewEmpServlet.java

```
import javax.naming.*;
import javax.sql.*;

// Connection con;
// Statement stmt;
// PreparedStatement pstmt;
// ResultSet rs;
DataSource ds;
```

```
public void init(ServletConfig sc) throws ServletException {
  try {
    /*
    */
      Context ctxt=new InitialContext();
      logger.debug("ctxt created");
      ds=(DataSource)ctxt.lookup("java:comp/env/jdbc/OP");
      logger.debug("ds obtained ");
  }
}

public void service(req,resp)...{
  try {
    Connection con=ds.getConnection();
    Statement stmt=con.createStatement();
    PreparedStatement pstmt=con.prepareStatement("INSERT INTO employee VALUES
(?,?,?,?,?,?,?,?)");
    ResultSet rs=
  }
}

 public void destroy() {
   System.out.println("NewEmpServlet destroy() method is called");
   /*
   try{
   }catch(Exception e) {}
   */
 }
}
```

Servlets API

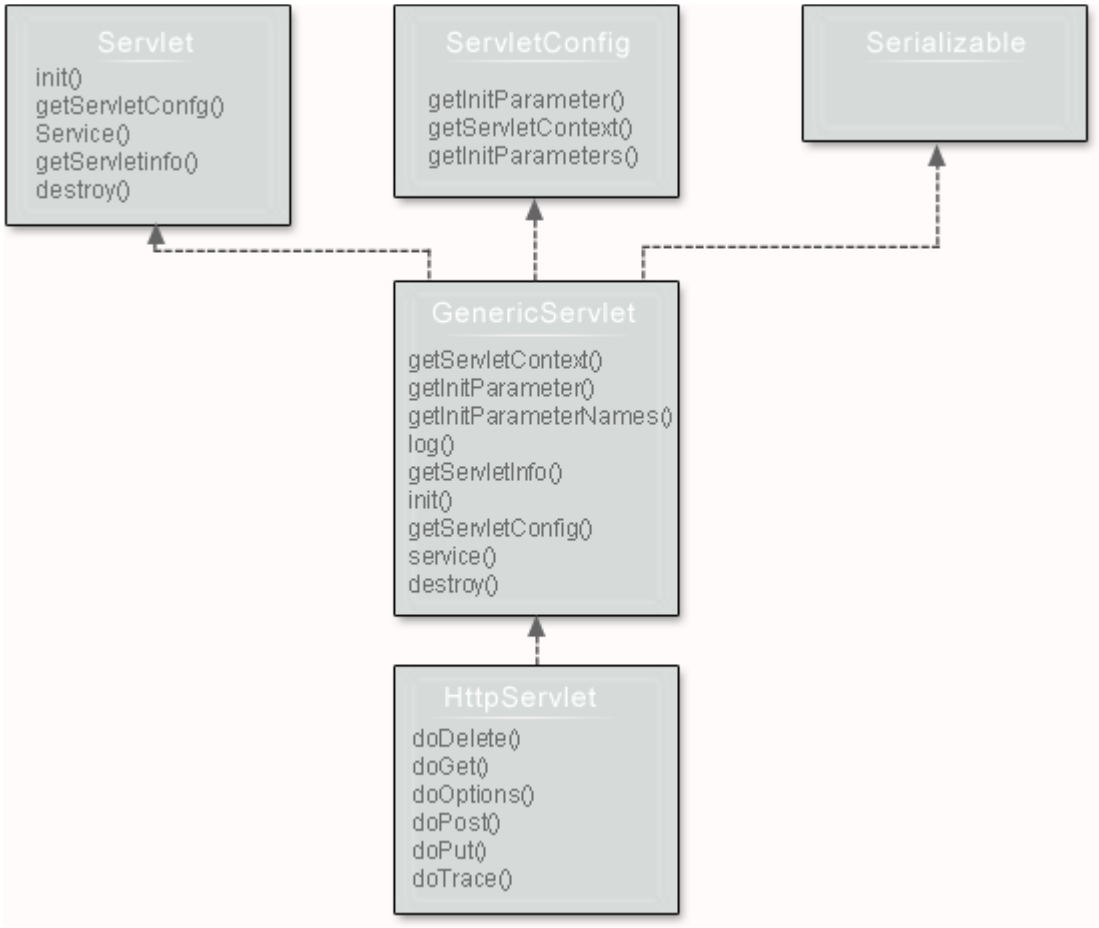
Servlet is a Java program that runs on Java Web Server running on server system, accepts request (HTTP) from the client HTML form, process it on server (validating data, DB operations, caching/client state persistence) and then responds back to client's browser.

Before Servlets, a web server (a program written in any language bounds server socket to one port number, waits for the client requests infinitely, creates one socket on server for client, assigns to one thread and runs) have only HTTP engine (I/O program that reads the client requested file using inputstream, using outputstream it writes byte-byte or line-line or buffer-buffer onto client system)

When such a HTML documents are downloaded onto client system, filled and resubmitted to web server again, a program must run on server system to read form data, perform some CRUD operations and then to respond back to client every language provides classes to run on server. Such a server side running java class is called as Servlet. In other languages ASP, PHP-PERL, ColdFusion, Ruby on Rails, Grails for Groovy, CGI for C++ and PHP, XSQL etc

These server side program extends the capabilities of a web server from processing static resources to dynamic resources.

To run such a servlet one more process runs in web server called Servlet Container. JSP container to convert JSP to servlet. Again JSP converted servlets are run by servlet container.



To develop servlets developer must implement a class from javax.servlet.Servlet interface.

```
public interface Servlet {
  // lifecycle methods
  public void init(ServletConfig sc) throws ServletException ;
  public void service(ServletRequest req, ServletResponse resp) throws ServletException, IOException;
  public void destroy();

  // miscellaneous methods
  public String getServletInfo();
  public ServletConfig getServletConfig();
}
```

Developer must implement all the abstract methods. If not developer can extend from 2 Servlet interface adapter classes

i) GenericServlet

ii) HttpServlet

In GenericServlet except service() all the methods are implemented hence service() method remained as abstract. Sub class must implement service() method.

Why developer must derive from Servlet interface directly or indirectly and must provide / override these methods?

Because servlet container is responsible for servlet class lifecycle such as instantiating, asking servlet to initialize, requesting servlet to process client request and deinitialize class instance variables before destroying servlet object.

Through servlet interface container can force developer to implement these methods or ensure to have the same methods.

class GenericServlet

GenericServlet class extended from Object implemented from Servlet, ServletConfig, Serializable interfaces. Implemented all the methods of Servlet (5), ServletConfig (3) interfaces except service() method because of that reason GS is also declared as abstract. Why service() is left as abstract because GS super class must force at-least developer to implement service() method in every servlet.

The methods of ServletConfig are:

String getInitParameter(String paramName);
Enumeration getInitParameterNames();
ServletContext getServletContext();
String getServletName();

In addition to all the above 8 methods, GenericServlet class is having one additional method called

```
public void log(String msg) {
  getServletContext().log(getServletName()+" : "+msg);
}
```

one more method is getServletName()

When log() method is used it will write the text into \tomcat\logs\localhost.2014-03-18.log

Servlets are meant to process HTML form requests on server system. But if client wants to make multiple HTML form requests to get the response back and again if form want to be submitted for request processing (multiple conversations with web server) then we need to extends servlets from HttpServlet instead of GenericSevlet.

Most of the real time web applications are all HttpServlet subclasses because client usually submits username and password to server for login validation. LoginServlet verifies username and password on server DB, if login successful the servlet sends user mailbox back to browser. From the browser

client makes another request to open the selected mail. Followed by mail content want to be deleted or resend to another email account. User is making multiple requests to server, as long as client is doing conversations, server must remember client's user credentials and past conversation state on the server system.

This process is called as client state persistence.

In order to maintain client state persistence web server/container must manage client's session management also.

Container in order to manage session, during client's first visit web container generates one 32 char length unique id (also called as sessionid/rmi UID), during first response the container sends that ID to browser via "Set-Cookie:jsessionid=32charlen". Browser will remember that id as cookie, from the second conversation onwards browser resends all the cookies back to server. In those Cookies if sessionid exist web container will recognize the client passes the same HttpSession object again and again back into servlet. Servlet will append old conversation state to new conversation state using setAttribute(name,value), getAttribute(name), removeAttribute(name) and getAttributeNames() methods. If no jsessionid is found container creates one new HttpSession object, in that it will store

- String sessionid
- boolean isNew
- long creationTime
- long lastAccessedTime
- int maxInactiveInterval
- and one empty hashtable object.

Issuing id during first visit and recognizing client request with that id from second conversation onwards on server system is called as session management.

Storing and retrieving client conversation state from HttpSession object is called as client state persistence.

Servlet container can create such a HttpSession object only to the requests coming to HttpServlet.

The reason behind extending Servlet class from HttpServlet is to obtain session management and client state persistence.

The another reason of extending from HttpServlet is only HTTP protocol 1.1 onwards, cookies concept introduced with which we can exchange server information to client and vice-versa such as jsessionid, Locale (displaying web page user interested language), themes etc.

HttpServlet is protocol HTTP protocol dependent whereas GenericServlet is protocol independent.

Third Java Web Application

Developing application on session management and client state persistence using Ecipse IDE:

Create a table in the database:

```
CREATE TABLE reg1 (regid number primary key,
                   fname varchar2(30),
                   lname varchar2(30),
```

email varchar2(30),
pass varchar2(30),
mobile varchar2(30),
address varchar2(30));

Following are the steps to create a new web application and configure tomcat in Eclipse:

Open Eclipse (any version)
Create a New DynamicWebProject
File-> New-> DynamicWebProject
Project Name: sessionman_3
Click on New Runtime
Choose Apache Tomcat v7.0-> click on Next->
Name: Apache Tomcat v6.0
Tomcat installation directory: D:\ apache-tomcat-7.0.91
JRE: jdk1.8.0_131
Click on Finish
Dynamic Web Module: 3.0
Click on Next
Click on Next
Select Generate web.xml deployment descriptor
Click on Finish

The directory structure looks as below:

sessionman_3
Java Resources
src
Libraries
Apache Tomcat v7.0
EAR Libraries
JRE System Libraries
Web App Libraries
JavaScript Resources
build
WebContent
META-INF
WEB-INF
web.xml

Create a new HTML

Right click on the Project-> New-> HTML
Name: 1.html

```
<HTML>
 <BODY>
  <PRE style="border: solid red 5px;text-align:center;text-shadow: lime; ">
```

```
 <FORM method="post" action="./mfrs">
  FName : <input type="text" name="fname">
  LName : <input type="text" name="lname">
  <input type="hidden" name="formno" value="1">
  <input type="submit" name="submit" value="Next">
 </FORM>
 </PRE>
</BODY>
</HTML>
<!--
<TABLE>
 <TR>
  <TH>FName</TH>
  <TD><input type="text" name="fname"></TD>
 </TR>
 <TR>
  <TH>LName</TH>
  <TD><input type="text" name="lname"></TD>
 </TR>
 <TR>
  <TH><input type="hidden" name="fornmno" value="1"></TH>
  <TD><input type="submit" name="submit" value="Next"></TD>
 </TR>
</TABLE>
-->
```

Create a New servlet

Right click on the Project-> New-> Servlet
Class name: MFRS
Superclass: javax.servlet.http.HttpServlet
Click on Next
Name: MFRS
URL mappings: /mfrs
Click on Next
Select doPost() method checkbox
and click on Finish

// MFRS Servlet before non-idempotency prevention
// MFRS.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```
import java.sql.Statement;
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 MFRS extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();

		int formno=Integer.parseInt(request.getParameter("formno"));
		HttpSession hs=request.getSession();

		if(formno == 1) {
			// client state persistence
			hs.setAttribute("fname", request.getParameter("fname").trim());
			hs.setAttribute("lname", request.getParameter("lname").trim());

			// second form generation
		out.println("<PRE>");
			 out.println("<FORM method='post' action='./mfrs'>");
			 out.println("Email  <INPUT type='text' name='email'>");
			 out.println("Pass   <INPUT type='password' name='pass'>");
			 out.println("<INPUT type='hidden' name='formno' value='2'>");
			 out.println("<INPUT type='submit' name='submit' value='Next'>");
			 out.println("</FORM>");
			 out.println("</PRE>");
			 out.flush();
		} else if(formno == 2) {
			// client state persistence
			hs.setAttribute("email", request.getParameter("email").trim());
			hs.setAttribute("pass", request.getParameter("pass").trim());

			// third form generation
			out.println("<PRE>");
			out.println("<FORM method='post' action='./mfrs'>");
			out.println("Mobile <INPUT type='text' name='mobile'>");
			out.println("Address<textarea name='address'></textarea>");
			out.println("<INPUT type='hidden' name='formno' value='3'>");
			out.println("<INPUT type='submit' name='submit' value='Finish'>");
			out.println("</FORM>");
			out.println("</PRE>");
```

```
                    out.flush();
                } else {
                    String fname=hs.getAttribute("fname").toString();
                    String lname=hs.getAttribute("lname").toString();
                    String email=hs.getAttribute("email").toString();
                    String pass=hs.getAttribute("pass").toString();

                    String mobile=request.getParameter("mobile").trim();
                    String address=request.getParameter("address").trim();
                    int regid=0;

                    try {
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                        Connection con=DriverManager.getConnection
                        ("jdbc:odbc:oracledsn","active","activenet");
                        Statement stmt=con.createStatement();
                        PreparedStatement pstmt=con.prepareStatement("INSERT INTO
reg1 VALUES (?,?,?,?,?,?,?)");
                        ResultSet rs=stmt.executeQuery("SELECT max(regid) FROM reg1");
                        if(rs.next()) {
                            regid=rs.getInt(1);
                        }
                        regid++;

                        pstmt.setInt(1, regid);
                        pstmt.setString(2, fname);
                        pstmt.setString(3, lname);
                        pstmt.setString(4, email);
                        pstmt.setString(5, pass);
                        pstmt.setString(6, mobile);
                        pstmt.setString(7, address);
                        pstmt.executeUpdate();
                        out.println(" Record inserted, regid is "+regid);
                    }
                    catch(Exception e) {
                        e.printStackTrace();
                    }// catch()
                }// else
            }// service()
}// class
```

<u>Deploying web application</u>

Right click on the project-> Run As-> Run on Server, click Next, and click Finish

Accesing the home page of the application:

Open browser and type ***http://localhost:8081/ sessionman_3/1.html***

Fill the 1.html file and click on Next button, you will get 2nd form, fill 2nd form and click on Next, you will get 3rd form, fill-in 3rd form and click on Finish button, the form will be stored by MFRS into DB **that shows how session management and client state persistence is taking place between client and server.**

The proble lies in the above application is after submitting 3rd form we will get registration confirmation page saying your regid is 1, press F5 again the form resubmits to servlet and the same record inserts once again with new regid.

The MFRS after non-idempotency prevention

// MFRS.java

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
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 MFRS extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();

		int formno=Integer.parseInt(request.getParameter("formno"));
		HttpSession hs=request.getSession();

		if(formno == 1) {
			// client state persistence
			hs.setAttribute("fname", request.getParameter("fname").trim());
			hs.setAttribute("lname", request.getParameter("lname").trim());

			// second form generation
			out.println("<PRE>");
			 out.println("<FORM method='post' action='./mfrs'>");
			  out.println("Email  <INPUT type='text' name='email'>");
```

```
                    out.println("Pass   <INPUT type='password' name='pass'>");
                    out.println("<INPUT type='hidden' name='formno' value='2'>");
                    out.println("<INPUT type='submit' name='submit' value='Next'>");
                   out.println("</FORM>");
                  out.println("</PRE>");
                  out.flush();
            } else if(formno == 2) {
                  // client state persistence
                  hs.setAttribute("email", request.getParameter("email").trim());
                  hs.setAttribute("pass", request.getParameter("pass").trim());

                  // third form generation
                  out.println("<PRE>");
                   out.println("<FORM method='post' action='./mfrs'>");
                    out.println("Mobile <INPUT type='text' name='mobile'>");
                    out.println("Address<textarea name='address'></textarea>");
                    out.println("<INPUT type='hidden' name='formno' value='3'>");
                    out.println("<INPUT type='submit' name='submit' value='Finish'>");
                   out.println("</FORM>");
                  out.println("</PRE>");
                  out.flush();

                  // store one token in session object like a movie ticket
                  hs.setAttribute("token", new Integer(1));
            } else {
                  if(hs.getAttribute("token")!=null) {
                        String fname=hs.getAttribute("fname").toString();
                        String lname=hs.getAttribute("lname").toString();
                        String email=hs.getAttribute("email").toString();
                        String pass=hs.getAttribute("pass").toString();

                        String mobile=request.getParameter("mobile").trim();
                        String address=request.getParameter("address").trim();
                        int regid=0;

                        try {
                              Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                              Connection con=DriverManager.getConnection
                              ("jdbc:odbc:oracledsn","active","activenet");
                              Statement stmt=con.createStatement();
                              PreparedStatement    pstmt=con.prepareStatement("INSERT
INTO reg1 VALUES (?,?,?,?,?,?,?)");
                              ResultSet  rs=stmt.executeQuery("SELECT  max(regid)  FROM
reg1");
                              if(rs.next()) {
```

```
                                        regid=rs.getInt(1);
                                }
                                regid++;

                                pstmt.setInt(1, regid);
                                pstmt.setString(2, fname);
                                pstmt.setString(3, lname);
                                pstmt.setString(4, email);
                                pstmt.setString(5, pass);
                                pstmt.setString(6, mobile);
                                pstmt.setString(7, address);
                                pstmt.executeUpdate();

                                // tear-off token once first time the form is submitted
                                hs.removeAttribute("token");
                                out.println(" Record inserted, regid is "+regid);
                        }
                        catch(Exception e) {
                                e.printStackTrace();
                        }// catch()
                }// if()
                else {
                        /* control comes to else block, if token is not present in session
                        object, that means form is resubmitted again. It is a duplicate form
                        submission. Display error */
                        out.println("<font color='red'>Your trying to submit a duplicate form
                request");
                }
        }// else
    }// service()
}// class
```

Alternative ways of session management with the help of Cookies and URL Rewriting:

In Java web applications servlet container manages session management, client state persistence is managed by web developer using HttpSession object on server system.

Servlet container by default manages session management using Cookies. If Cookies are disabled on client system, browser cannot receive Cookie:jsessionid=32charlen, browser cannot resend jsessionid to server, if session id doesn't come in http requests servlet container will treat all such clients as new client, new httpSession object is created and passed into servlet.

The consequence in the above application is during third form submission the servlet is passed with a new session object but not the session object that already contains the two previous form data from such a session object we cannot retrieve old data the registration process cannot be succeeded.

To alternatively manage session management developer must send jsessionid with id in every response to browser as a "hidden field" or through "url rewriting", then in the subsequent request web container will verify jsessionid first in the http request header, if not it will verify url or hidden field, if id is found then the same initial HttpSession is repeatedly passed into Servlet service() method.

hidden field

```
out.println("<input type='hidden' name='jsessionid' value='"+hs.getId()+"'>");
```

URL rewriting

```
out.println("<FORM method='post' action='./mfrs; jsessionid="+hs.getId()+"'>" );
```

MFRS Example after session management with Cookie & URL rewriting

```
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.Statement;
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 MFRS extends HttpServlet {

	public void service(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out = response.getWriter();

		int formno=Integer.parseInt(request.getParameter("formno"));
		HttpSession hs=request.getSession();
		System.out.println(hs.getId());

		if(formno == 1) {
			// client state persistence
			hs.setAttribute("fname", request.getParameter("fname").trim());
			hs.setAttribute("lname", request.getParameter("lname").trim());

			// second form generation
			out.println("<PRE>");
			out.println("<FORM method='post'
action='./mfrs;jsessionid="+hs.getId()+"'>");
```

```
                        out.println("<input type='hidden' name='jsessionid'
value='"+hs.getId()+"'>");
                        out.println("Email  <INPUT type='text' name='email'>");
                        out.println("Pass   <INPUT type='password' name='pass'>");
                        out.println("<INPUT type='hidden' name='formno' value='2'>");
                        out.println("<INPUT type='submit' name='submit' value='Next'>");
                       out.println("</FORM>");
                      out.println("</PRE>");
                      out.flush();
              } else if(formno == 2) {
                      // client state persistence
                      hs.setAttribute("email", request.getParameter("email").trim());
                      hs.setAttribute("pass", request.getParameter("pass").trim());

                      // third form generation
                      out.println("<PRE>");
                      out.println("<FORM method='post'
action='./mfrs;jsessionid="+hs.getId()+"'>");
                        out.println("<INPUT type='hidden' name='formno' value='3'>");
                        out.println("Mobile <INPUT type='text' name='mobile'>");
                        out.println("Address<textarea name='address'></textarea>");
                        out.println("<input type='hidden' name='jsessionid'
value='"+hs.getId()+"'>");
                        out.println("<INPUT type='submit' name='submit' value='Finish'>");
                       out.println("</FORM>");
                      out.println("</PRE>");
                      out.flush();
                      hs.setAttribute("token", new Integer(1));
              } else {
                      if(hs.getAttribute("token")!=null) {
                              String fname=hs.getAttribute("fname").toString();
                              String lname=hs.getAttribute("lname").toString();
                              String email=hs.getAttribute("email").toString();
                              String pass=hs.getAttribute("pass").toString();

                              String mobile=request.getParameter("mobile").trim();
                              String address=request.getParameter("address").trim();
                              int regid=0;

                              try {
                                      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                                      Connection con=DriverManager.getConnection
                                      ("jdbc:odbc:oracledsn","active","activenet");
                                      Statement stmt=con.createStatement();
```

```
                                        PreparedStatement pstmt=con.prepareStatement("INSERT INTO reg1 VALUES (?,?,?,?,?,?,?)");
                                        ResultSet rs=stmt.executeQuery("SELECT max(regid) FROM reg1");
                                        if(rs.next()) {
                                                regid=rs.getInt(1);
                                        }
                                        regid++;

                                        pstmt.setInt(1, regid);
                                        pstmt.setString(2, fname);
                                        pstmt.setString(3, lname);
                                        pstmt.setString(4, email);
                                        pstmt.setString(5, pass);
                                        pstmt.setString(6, mobile);
                                        pstmt.setString(7, address);
                                        pstmt.executeUpdate();
                                        hs.removeAttribute("token");
                                        out.println(" Record inserted, regid is "+regid);
                                }
                                catch(Exception e) {
                                        e.printStackTrace();
                                }// catch()
                        }// if()
                        else {
                                out.println("<font color='red'>Your trying to submit a duplicate form request");
                        }
                }// else
        }// service()
}// class
```

GET and POST difference

- GET can carry only 255 characters of form data. Post can send unlimited form data.
- In GET form data is appended at the end of URL using ? separator, hence it appears in the location bar of the browser, for client it seems unsecure. In POST the form data is encoded and sent through HTTP request body. It wont appear on the browser, it seems secure.
- New form data is posted using POST method. Already existing data on server will be retrieved using GET method.
- GET method is processed faster that POST because server reads GET content from HTTP request header, whereas POST data is read from Http request body for that server consumes some time.

Fourth Java Web Application

Client state persistence on client system using Cookies:

Cookies are client side entities to store server information such as sessionid, user interested language (Locale), user interested theme etc.

Cookies are created and send from server to client.

To write Cookie from server, the HTTP request header contains statement like this:

Set-Cookie: request header is used.

Cookie contains name, value, Comment, Domain, Max-Age, Path, Secure, Version as its attributes.

Cookies are generally created only with name and value combination.

Cookies are not permanently stored on client's browser. They are having expire time. The exipre time is set through Max-Age attribute. If value is positive integer specified in terms of seconds. Cookie will remain for specified number of seconds. If Cookie Max-Age is specified as zero, then as soon it comes to browser, cookie will be removed. If its Max-Age is specified as negative integer value the cookie remains until the browser is closed.

javax.servlet package is having one class called Cookie.

RFC 1945 - HTTP / 1.0
RFC 2616 - HTTP / 1.1
RFC 2109 - State management HTTP 1.1 protocol using Cookies

Cookie class in javax.servlet.http package creates Cookie structure object on server system Using resp.addCookie() method the cookie can be set to HTTP request header.

In HTTP response using Set-Cookie, the cookie will be set.

Cookie class constructor takes name and value as argument. It is having setter and getter methods to store additional attributes (comment, domain, Max-Age, path, secure, version).

The limitation of Cookie is its size can be upto 4KB, from each server browser can accept 20 cookies, altogether a browser can accommodate 300 cookies.

Open Eclipse and create a DynamicWebProject

Follow the previous procedure to create DynamicWebProject

Name: cookies_4

Create a HTML file

```
<!-- ShoppingCart.html -->
<HTML>
 <BODY>
  <PRE>
   <FORM method="get" action="./shoppingCartServlet">
    Prod Name  <input type="text" name="prodName">
    Qty              <input type="text" name="qty">
    <input type="submit" name="submit" value="Add cart">
    <input type="submit" name="submit" value="Update cart">
    <input type="submit" name="submit" value="Find cart">
    <input type="submit" name="submit" value="Delete cart">
```

```
   <input type="submit" name="submit" value="View All Items">
  </FORM>
 </PRE>
</BODY>
</HTML>
```

// ShoppingCartServlet.java

```
public class ShoppingCartServlet extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
    String submit=req.getParameter("submit");

          if(submit.equals("Add cart"))  {
            Cookie ck=new Cookie(req.getParameter("prodName").trim(),
req.getParameter("qty").trim());

                resp.addCookie(ck);
                resp.getWriter().println("Item added to cart");
          } else
          if(submit.equals("Update cart"))  {
            Cookie ck=new Cookie(req.getParameter("prodName").trim(),
req.getParameter("qty").trim());

                resp.addCookie(ck);
                resp.getWriter().println("cart updated");
          } else
          if(submit.equals("Find cart"))  {
            Cookie ck[]=req.getCookies();
                String prodName=req.getParameter("prodName").trim();
                for(int i=0; i<ck.length; i++) {
                  if(ck[i].getName().equals(prodName)) {
                        resp.getWriter().println(prodName+" named product requested for
"+ck[i].getValue()+" number of quantities");
                      }
                }

          } else
          if(submit.equals("Delete cart"))  {
            Cookie ck=new Cookie(req.getParameter("prodName").trim(), null);
                ck.setMaxAge(0);
                resp.addCookie(ck);
                resp.getWriter().println("Item deleted from cart");
          } else {
            Cookie ck[]=req.getCookies();
                for(int i=0; i<ck.length; i++) {
```

```
                    resp.getWriter().println(ck[i].getName()+" : "+ck[i].getValue()+"<br>");
      }// for()
           }// else
  }// doGet()
}// class
```

start tomcat
deploy web application
http://localhost:8081/ cookies_4/ShoppingCart.html

Fifth Java Web Application

Servlet-Servlet communication/Inter-servlet communication with the web application, across web application using RequestDispatcher class forward() and include() methods, across web servers using Socket programming and other one is using response.sendRedirect():

Communication between

- Servlet-> HTML,
- Servlet-> JSP,
- Servlet-> Servlet within the web application,
- across web applications in the same web server
- and across web servers.
- using response.sendRedirect() and RequestDispatcher.forward() & include()

Communication between Servlet-> HTML:

Usually client submits HTML form data to server, in response to form submission servlet will send another HTML file to client's browser. But in the last applications we generated HTML form from servlet to client using out.println() statements. Generating HTML form content in out.println() statements is complex and error prone also. Hence we can write response HTML as a separate .html extension files such files can be redirected from servlet to client's browser using response.sendRedirect(String urlOfTheFile).

+ open Eclipse
+ Create a DynamicWebProject
Name: redirect_5
+ Create a HTML
Name: Login.html

```
<PRE>
  <FORM method="POST" action="./loginServlet">
    User  <input type="text" name="user">
    Pass  <input type="password" name="pass">
          <input type="submit" name="submit" value="Login">
  </FORM>
</PRE>
```

+ Create a servlet

Name: LoginServlet
servlet-name: loginServlet
url-pattern: /loginServlet

```
public class LoginServlet extends HttpServlet {
        public void doPost(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {

                String user=request.getParameter("user").trim();
                String pass=request.getParameter("pass").trim();

                if(user.equals("abc") && pass.equals("xyz")) {
                  response.sendRedirect("./LoginSuccess.html");
                } else {
                  response.setStatus(307);
                  response.addHeader("Location",
"http://localhost:8081/fifth_redirect/LoginFailed.html");
                  // dispatch HTTP response to client
                  response.flushBuffer(); // PrintWriter out=response.getWriter(); // out.flush();
                }
        }
}
```

+ Create a HTML
Name: LoginSuccess.html
Login Success

Name: LoginFailed.html
Login Failed

Place this URL on browser:
http://localhost:8081/redirect_5/Login.html

Servlet-> JSP communication using RequestDispatcher.forward() and include()
ServletContext:
ServletContainer creates one ServletContext object per web application. ServletContext contains web DD file data (web.xml) which includes url-pattern mapped to servlet-name, servlet-name mapped to servlet-class, context parameters, servlet context attributes, objects of singleton servlets.

In Servlets old API from Servlets 1.0 - 2.0 ServletContext object contains 3 methods called

- Servlet getServlet(String servletName)
- Enumeration getServletNames()
- Enumeration getServlets()

If one servlet1 requests sctxt.getServlet("s2"), the context will return s2 object. With the s2 object we can access s2 completely means we can access its class instance variables, its instance methods, its lifecycle methods including init(), service() and destroy(). But ServletContext object is giving privilage to Servlet1 only to communicate with Servlet2 service() method to share their business logic execution but not to communicate with other methods of Servlet2 which will be treated as misusing servlet privilage. Due to security concern these 3 methods are deprecated in Servlets 2.1. Until Servlet 2.3 no alternative API is given to communicate between servlets. In Servlets 2.3 RequestDispatcher interface is introduced as an alternative.

To create RequestDispatcher object two methods given in ServletContext called:

- RequestDispatcher getRequestDispatcher(String urlPattern)
- RequestDispatcher getNamedDispatcher(String servletName)

To communicate with other servlets we can use any one of the method because every servlet is having both servlet-name and url-pattern configured in web.xml but if servlet wants to communicate with JSP because JSPs are not configured in web.xml file we must use only getRequestDispatcher() method but not getNamedDispatcher().

****

Suppose if we are having Login.jsp that can be configured in web.xml file as follows:

```
<servlet>
 <servlet-name>login</servlet-name>
 <jsp-file>Login.jsp</jsp-file>
</servlet>
```

If JSP is configured in web.xml file then we can obtain its RequestDispatcher object using getNamedDispatcher() also.

****

What is RequestDispatcher?

RequestDispatcher is an interface, its implementation is provided by web server vendor. During sctxt.getRequestDispatcher()/getNamedDispatcher() the context object obtains servlet object, passes into RequestDispatcher object and that object reference is returned to client/servlet1.

When client communicates with the methods of RequestDispatcher, RD communicates with the service() method of other servlet.

RequestDispatcher contains two methods called forward()) and include(). The difference is in case if servlets are communicated using include () then all the servlet out.println() statements are included in HTTP response body. If any servlet communicates with other servlet using forward(), then the preceding servlet output written into HTTP response body will be cleared and the next servlet out.println() statement output will be included in resp body. As soon as second servlet service() method execution is completed control will come to forward() method of RD, that will flush() the response, the resultant is HTTP response is flushed/committed to client.

Sixth Java Web Application

Example on RequestDispatcher

+ open Eclipse

+ Create a DynamicWebProject

Name: disp_6

+ Create 3 Servlet Named Servlet1, Servlet2, Servlet3, their names and url-patterns are s1, s2, s3, /s1, /s2, /s3

Right click on the web project-> New-> Servlet

Servlet class name: Servlet1

Servlet Name: s1

Url Pattern: /s1

// Servlet1.java

```
@WebServlet(name = "s1", urlPatterns = { "/s1" })
public class Servlet1 extends HttpServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		out.println("msg from s1 before");

			// communicating with s2
			ServletContext sctxt=getServletContext();
			RequestDispatcher disp=sctxt.getNamedDispatcher("s2");
			disp.include(request, response);

		out.println("msg from s1 after");
	}
}
```

// Servlet2.java

```
@WebServlet(name = "s2", urlPatterns = { "/s2" })
public class Servlet2 extends HttpServlet {
	public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
		response.setContentType("text/html");
		PrintWriter out=response.getWriter();
		out.println("msg from s2 before");

			// communicating with s2
			ServletContext sctxt=getServletContext();
			RequestDispatcher disp=sctxt.getRequestDispatcher("/s3");
			disp.include(request, response);

		out.println("msg from s2 after");
	}
}
```

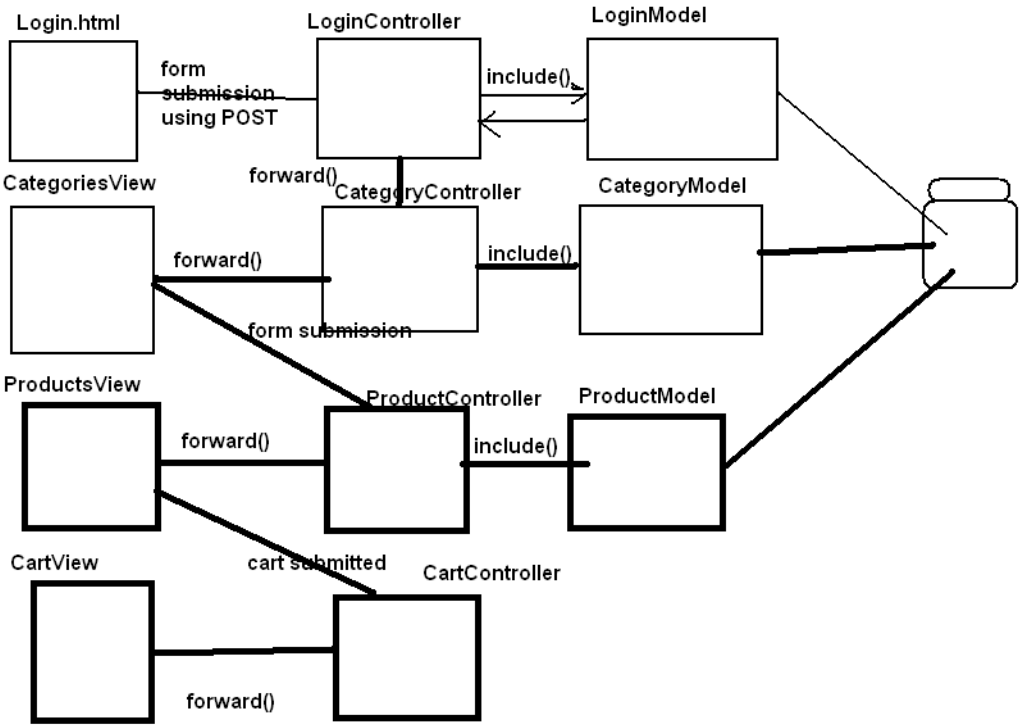
// Servlet3.java

```
@WebServlet(name = "s3", urlPatterns = { "/s3" })
public class Servlet3 extends HttpServlet {
        public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                response.setContentType("text/html");
                PrintWriter out=response.getWriter();
                out.println("msg from s3 ");
        }
}
```

Deploy web application
Right click on the project-> RunAs-> Run on Server
http://localhost:65535/disp_6/s1

Seventh Java Web Application

The following example covers how to implement MVC architecture in which communication exist between servlet using RequestDispatcher.forward()/include() methods:
ShoppingCart application using RequestDispatcher.forward() and include() methods:



Create tables in database

```
create table users (username varchar2(20) primary key,
                password varchar2(20));

create table categories (cid number primary key,
                        cname varchar2(30));
```

```
create table products (pid number primary key,
                       pname varchar2(30),
                       price number,
                       pdescr varchar2(50),
                       cid number references categories(cid));

insert into users values ('abc', 'xyz');
insert into users values ('xyz', 'abc');
insert into users values ('lmn', 'pqr');
insert into users values ('pqr', 'lmn');

insert into categories values (1, 'Books');
insert into categories values (2, 'Toys');
insert into categories values (3, 'Sports ware');
insert into categories values (4, 'Foodie');

insert into products values (1, 'Saga of Melusha', 500.00, 'Amish Tripati', 1);
insert into products values (2, 'Two States', 450.00, 'Chetan Bhagat', 1);
insert into products values (3, 'The promise', 450.00, 'Nikita Singh', 1);

insert into products values (4, 'Barbie Doll', 1050.00, 'Barbie', 2);
insert into products values (5, 'Hot wheels', 150.00, 'Hot wheels', 2);
insert into products values (6, 'Logo Bricks', 550.00, 'Logo Bricks', 2);

insert into products values (7, 'Nike Shoes', 2500.00, 'Nike', 3);
insert into products values (8, 'MRF Cricket Bat', 1500.00, 'Bat', 3);
insert into products values (9, 'Nike Tennis Rocket', 2550.00, 'Rocket', 3);

insert into products values (10, 'Schezwan Noodles', 120.00, 'Schezwan', 4);
insert into products values (11, 'Rasagulla', 350.00, 'Resagulla', 4);
insert into products values (12, 'Hyderabad Biryani', 250.00, 'Biryani', 4);

commit;
```

Create a DynamicWeb Project

Name: cart_7

src

- **controller**
  - LoginController
  - CategoryController
  - ProductController
  - CartController
- **model**
  - LoginModel

CategoryModel
ProductModel

view

CategoriesView
ProductsView
CartView

beans

Category
Product

WebRoot

WEB-INF

web.xml

Login.html
LoginFailed.html

Login.html

```
<PRE>
 <FORM method="post" action="./lc">
  User   <input type="text" name="user">
  Pass   <input type="password" name="pass">
  <input type="submit" name="submit" value="Login">
 </FORM>
</PRE>
```

Create a Servlet
Package: controller
Name: LoginController
lc, /lc
Press CTRL+SHIFT+O to automatically import packages

```
public class LoginController extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
				throws ServletException, IOException {

		String user=request.getParameter("user").trim();
		String pass=request.getParameter("pass").trim();

		if(user.isEmpty() && pass.isEmpty()) {
		  response.sendRedirect("./LoginFailed.html");
		} else {
		  getServletContext().getNamedDispatcher("lm").include(request,response);
		  String loginStatus=request.getAttribute("loginStatus").toString();
		  if(loginStatus.equalsIgnoreCase("Success")) {
		    getServletContext().getRequestDispatcher("/cc").forward(request,response);
```

```
                    } else {
                      response.sendRedirect("./LoginFailed.html");
                    } // else
                  }// else
          }// service()
}// class
```

Create a Servlet
Package: model
Name: LoginModel
lm, /lm
Press CTRL+SHIFT+O to automatically import packages

```
public class LoginModel extends HttpServlet {

          public void service(HttpServletRequest request, HttpServletResponse response)
                              throws ServletException, IOException {

                    String user=request.getParameter("user").trim();
                    String pass=request.getParameter("pass").trim();

      try {
                     Connection con=factory.DBConn.getConn();
                     Statement stmt=con.createStatement();
                     ResultSet rs=stmt.executeQuery("SELECT * FROM users WHERE
username='"+user+"' AND password='"+pass+"' " );
                     if(rs.next()) {
                      // username is stored in session scope
                      request.getSession().setAttribute("user", user);

                              // user loginstatus in request scope
                      request.setAttribute("loginStatus", "Success");
                     } else {
                      request.setAttribute("loginStatus", "Failed");
                     }
                    }
                    catch(Exception e) {}
          }
}
```

Create a Class
Package: beans
Name: Category
Press CTRL+SHIFT+O to automatically import packages

```
public class Category {
   private int cid;
```

```
	private String cname;
	List productList;
}
```

Create a Class
Package: beans
Name: Product
Press CTRL+SHIFT+O to automatically import packages

```
public class Product {
   private int pid;
	private String pname;
	private double price;
	private String pdescr;
	private Category category;
}
```

Create a Servlet
Package: controller
Name: CategoryController
cc, /cc
Press CTRL+SHIFT+O to automatically import packages

```
public class CategoryController extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
				throws ServletException, IOException {
				/*
				verify categoriesList exist in the sctxt or not; if not request CategoryModel; if
exist forward CategoriesView to client
				*/
				ServletContext sctxt=getServletContext();
				if(sctxt.getAttribute("categoriesList")==null) {
				   sctxt.getRequestDispatcher("/cm").include(request, response);
					sctxt.getRequestDispatcher("/cv").forward(request, response);

				} else {
				   sctxt.getRequestDispatcher("/cv").forward(request, response);
				}
	}
}
```

Create a Servlet
Package: model
Name: CategoryModel
cm, /cm
Press CTRL+SHIFT+O to automatically import packages

```
public class CategoryModel extends HttpServlet {
```

```
	public void service(HttpServletRequest request, HttpServletResponse response)
				throws ServletException, IOException {
				/*
				retrieve categories, populate each record into one Category bean, store all the bean objects into categoriesList and then store the list into sctxt
				*/
				ServletContext sctxt=getServletContext();
				try {
				List<Category> list=new ArrayList();
				Connection con=factory.DBConn.getConn();
				Statement stmt=con.createStatement();
				ResultSet rs=stmt.executeQuery("SELECT * FROM categories");
				while(rs.next()) {
				  Category c=new Category(rs.getInt(1), rs.getString(2), null);
				  list.add(c);
				}// while
				sctxt.setAttribute("categoriesList", list);
				}// try
				catch(Exception e) {
				 e.printStackTrace();
				}// catch
	}
}
```

Create a Servlet
Package: view
Name: CategoriesView
cv, /cv
Press CTRL+SHIFT+O to automatically import packages

```
public class CategoriesView extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
				throws ServletException, IOException {
				ServletContext sctxt=getServletContext();
				Object o=sctxt.getAttribute("categoriesList");
				List<Category> list=(List)o;
				PrintWriter out=response.getWriter();
				out.println("<UL>");
				for(Category c:list) {
				  out.println("<LI><A href='./pc?catid="+c.getCid()+"'>"+c.getCname()+"</A></LI>");
				}
				out.println("</UL>");
	}
}
```

http://localhost:8081/cart_6/Login.html

Create a Servlet
Package: controller
Name: ProductController
pc, /pc
Press CTRL+SHIFT+O to automatically import packages

```
public class ProductController extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

			/*
			  retrieve catid from req
			  verify with the catid attribute exist in sctxt or not
			  if exist forward ProductsView to client
			  if not request ProductModel using disp.incude() and then forward ProductsView to client
			*/
			ServletContext sctxt=getServletContext();

			String catid=request.getParameter("catid");
			if(sctxt.getAttribute(catid)==null) {
			  sctxt.getRequestDispatcher("/pm").include(request, response);
			  sctxt.getRequestDispatcher("/pv").forward(request, response);
			} else {
			  sctxt.getRequestDispatcher("/pv").forward(request, response);

			}
	}
}
```

Create a Servlet
Package: model
Name: ProductModel
pm, /pm
Press CTRL+SHIFT+O to automatically import packages

```
public class ProductModel extends HttpServlet {
	public void service(HttpServletRequest request, HttpServletResponse response)
			throws ServletException, IOException {

			/*
			  retrieve catid from req
			  with catid retrieve products from product table, store records into Product object, store all such objects into List, store List into sctxt object with catid
			*/
```

```
                        ServletContext sctxt=getServletContext();

                        String catid=request.getParameter("catid");
                        try {
                          Connection con=factory.DBConn.getConn();
                          Statement stmt=con.createStatement();
                          ResultSet rs=stmt.executeQuery("SELECT * FROM products WHERE
cid="+catid);
                          List<Product> list=new ArrayList<Product>();
                          while(rs.next()) {
                             Product p=new Product(rs.getInt(1), rs.getString(2), rs.getDouble(3),
rs.getString(4), null);
                                    list.add(p);

                          }// while()
                           sctxt.setAttribute(catid, list);
                        }// try
                        catch(Exception e) {
                         e.printStackTrace();
                        }// catch()
        }// service()
}// class
```

Create a Servlet
Package: view
Name: ProductsView
pv, /pv
Press CTRL+SHIFT+O to automatically import packages

```
public class ProductsView extends HttpServlet {
        public void service(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                        ServletContext sctxt=getServletContext();
                        String catid=request.getParameter("catid");
                        List<Product> list=(List)sctxt.getAttribute(catid);
                        out.println("<FORM method='post' action='./cartc'>");
                        out.println("<TABLE>");
                         out.println("<TR>");
                          out.println("<TH>PID</TH>");
                          out.println("<TH>PName</TH>");
                          out.println("<TH>Price</TH>");
                          out.println("<TH>Descr</TH>");
                          out.println("<TH>Qty</TH>");
                         out.println("</TR>");
                         for(Product p:list) {
```

```
                        out.println("<TR>");
                                out.println("<TD>"+p.getPid()+"</TD>");
                                out.println("<TD>"+p.getPname()+"</TD>");
                                out.println("<TD>"+p.getPrice()+"</TD>");
                                out.println("<TD>"+p.getPdescr()+"</TD>");
                                out.println("<TD><input type='text' name='"+p.getPid()+"'
value=''></TD>");
                        out.println("</TR>");
                       }
                       out.println("<TR>");
                       out.println("<TD colspan='2'>");
                       out.println("<input type='submit' name='submit' value='Add To Cart'>");
                       out.println("</TD>");
                       out.println("<TD colspan='2'>");
                       out.println("<A href='./cv'>Continue</A>");
                       out.println("</TD>");
                      out.println("</TABLE>");
                      out.println("</FORM>");
        }
}
```

Create a Servlet
Package: model
Name: CartController
cartc, /cartc
Press CTRL+SHIFT+O to automatically import packages

```
public class CartController extends HttpServlet {
        public void service(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                ServletContext sctxt=getServletContext();
                HttpSession hs=request.getSession();

                Map map;
                if(hs.getAttribute("shoppingCartMap")==null) {
                        map=new HashMap();
                        hs.setAttribute("shoppingCartMap", map);
                } else {
                        map=(Map)hs.getAttribute("shoppingCartMap");
                }

                Enumeration en=request.getParameterNames();
                while(en.hasMoreElements()) {
                  String name=en.nextElement().toString();
                  if(name.equals("submit"))  {
```

```
                } else {
                   String qty=request.getParameter(name).trim();
                          // client state persistence; store pid and qty as key & value pairs
                          if(!qty.isEmpty()) {
                            // hs.setAttribute(name, qty);
                           map.put(name, qty);
                          }// if()
                  }// else
                }// while()

                sctxt.getRequestDispatcher("/cartView").forward(request, response);
        }
}
```

Create a Servlet
Package: model
Name: CartView
cartv, /cartv
Press CTRL+SHIFT+O to automatically import packages

```
public class CartView extends HttpServlet {
        public void service(HttpServletRequest request, HttpServletResponse response)
                        throws ServletException, IOException {
                PrintWriter out=response.getWriter();
                ServletContext sctxt=getServletContext();
                HttpSession hs=request.getSession();

                Map shoppingCartMap =(Map)hs.getAttribute("shoppingCartMap");
                Set<String> pid=map.keySet();
                Iterator iter=pid.iterator();
                while(iter.hasMoreElements()) {
                   String name=iter.nextElement().toString();
                   String value= shoppingCartMap.get (name);
                   out.println(name+" "+value+"<br>");
                }
        }
}
```

Eight Java Web Application

Filters

Filters are "Chain of Reposibility" design pattern implemented classes.

Filters are also a web components as same as Servlets.

Servlets are written per HTML form basis. The logic written in servlet is to process HTML form data. Whereas the logic written in Filter is not to process HTML form data instead the logic which is common to execute in more than one servlet is written in filter.

The logic which is commonly required in more than one servlet are:
i) Logging
ii) Field Validations
iii) Authentication & Authorization
iv) File-upload processing
v) Image processing
vi) Encryption & Decryption of data
vii) Compression
viii) Detection harmful data submitted by hackers
ix) Page Decoration Filter

To implement Filter developer must implement one class that implements from javax.servlet.Filter interface. Filter interface is having 3 methods called
public void init(FilterConfig fc)
public void doFilter(ServletRequest req, ServletResponse resp, FilterChain fc)
public void destroy()

Filters are intended to execute both before and after servlet execution.

To execute Filter before servlet execution, servlet url-pattern must be applied to filter url-pattern.

To dispatch control from filter to next filter or servlet, in doFilter() method of Filter we need to call fc.doFilter(req, resp)

Code which is written in Filter before fc.doFilter(req, resp) will execute before servlet execution (acts like a pre-filter), code written after fc.doFilter(req, resp) will execute after servlet execution (acts like post-filter).

Open Eclipse
Create a DynamicWeb Project
Name: filters_8
src
- LoginServlet
- RegServlet
- **filters**
  - LoggingFilter
  - PageDecorationFilter
  - ValidationFilter

WebRoot
- **WEB-INF**
  - web.xml

Login.html
Reg.html

<!-- Login.html -- >

```
<pre>
 <form method="post" action="./ls">
  User  <input type="text" name="user">
  Pass  <input type="password" name="pass">
  <input type="submit" name="submit" value="Login">
 </form>
</pre>
```

<!-- Reg.html -->

```
 <pre>
 <form method="post" action="./rs">
  SName  <input type="text" name="sname">
  Email    <input type="text" name="email">
  Mobile  <input type="text" name="mobile">
  Course  <input type="text" name="course">
  Address <input type="text" name="address">
  <input type="submit" name="submit" value="Register">
 </form>
 </pre>
```

// LoginServlet.java

```
public class LoginServlet extends HttpServlet {
  public void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
     resp.getWriter().println("Login Successful");
  }
}
```

// RegServlet.java

```
public class RegServlet extends HttpServlet {
  public void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
     resp.getWriter().println("Registration Successful");
  }
}
```

// LoggingFilter.java

```
public class LoggingFilter implements Filter {
   @Override
   public void init(FilterConfig arg0) throws ServletException {
         // TODO Auto-generated method stub
```

```
   }

   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain fc) throws
ServletException, IOException {

            System.out.println("LoggingFilter executed before
"+((HttpServletRequest)req).getServletPath());

                  fc.doFilter(req, resp);

            System.out.println("LoggingFilter executed after
"+((HttpServletRequest)req).getServletPath());

          }
   public void destroy()  { }
}
```

// PageDecorationFilter.java

```
public class PageDecorationFilter implements Filter {
   public void init(FilterConfig fc) throws ServletException { }
   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain fc) throws
ServletException, IOException {

    PrintWriter out=resp.getWriter();
    out.println("<H1><CENTER>ActiveNET Inc IN, US, UK, UAE, Africa, Oceana</CENTER></H1>");

    out.println("<CENTER>Home || Training || Development || Projects || Contact Us</CENTER>");

out.println("<UL><LI>Home</LI><LI>Training</LI><LI>Development</LI><LI>Projects</LI><LI>Contac
t Us</LI>");

            fc.doFilter(req, resp);

            out.println("New Batch Staring<br/>Openings on java<br/>Immediate Openings<br/>");

    out.println("<H5><CENTER>Copyrights reserved to ActiveNET Inc 2014-3014</CENTER></H5>");
          }
   public void destroy()  { }
}
```

// ValidationFilter.java

```
public class ValidationFilter implements Filter {
   public void init(FilterConfig fc) throws ServletException { }
```

```
   public void doFilter(ServletRequest req, ServletResponse resp, FilterChain fc) throws
ServletException, IOException {
    resp.setContentType("text/html");
    PrintWriter out=resp.getWriter();
            String err="";
            Enumeration en=req.getParameterNames();
            while(en.hasMoreElements()) {
              String name=en.nextElement().toString();
                    String value=req.getParameter(name);

                    if(value.isEmpty()){
                      err=err+"<br/> "+name+" is empty<br>";
                    }// if()
            }// while()

            if(err.length()>0) {
              out.println("<font color='red'>"+err+"</font>");
    } else {
               fc.doFilter(req, resp);
            }
           }
   public void destroy()  { }
}
```

web.xml

```
<web-app>
 <filter>
  <filter-name>lf</filter-name>
  <filter-class>filters.LoggingFilter</filter-class>
 </filter>
 <filter>
  <filter-name>pdf</filter-name>
  <filter-class>filters.PageDecorationFilter</filter-class>
 </filter>
 <filter>
  <filter-name>vf</filter-name>
  <filter-class>filters.ValidationFilter</filter-class>
 </filter>

 <filter-mapping>
  <filter-name>lf</filter-name>
  <url-pattern>/ls</url-pattern>
 </filter-mapping>
 <filter-mapping>
  <filter-name>lf</filter-name>
```

```
  <url-pattern>/rs</url-pattern>
 </filter-mapping>

 <filter-mapping>
  <filter-name>pdf</filter-name>
  <url-pattern>/ls</url-pattern>
 </filter-mapping>
 <filter-mapping>
  <filter-name>pdf</filter-name>
  <url-pattern>/rs</url-pattern>
 </filter-mapping>

  <filter-mapping>
  <filter-name>vf</filter-name>
  <url-pattern>/ls</url-pattern>
 </filter-mapping>
 <filter-mapping>
  <filter-name>vf</filter-name>
  <url-pattern>/rs</url-pattern>
 </filter-mapping>

 <servlet>
  <servlet-name>ls</servlet-name>
  <servlet-class>LoginServlet</servlet-class>
 </servlet>
 <servlet>
  <servlet-name>rs</servlet-name>
  <servlet-class>RegServlet</servlet-class>
 </servlet>

 <servlet-mapping>
  <servlet-name>ls</servlet-name>
  <url-pattern>/ls</url-pattern>
 </servlet-mapping>
 <servlet-mapping>
  <servlet-name>rs</servlet-name>
  <url-pattern>/rs</url-pattern>
 </servlet-mapping>

</web-app>
```

http://localhost:8081/filters_8/Login.html

Ninth Java Web Application

Listeners:

Listeners are event handlers. Class which handles some logic during events is a EventHandler/Listener. onClick, onChange, onSelect etc are UI events. But Listeners in servlets handles server side events.

What are server side events?

ServletContainer creating & destroying ServletContext, Servlet, ServletRequest and HttpSession objects are all events.

Have you received any event notification for the above said objects before?

If you say No! I says Yes, because when servlet container create and destroy servlet object it is invoking init() and destroy() methods on servlet instance, we feel invoking init() and destroy() methods is called as calling servlet lifecycle methods but they are servlet event listener methods.

But no events are listened yet on ServletRequest, HttpSession and ServletContext objects.

In javax.servlet package

```
i)  public interface ServletContextListener {
      public void contextInitialized(ServletContextEvent sce);
      public void contextDestroyed(ServletContextEvent sce);
    }

ii) public interface ServletRequestListener {
      public void requestInitialized(ServletRequestEvent sre);
      public void requestDestroyed(ServletRequestEvent sre);
    }
```

In javax.servlet.http package

```
iii)public interface HttpSessionListener {
       public void sessionCreated(HttpSessionEvent hse);
       public void sessionDestroyed(HttpSessionEvent hse);
            }
```

To receive these events developer must write a sub class of these interfaces, implement all the 6 methods and the class must be configured in web.xml file under

```
<listener>
 <listener-class>listeners.MyListener</listener-class>
</listener>
```

The purpose of ServletContextListener, contextInitialized() method is if we want to initialize anything at the time of web application deployment such as Connection object/DataSource object creation,loading hibernate.cfg.xml file and and store SessionFactory object into ServletContext such things will be performed in ServletContextListener.

Example on Listeners

+ Open Eclipse

+ Create a DynamicWebProject

Name: listeners_9

+ Create a Class

Package: listeners

Name: MyListener

super interfaces:ServletRequestListener,HttpSessionListener, ServletContextListener

```
public class MyListener implements ServletRequestListener, HttpSessionListener,
ServletContextListener {

    public void requestInitialized(ServletRequestEvent sre) {
            System.out.println("request initialized");
        }
    public void requestDestroyed(ServletRequestEvent sre) {
            System.out.println("request destroyed");
        }

     public void sessionCreated(HttpSessionEvent hse) {
            System.out.println("session created");
                }
     public void sessionDestroyed(HttpSessionEvent hse) {
            System.out.println("session destroyed");
                }

    public void contextInitialized(ServletContextEvent sce) {
            System.out.println("context initialized");
        }
    public void contextDestroyed(ServletContextEvent sce) {
            System.out.println("context destroyed");
        }
}
```

+ Create one servlet

Name: ListenerServlet

ls, /ls

```
public class ListenerServlet extends HttpServlet {
   public void service(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
            HttpSession hs=req.getSession();
            PrintWriter out=resp.getWriter();
            out.println("msg from ListenerServlet");
        }
```

}

web.xml

```
<web-app>
 <listener>
  <listener-class>listeners.MyListener</listener-class>
 </listener>
 <servlet>
   <servlet-name>ls</servlet-name>
   <servlet-class>ListenerServlet</servlet-class>
 </servlet>
 <servlet-mapping>
   <servlet-name>ls</servlet-name>
   <url-pattern>/ls</url-pattern>
 </servlet-mapping>
 <session-config>
  <session-timeout>1</session-timeout>
 </session-config>
</web-app>
```

http://localhost:8081/listeners_9/ls

****************************The End of Servlets*************************