CGI = common gateway interface

M S N V Dar

SOWMIKA D

IT - B

Introduction To Servlets

In the early days of Internet, web sites were comprised of only static HTML documents stored in the server's file system. The communication was only one way. There was no way for the client to transmit information to the server so as to customize the information returned from the server or store data in a database. Web pages were only informative but not interactive. Introduction of CGI gave life to the static World Wide Web.

In combination with the HTML forms, CGI provided a mechanism for true two-way communication between the web client and the web browser. This new technology paved the way for interactive web applications like online shopping, online customer support, online banking. As industry embraced WWW as the place of global business, CGI was showing signs of aging in terms of performance and flexibility. Emergence of new technologies became inevitable to replace CGI as the standard interface for building dynamic web sites.

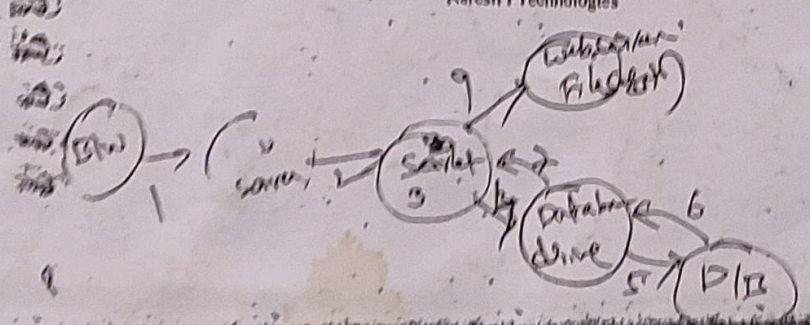
What is a Servlet?

A Servlet is a replacement of a CGI program in the context of a dynamic website. A Servlet is a server side web component, written in Java, that extends the functionality of a web server.

How are Servlets superior to CGI?

A CGI program dynamically builds and return an HTML page based on the kind of client request. It processes user input passed by the HTML form and stores it into the database. This is what is required in online applications. Servlets, as replacement of CGI programs, provide the same functionality with increased performance, portability and security.

For each client request, CGI based server creates a new process and after response is delivered the process is destroyed. Since a new process must be created for each request, database-driven pages require a new database connection with every request. It incurs a high performance penalty. In contrast, a Servlet shares a single database request across multiple client requests. In case of Servlets, for attending multiple client requests only multiple threads are created and process will be only one. Unlike CGI applications, Servlets can be run on different servers and platforms without modifications. Servlets being written in Java offer much better security than that of CGI applications.



11:30 to 12:30
12:30 to 1:30

SERVLET BASICS

Even though a Servlet is a java program, it is special in many areas. We cannot write a Servlet class as we wish. We should write it according to SERVLETS specification. A Servlet class contains methods. But they are special methods. Only designated methods we should define. We don't call them. Server calls them at appropriate time. We don't create the instance of the Servlet class. Server creates one. As a Servlet programmer our duty is to know what are the special methods we need to define in a Servlet and in what method what kind of application logic to write.

Skeleton structure of a Servlet class

```
import javax.servlet.*; import javax.servlet.http.*; import java.io.*;
public class OurServlet extends GenericServlet
{
  public void init(ServletConfig config) throws ServletException
  {
  }
  public void service(ServletRequest req,ServletResponse res) throws
                  ServletException, IOException
  {
  }
  public void destroy()
  {
  }
}
```

GenericServlet
→ init
→ service
→ destroy

The above three methods are known as life cycle methods. Server calls them at appropriate time in the life of the Servlet. While doing Servlet programming, we generally write Servlet initialization information retrieval code in the init(). We can provide database connection also to the Servlet in this method. Server calls this method only once in the lifetime of a Servlet. The heart of the dynamic functionality of the Servlet is its service method. In this method we write the complete application logic. Server calls this method whenever the client makes a request. Capturing html form data, communicating with the database, generating the dynamic web content and sending the output to the client is done in service method. Resources clean up code is written in the destroy method. This method is called just before the Servlet instance is garbage collected.

Servlet object is created by container (server)

Servlet Life cycle

Servlets are container managed. The life and death of Servlets are under the control of the Servlet engine. Servlet engine is implemented according to SERVLET specification and the Servlet also is developed according to the same specification. That is the reason why container is able to manage the Servlet life cycle. The Servlet life cycle steps describe the process by which the server (container/engine) invokes the Servlet.

Step 1: - The Servlet container (simply server) loads the Servlet class that we developed and deployed into it. This can happen either at the time of first client request or at the time of application startup.

Step 2: - The Servlet engine (container) creates the instance of the just loaded Servlet class.

Step 3: - The Servlet engine creates the instance of some child class of ServletConfig interface. This child class implementation changes from container to container. This instance contains initialization information and application/container context information.

Step 4: - Servlet container calls the init method on the Servlet instance. While calling, it supplies the reference of ServletConfig type to this method. This reference refers to the object created in step three. We can use this reference to retrieve the initialization and context information.

Step 5: - Servlet engine creates two instances basing upon the client request. These two instances are container implemented some child classes of ServletRequest interface and ServletResponse interface respectively.

Step 6: - Servlet container calls the service method on the Servlet instance. While calling this method, container supplies the references of type ServletRequest and ServletResponse to this method. These references refer to the objects constructed in step five. By using these two references, client request processing code can be implemented.

Step 7: - If the server receives another client request, process starts from Step 5 again.

Step 8: - When the container is instructed to keep the Servlet out of service (un-deployment), container calls the destroy method just before the Servlet instance is garbage collected.

DEVELOPING THE FIRST WEB APPLICATION

As we know, a Servlet is a server side program. Its basic functionality is to provide dynamism for the web pages. In real time website development, there involves not only servlets but also pre-created html pages, images etc. Therefore, to bundle these resources in a structured manner, SERVLETS specification mandates a hierarchy of directories to place the web resources at appropriate location.

According to the specification, every web application (interactive online application) should have a root directory. Under that directory we directly place our html files, images and jsps. Root directory should have a sub directory called WEB-INF. Within WEB-INF directory we should have a file by name *web.xml* and two directories by name classes and *lib*. Our Servlet class files are to be kept in the classes directory. Any Java archive files are to be placed in the lib directory. Web.xml is known as the deployment descriptor. Primarily we register the servlets with the application through this file. Container knows about the details of the servlets and their configuration and deployment information through this file only.

Example web application: - In this application, we have an html page that acts as login page for the website. Our first Servlet receives user name and password from the form and authenticates the user. The application directory structure is as follows.

```
Loginapp
    login.html
    WEB-INF
        web.xml
        classes
            AuthenticationServlet.class
        lib
            Classes12.jar
```

In the above hierarchy, loginapp is the root directory. Login.html is the login web page for the application. Servlet class is AuthenticationServlet. For the database connection we are using oracle thin driver. Therefore classes12.jar is required.

Source code of login.html

```
<HTML>
<BODY BGCOLOR=cyan>
  <CENTER>
    <H2> LOG IN TO OUR WEBSITE </H2>
    <FORM ACTION="http://localhost:8080/loginapp/authenticate">
        USER NAME<INPUT TYPE="text" NAME="user"> <BR>
        PASSWORD<INPUT TYPE="password" NAME="pwd"> <BR>
        <INPUT TYPE="submit" VALUE=" click here to login" >
    </FORM>
  </CENTER>
</BODY>
</HTML>
```

Source code of web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>login</servlet-name>
    <servlet-class>AuthenticationServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>login</servlet-name>
    <url-pattern>/authenticate</url-pattern>
  </servlet-mapping>
</web-app>
```

if (name.equals("admin")) & & (pwd.equals("admin"))

{ res.sendRedirect("Valid servlet");

}

else

{ res.sendRed("Invalid Ser

Source code of AuthenticationServlet.java

```
import java.io.*;import java.sql.*;import javax.servlet.*;
public class AuthenticationServlet extends GenericServlet
{
    Connection con;
    public void init(ServletConfig config)throws ServletException
    {
        try
        {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            String url="jdbc:oracle:thin:@localhost:1521:ord";
            con=DriverManager.getConnection(url,"scott","tiger");
        }
        catch(Exception e)
        {
            //  e.printStackTrace();
        }
    }//init()
    public void service(ServletRequest req,ServletResponse res)
            throws ServletException,IOException
    {
        Statement st=null;   ResultSet rs=null;
        String user=req.getParameter("uid");
        String pwd=req.getParameter("pwd");
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("<HTML>");
        out.println("<BODY BGCOLOR=wheat>");
        try
        {
            st=con.createStatement();
            String sql="SELECT * FROM OURUSERS WHERE USR='"+user+"'
                    AND PASSWORD='"+pwd+"'";
            rs=st.executeQuery(sql);
```


AVS

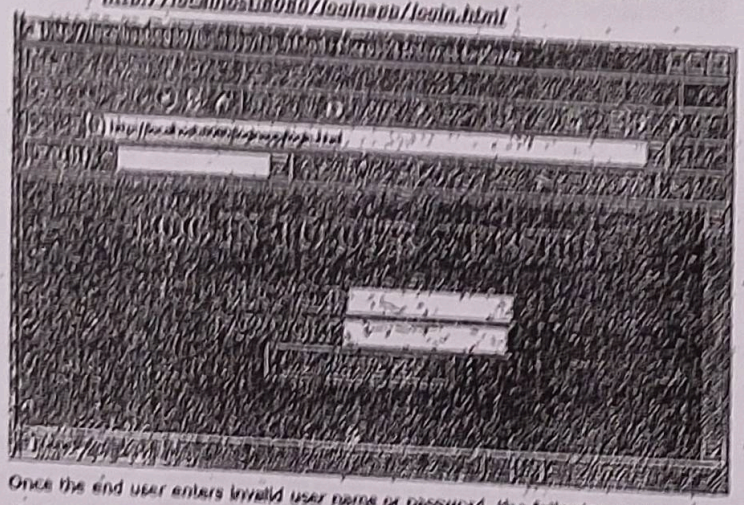
```
        {
        if(rs.next())
         out.println("<H2>WELCOME"+user+ "TO OUR WEB SITE</H2>");
        else
        {
            out.println("<H2>INVALID USER OR PASSWORD</H2>");
            out.println("<A HREF=./login.html> LOGIN AGAIN</A>");
        }
    }//try
    catch(Exception e){  e.printStackTrace();  }
    finally
    {
        try
        {
          if(rs !=null)
                rs.close();
          if(st !=null)
                st.close();
        }
        catch(Exception e){ e.printStackTrace(); }
    }//finally
    out.println("</BODY >");   out.println("</HTML>");
    out.close();
}//service()
public void destroy()
{
        try
        {
          if(con !=null)
                con.close();
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
}//destroy()  }//class
```

After developing the Servlet source code, compile it and copy the class file into the classes directory. Once all the files in the directory structure are ready, copy the loginapp into the webapps directory of the tomcat installation directory. Then start the Tomcat container. We get the following message on the Tomcat console.

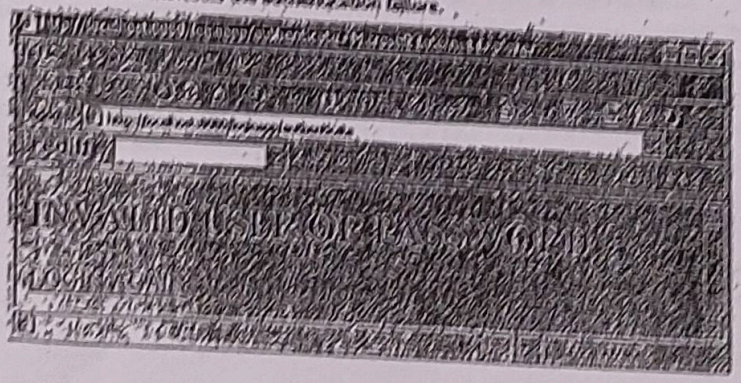
Installing web application at context path /loginapp from URL file:D:\Tomcat 5.0\webapps\loginapp

The above message is an indication that the web application is successfully deployed. Now, open the browser and type the following URL to get the login screen.

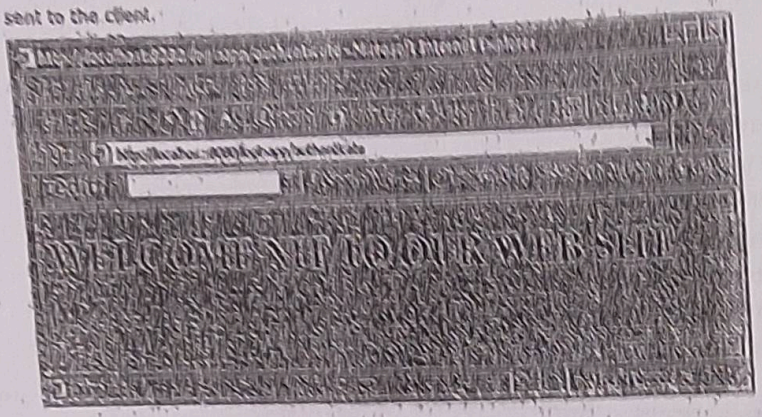
http://localhost:8080/loginapp/login.html



Once the end user enters invalid user name or password, the following screen will be displayed which indicates the authentication failure.



If the user name and password are valid for the web site, the following screen will be sent to the client.



In the above welcome message, NIT is the user name. The greeting message is dynamically generated basing on the user name. This is what is known as dynamic web content generation.

If we observe the implementation of AuthenticationServlet class, inside the init method we provided the database connection to the Servlet. It ensures that during its initialization itself, the Servlet acquires the database connection.

In the service method, by calling getRequestParameter method, we retrieved the user name and password entered in the login form.

In the database already a table by name ourusers exists. It has two columns usr and password. In that table, the list of valid users is maintained. Servlet needs to find whether the user name and password it received from the html form does exist in the database. Therefore, Servlet submitted SELECT SQL statement to the database by calling executeQuery method on the Statement object. This method returned the ResultSet object. In the if condition next method is called on the ResultSet object to find if the intended record exists. In the else part of the condition error message is displayed, a hyper link is created and sent to the client to enable the login again.

ServletResponse object is used to get an output stream to the web client by calling getWriter method. Calling setContentType method on the response object sets the MIME type to the server response to be given to the client.

In the destroy method, connection is closed. This method is called only when the Servlet is un-deployed.

Deployment Descriptor

Deployment descriptor is an xml file. According to SERVLET specification, each web application should have one deployment descriptor whose name should be web.xml. Its location is directly under the WEB-INF directory.

Deployment descriptor plays a major role and it is a very important part of the web application. It conveys to the container the requirements of our web application. In the deployment descriptor we perform one or more of the following things.

- Registering the servlets
- Creating URL mappings for the servlets
- Supplying initialization parameters to the individual servlets and to the whole web application
- Configuring the filters to the application
- Specifying the listeners to the application web components
- Specifying error pages
- Configuring application's welcome files
- Configuring the session timeouts
- Configuring the security constraints
- Specifying the kind of initialization for the servlets
- Specifying the tag libraries
- Specifying the MIME type mappings

The above list is the only few things that we can specify in the web.xml.

Assignment web applications

1) Web application in which end-user enters an employee number into the web page. Web server should send the employee details to the web client. For employee details, database should be contacted.
2) Web application that gives a web form to the end user upon request. In that form end user should be able to enter book details like ISBN, title, author and price. Data should be inserted into the Books table of the database.
3) Web application that provides a web form to the end user. End user should be able to get the book details in to the form fields when ISBN is entered. Provide a control on the form to perform update operation as well.
4) Web application that gives a web form to the user in which end-user enters the salary range in the web form field. Server should send all the employees' details whose salaries are beyond the specified salary range.

HttpServlet Life cycle

GenericServlet cannot provide the complete http functionality required for our servlets. That is the reason why almost all the times our Servlet class extends the HttpServlet. How does the Servlet container invokes our http Servlet is described below.

Step 1: - The Servlet container (simply server) loads the Servlet class that we developed and deployed into it. This can happen either at the time of first client request or at the time of application startup.

Step 2: - The Servlet engine (container) creates the instance of the just loaded Servlet class.

Step 3: - The Servlet engine creates the instance of some child class of ServletConfig interface. This child class implementation changes from container to container. This instance contains initialization information and application/container context information.

Step 4: - Servlet container calls the init method on the Servlet instance. While calling, it supplies the reference of ServletConfig type to this method. This reference refers to the object created in step three. We can use this reference to retrieve the initialization and context information.

Step 5: - Servlet engine creates two instances basing upon the client request. These two instances are container implemented some child classes of ServletRequest interface and ServletResponse interface respectively.

Step 6: - Servlet container calls the service method on the Servlet instance. While calling this method, container supplies the references of type ServletRequest and ServletResponse to this method. In this step http specific ness commences. The request object becomes HttpServletRequest and the response object becomes HttpServletResponse object.

Step 7: - Within the public service method a call is made to the protected service method. The arguments supplied to this method are HttpServletRequest object and HttpServletResponse object. The default functionality of this method is to calibrate the incoming http request type (GET/POST). Basing on the request type, this method calls either doGet method or doPost method on the Servlet instance.

Step 8: - In HttpServlet we always write our application logic either in the doGet() or doPost(). If the server receives another client request, process starts from Step 5 again.

Step 9: - Container calls destroy() when the Servlet is un-deployed.

SERVLET API BASICS

SERVLET API is packaged and made available to Servlet developers through two packages.

javax.servlet: The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a Servlet class and the runtime environment provided for an instance of such a class by a conforming Servlet container.

List of important interfaces and classes in javax.servlet package are as follows.

- Servlet
- ServletConfig
- ServletRequest
- ServletResponse
- ServletContext
- ~~RequestDispatcher~~
- Filter
- FilterConfig
- FilterChain
- GenericServlet
- ServletOutputStream
- ServletInputStream
- ServletException

javax.servlet.http: The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a Servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming Servlet container.

List of important interfaces and classes in javax.servlet.http package are as follows.

- HttpServletRequest
- HttpServletResponse
- HttpSession
- HttpSessionListener
- HttpServlet
- Cookie

javax.servlet.Servlet interface

This interface provides the life cycle methods to a Servlet class. The methods of this interface are,

- init(ServletConfig)
- service(ServletRequest, ServletResponse)
- destroy()
- ServletConfig getServletConfig()
- String getServletInfo()

init(ServletConfig config) : Servlet engine calls this method after the Servlet instance creation. In the specification, guarantees that init method is called only once in just after instance of a Servlet, and this method will be completed one before any requests are passed to the Servlet. In case the init() method is not completely executed correctly, it throws ServletException.

service(ServletRequestrequest, ServletResponse response) : This is the entry point for executing application logic for the servlets. This method is called only after successful completion of the init() method. This method takes two arguments. ServletRequest object provides methods to access the web client request data and response object provides methods to build the response.

destroy() : Container calls this method before removing Servlet from service. getServletConfig() method returns ServletConfig object, which contains initialization and startup parameters for this Servlet. The ServletConfig object returned is the one passed to the init method.

getServletInfo() method returns a String that allows the servlet to return basic information about itself, such as author, version and copyright.

According to SERVLET specification, every Servlet instance should directly or indirectly implement this interface. Rather all the times we implement it indirectly by defining our own Servlet class extending either GenericServlet class or HttpServlet class. Servlet interface specifies the contract between the Servlet container and our Servlet instance. According to the principles of object oriented programming, an object can communicate with another object as long as the first object can reference the second object with a known interface without need of knowing the name of the implementation class. In case of servlets, your servlet class is the interface Servlet engine use to interact our Servlet instance.

Harish Vishwakarma

javax.servlet.ServletConfig Interface

Container creates the instance of the implementation class of this interface. During initialization phase, container supplies the ServletConfig reference to the Servlet instance via init method. Servlet engine provides initialization and context information to the Servlet instance using ServletConfig object. This interface has the following methods.

- String getInitParameter(String): - This method returns a string that contains the value of the named initialization parameter. It takes the name of the parameter as argument.
- Enumeration getInitParameterNames(): - This method returns the names of all the initialization parameters in the form an Enumeration of String objects.
- String getServletName(): - This method returns the registration name of the Servlet instance in that web application to which this ServletConfig object is supplied.
- ServletContext getServletContext(): - This method returns the reference to the ServletContext in which the caller is executing.

Supplying initialization parameters to the Servlet

In the deployment descriptor (web.xml) we supply the initialization parameters to the Servlet as follows..

```
<init-param>
    <param-name>name</param-name>
    <param-value>value</param-value>
</init-param>
```

Servlet container reads the parameters and supplies them to the Servlet using ServletConfig object. We call getInitParameter method on the config object in the Servlet to retrive the param values.

For example, init method can be implemented as follows.

```
public void init(ServletConfig config) throws ServletException
{
  String value=config.getInitParameter("name");
}
```

Whatever name is given to the parameter in the DD file that name should be supplied to the getInitParameterName method. It returns the value specified in the DD file as a String object. We can supply any number of init-prams to a Servlet.

Example Servlet that handles a post request from a web form

Web application in which, a web form allows the end-user to perform online transfer of funds from Savings Account to Current Account.

Directory Structure

```
fundstransferapp
    transfer.html
    WEB-INF
        web.xml
        classes
            MoneyTransferServlet.class
        lib
            classes12.jar
```

Web form URL

http://localhost:8080/fundstransferapp/transfer.html

Source code of transfer.html

```
<HTML>
<BODY>
<CENTER>
<H2> ON LINE FUNDS TRANSFER INTERFACE </H2>
<FORM ACTION="./transfer" METHOD="POST">
    Savings A/C NO<INPUT TYPE=text NAME="sbno"><BR>
    Current A/C NO<INPUT TYPE=text NAME="sbno"><BR>
    TransferAmount<INPUT TYPE=text NAME="amount"><BR>
    <INPUT TYPE=submit VALUE=Transfer the fund >
</FORM>
</CENTER>
</BODY>
</HTML>
```

Step2

Source code of web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
  <servlet>
    <servlet-name>moneytransfer</servlet-name>
    <servlet-class>MoneyTransferServlet</servlet-class>
    <init-param>
      <param-name>driver</param-name>
      <param-value>oracle.jdbc.driver.OracleDriver</param-value>
    </init-param>
    <init-param>
      <param-name>url</param-name>
      <param-value>jdbc:oracle:thin:@localhost:1521:orcl</param-value>
    </init-param>
    <init-param>
      <param-name>user</param-name>
      <param-value>scott</param-value>
    </init-param>
    <init-param>
      <param-name>pass</param-name>
      <param-value>tiger</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
      <servlet-name>moneytransfer</servlet-name>
      <url-pattern>/transfer</url-pattern>
  </servlet-mapping>
</web-app>
```

Source code of the Servlet

```
import java.io.*;import java.sql.*;
import javax.servlet.*;import javax.servlet.http.*;
public class  MoneyTransferServlet extends HttpServlet
{
    Connection con;
    public void init(ServletConfig config)throws ServletException
    {
      try
      {
        String driver=config.getInitParameter("driver");
        String url= config.getInitParameter("url");
        String user= config.getInitParameter("user");
        String pwd= config.getInitParameter("pass");
        Class.forName(driver);
        con=DriverManager.getConnection(url,user,pwd);
      }
      catch(Exception e)  { e.printStackTrace();  }
    }//init
    public void doPost(HttpServletRequest req,HttpServletResponse res)
          throws ServletException,IOException
    {
      Statement st=null;
      int sbaccno=Integer.parseInt(req.getParameter("sbno"));
      int caccno=Integer.parseInt(req.getParameter("cano"));
      float amount=Float.parseFloat(req.getParameter("amount"));
      res.setContentType("text/html");   PrintWriter out=res.getWriter();
      out.println("<HTML>");out.println("<BODY BGCOLOR=wheat>");
      try
      {
        st=con.createStatement();
        String sql1="UPDATE SAVINGS SET BALANCE=BALANCE -"+amount+
                    "WHERE ACCNO="+sbaccno;
        String sql2="UPDATE CURRANT SET BALANCE=BALANCE +" +amount+
                    "WHERE ACCNO="+caccno;
```

tablename

Naresh I Technologies

MjNv SaI

MjNV SaI

Savings

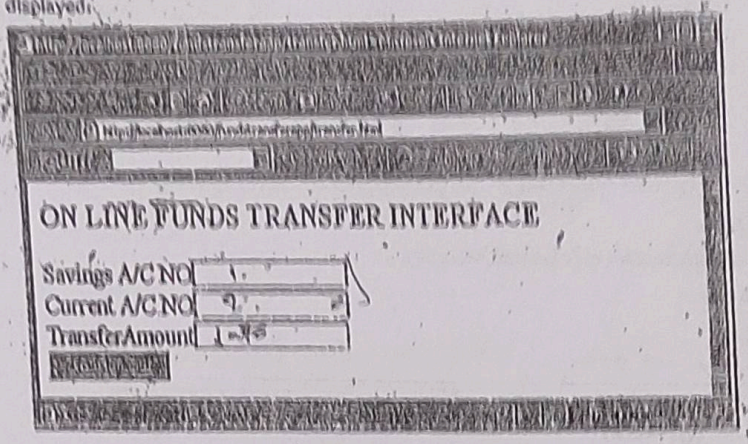
Ac	Bal

CURRENT

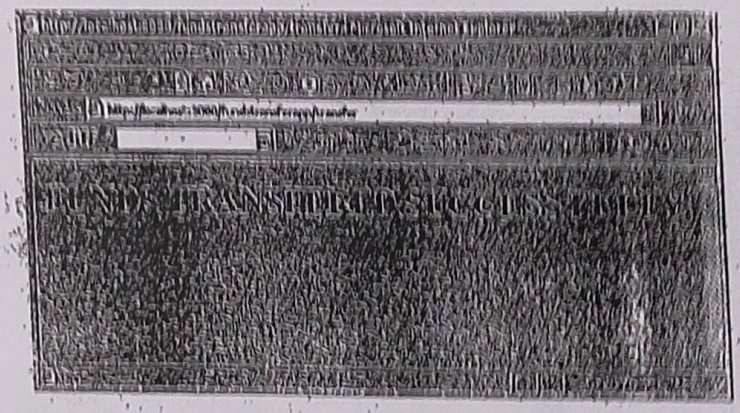
Acc	Balance

```
st.executeUpdate(sql1);
st.executeUpdate(sql2);
out.println("<h2>FUNDS TRANSFERED SUCCESSFULLY</h2>");
}//try
catch(Exception e){e.printStackTrace();}
finally
{
  try
  {
    if(st!=null)
      st.close();
  }
  catch(Exception e)
  {
    e.printStackTrace();
  }
}//finally
out.println("</BODY>");
out.println("</HTML>");
out.close();
}//doPost()
public void destroy()
{
    try
    {
      if(con!=null)
          con.close();

    }
    catch(Exception e)
    {
          e.printStackTrace();
    }
}
}//class
```

When the end-user requests for the funds transfer web page, the following screen is displayed:



If funds transfer occurs normally, the following screen is displayed



In the init method of the MoneyTransferServlet class we retrieved the initialization parameters. Database driver class, connection URL, user name and password are supplied from the web.xml declaratively. Whenever we want to change the connection related things, we need not change the source code of the Servlet. Only few entries in the web.xml are to be modified.


```
Enumeration<String> headerNames = req.getHeaderNames();
while(headerNames.hasMoreElements())
{ String headerName = headerNames.nextElement();
  out.println("Header Name : <em>" + headerName);
  String headerValue = req.getHeader(headerName);
  out.println("<em>, Header Value : <em>" +
                                headerValue);
  out.println("<br />");
}
```

Request Header is used to pass additional information about the requested itself to the server. It can be used by the client to pass the useful information. These methods used to get the header information

VSC

javax.servlet.http.HttpServletRequest Interface

This interface extends ServletRequest interface and provides the information for Http Servlet. The Servlet engine creates an object that implements HttpServletRequest and passes it as an argument to the servlet's servicing methods.

List of important methods in this interface

- String getMethod(): - It returns the name of the HTTP method with which this request is made.
- String getQueryString(): - Returns the query string that is contained in the request URL after the path. This method returns null if the URL does not have a query string.
- HttpSession getSession(): - Returns the current session associated with the request.
- void setAttribute(String name, Object value): - It stores an attribute in the request object.
- Object getAttribute(String name): -Returns the value of the named attribute as an Object, or null if the request has no attributes.
- Enumeration getAttributeNames(): - Returns an Enumeration containing the names of the attributes available in this request. This method returns an empty Enumeration if there are no attributes for the request object.
- boolean isRequestedSessionIdFromCookie(): - It returns true if the session id came in as a cookie; otherwise, false.
- boolean isRequestedSessionIdFromURL(): - It returns true if the session id came in as a part of URL; otherwise, false.
- String getServletPath(): - Returns the part of this request's URL that calls the Servlet. This path starts with a "/" character and includes either the Servlet name or a path to the Servlet, but does not include query string.
- Cookie[] getCookies(): - Returns an array containing all of the Cookie objects the client sent with this request. This method returns null if no cookies are sent.
- String getHeader(String header): - Returns the value of the specified request header as a String. If the request does not include a header of the specified name, this method returns null.
- Enumeration getHeaderNames(): - Returns an enumeration of all the header names this request contains.

HTTP header

Header Name	Header value
host	localhost:8080 (server-name)
connection	
Accept	MIME type
User-agent	name of the b/w
accept-encoding	what type of algorithm we using to apply gzip
accept-language	type of language b/w supply

URI (path) of servlet → /myurlpat

output

Not the alias name and not complete URL, to retrive complete URL use (getRequestURL())

Request obj

- int getContentLength(): - Returns the length, in bytes, of the request body made available by the input stream. Returns -1 if the length is not known.
- String getContentType(): - Returns the MIME type of the body of the request, or null if the type is not known.
- String getParameter(String name): - Returns the value of the request parameter as String, or null if the parameter does not exist.
- Enumeration getParameterNames(): - Returns an Enumeration of String objects containing the names of the parameters contained in this request. If the request has no parameters, the method returns an empty Enumeration.
- String getProtocol(): - Returns the name and version of the protocol the request uses in the form protocol/major version.minor version, for example, HTTP/1.1.
- String getServerName(): -Returns the host name of the server to which the request was sent. It is the value of the part before ":" in the Host header, if any, or the resolved server name or the server IP address.
- String getRemoteHost: - Returns the fully qualified name of the client or the last proxy that sent the request. If the container cannot or chooses not to resolve the hostname, this method returns the dotted-string form of the IP address.
- int getServerPort(): - Returns the port number to which the request was sent.

javax.servlet.http.HttpServletResponse interface

This interface extends ServletResponse interface. The Servlet engine creates an object that implements HttpServletResponse and passes it as an argument to the servlet's servicing methods. We use this object to customize the response that has to be given to the client.

List of important methods in this interface

- addCookie(Cookie c): -This method adds the specified cookie to the response. This method can be called multiple times to set more than one cookie.
- addHeader(String name, String value): Adds a response header with the given name and value.
- sendError(int sc, String msg): - This method sends an error response to the client using the specified status. The server defaults to creating the response to look like an HTML formatted server error page containing specific message.

Date d = new Date();

out.println("Date is" + d);

response.setHeader("Refresh", "10");

imp request or response

- String encodeURL(String url): - This method encodes the specified URL by including the session ID in it, or , if encoding not needed, returns the URL unchanged. The implementation of this method includes logic to determine whether the session id needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary. For robust session tracking, all URLs emitted by a Servlet should be run through this method. Otherwise, URL rewriting cannot be used with browsers, which do not support cookies.

- sendRedirect(String location): -sends a temporary redirect response to the client using the specified redirect location URL .

- setContentType(String type): - Sets the content type of the response being sent to the client, if the response has not been committed yet. It takes a String specifying the MIME type of the content as argument.

- setContentLength(int len): - Sets the length of the content body in the response. In Http servlets, this method sets the HTTP Content-Length header.

- String getContentType(): - Returns the content type used for the MIME body sent in this response.

- ServletOutputStream getOutputStream(): - Returns ServletOutputStream suitable for writing binary data in the response. Calling flush() on the ServletOutputStream commits the response.

- PrintWriter getWriter(): - Returns a PrintWriter object than can send character text to the client. Calling flush() on the PrintWriter commits the response.

- Boolean isCommitted(): - Returns true or false indicating if the response has been committed. A committed response has already had its status code and headers written.

To share the data b/w several servlets

javax.servlet.ServletContext

pplication can have multiple servlets to process client requests. Each Servlet Individual ServletConfig object. However, all the servlets are running in the text. This context is unique for the web application. ServletConfig instances reference to the same context. When the web application is installed in the web container, an instance that implements ServletContext interface is created. This instance is the abstraction of the web application itself. A ServletContext object is the Servlet's connection to both the container and other parts of the web application. multiple servlets can share information and resources by using ServletContext object. With the help of this object one Servlet can communicate with another Servlet running in the same web application.

List of important methods in ServletContext Interface

Methods to deal with attributes and init parameters:-

- (public String getInitParameter(String name)):- Returns a String containing the value of the named context-wide initialization parameter, or null if the parameter does not exist. This method can make available configuration information useful to an entire "web application". For example, it can provide a webmaster's email address or the name of a system that holds critical data.
- (public Enumeration getInitParamNames()):- Returns the names of the context's initialization parameters as an Enumeration of String objects, or an empty Enumeration if the context has no initialization parameters.
- (public void setAttribute(String name, Object value)):- Binds an object to a given attribute name in this servlet context. If the name specified is already used for an attribute, this method will replace the attribute.
- (public Object getAttribute(String name)):- Returns the attribute bound to the ServletContext object with the given name, or null if there is no attribute by that name.
- (public Enumeration getAttributeName()):- Returns an Enumeration containing the attribute names available within this Servlet context.

Methods to get info about Servlet container and logging information:

- (public void log(String info)):- Writes the specified message to a Servlet log file, usually an event log. The name and type of the Servlet log file is specific to the Servlet container.
- (public String getServerInfo()):- Returns the name and version of the Servlet container on which the Servlet is running.

(1) Define cookies
(2) What are web services
(3) What is EJB
(4) Discuss SAX
(5) What is a JSP page
(6) Discuss XML
(7) define frames
(8) define list
(9) What is ADO
(10) Define HTTP

UNIT - I

1. What are Java Scripts
   b. List, Tables, Images, Forms
2. What are frames
   b. Briefly explain the log
   c. What are cascading style sheets
3. What are Servlets
   b. Write with a program

UNIT - II

3. Discuss DOM & SAX
   b. Explain document type definition
4. Explain Bean managed persistence (BMP)
   b. Illustrate stateful class

- public int getMajorVersion():-Returns the major version of the Java Servlet API that this Servlet container supports. It returns a number 2. (In Servlets 2.4 version 2 is major version)
- public int getMinorVersion():- This method returns a number 4 (If container supports Servlets 2.4).

Method to communicate with other servlets in the web application

- RequestDispatcher getRequestDispatcher(String name):- Returns the RequestDispatcher object.

Example web application in which, two servlets share data. First Servlet retrieves data from the web form and shares it with another Servlet. Second Servlet retrieves it and sends the output to the client.

Directory Structure

```
sharingapp
    userdetails.html
    WEB-INF
        web.xml
        classes
            StorageServlet.class
            RetrieveServlet.class
```

Web application URL

http://localhost:8080/sharingapp/userdetails.html

Source code of userdetails.html

```
<HTML>
  <BODY BGCOLOR=pink>
  <CENTER>
      <H1>PERSONAL DETAILS</H1>
      <FORM ACTION="./store">
          EMAIL ID<INPUT TYPE=text NAME="email"><BR>
          PHONE  <INPUT TYPE=text NAME="phone"><BR><BR>
          <INPUT TYPE=submit VALUE=SUBMITDETAILS>
      </FORM>  </CENTER>  </BODY></HTML>
```

(8.) what is Jdbc & database programming
b. Explain the STRUTS framework
c. How to deploy. Display JavaBeans in JSP Pag.

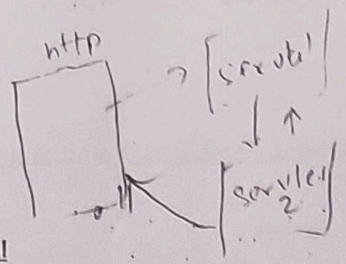
UNIT-III

5. Explain Servlet API
b. How are HTTP Request handled
c. How are Servlet parameters read?
6. Discuss the anatomy of a JSP Page
b. Explain the Tomcat Server & Tomcat Testing?

UNIT-IV

7. How do you generate dynamic Content
b. How do you display values using an Expression to set an attribute.

Res. Send Redirect ("—");



Source code of web.xml

```
<web-app>
  <context-param>
    <param-name>admin  </param-name>
    <param-value>admin@nit.com</param-value>
  </context-param>
  <servlet>
      <servlet-name>storage</servlet-name>
      <servlet-class>StorageServlet</servlet-class>
  </servlet>
  <servlet>
      <servlet-name>retrieval</servlet-name>
      <servlet-class>RetrieveServlet</servlet-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>storage</servlet-name>
      <url-pattern>/store</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
      <servlet-name>retrieval</servlet-name>
      <url-pattern>/retrieve</url-pattern>
  </servlet-mapping>
</web-app>
```

|| setter

STEP - 20

Source code of StorageServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class StorageServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse response)
throws      IOException,ServletException
  {
        String email=request.getParameter("email");
        String phone=request.getParameter("phone");
        response.setContentType("text/html");
```

STEP - 3

// with in the server : init param
with different servers → context param //


```
        PrintWriter pw=response.getWriter();
        ServletContext sc=getServletContext();
        sc.setAttribute("mail",email);
        sc.setAttribute("telephone",phone);
        String adminmail=sc.getInitParameter("admin");
        pw.println("<HTML>");
        pw.println("<BODY BGCOLOR=wheat>");
        pw.println("<H3>Websiteadminisavailable at "+adminmail+"</H3>");
        pw.println("<H2><AHREF=./retrieve>GET YOUR DETAILS HERE
</A></H2>");
        pw.println("</BODY>");
        pw.println("</HTML>");
    pw.close();
  }
}
```

//setters

Source code of RetriveServlet.java

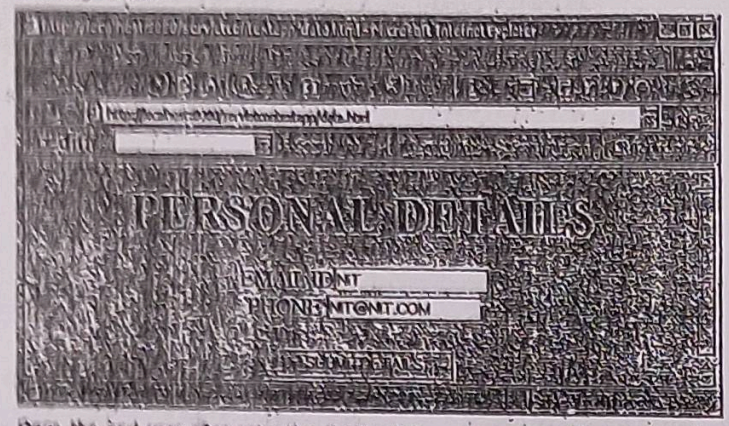
STEP - 4

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

        response.setContentType("text/html");
        PrintWriter pw=response.getWriter();
        ServletContext sc=getServletContext();
        String mail=(String)sc.getAttribute("mail");
        String phone=(String)sc.getAttribute("telephone");
        String adminmail=sc.getInitParameter("admin");
        pw.println("<HTML>");
        pw.println("<BODY BGCOLOR=cyan>");
```

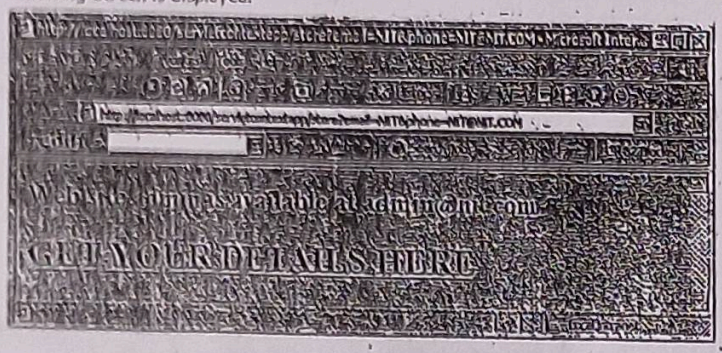
//getters

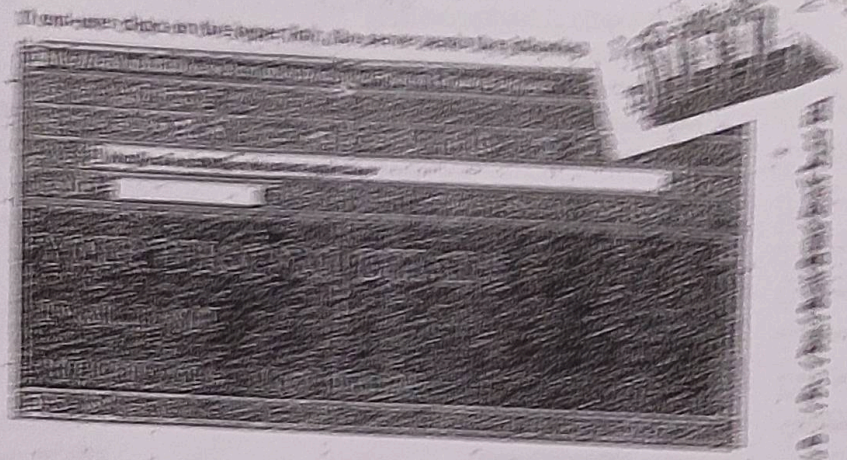
```
pw.println("<H2> YOUR PERSONAL DETAILS</H2>");
pw.println("<H3> EMAIL ID: "+mail+"</H2>");
pw.println("<H3> TELEPHONE # :"+phone+"</H2>");
pw.println("<H3>Websiteadminisavailable at "+adminmail+"</H3>");
pw.println("</BODY>");
pw.println("</HTML>");
pw.close();
}
```

Once we deploy the application and type the given URL in the browser, we get the following screen.



Once the end user after enter the details clicks on the submit details button, the following screen is displayed.





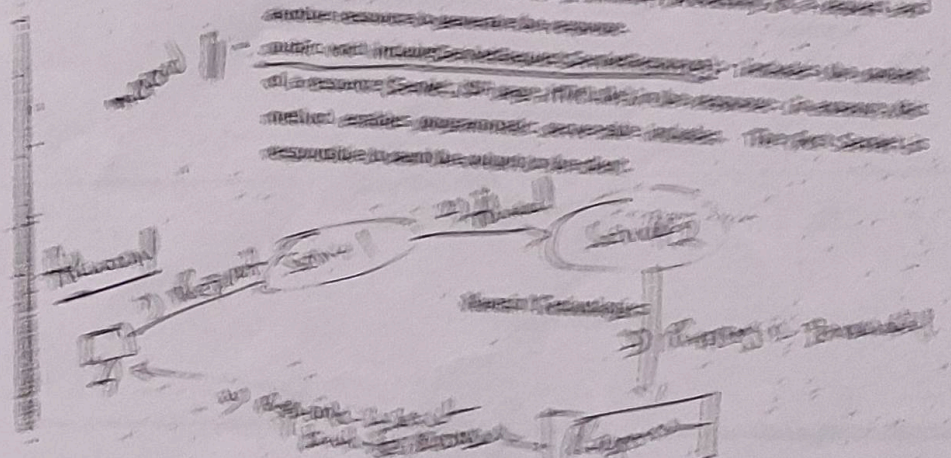
In the above application, ServerServer retrieves the from the set stored in the ServerContext object. ServerServer calling the client method (ServerContext) and retrieves the data. It does the response back to the client.

Request dispatching mechanism

A Servlet receives the HTTP request from the web client and does the processing partially and hands over the request to another Servlet for further processing. This mechanism is known as request dispatching. Servlet API has provided particular RequestDispatcher interface to implement request dispatching.

This interface has 2 methods:

- public void forward(ServletRequest, ServletResponse) throws ... : forwards a request from a Servlet to another resource (Servlet, JSP file, or HTML file). The method allows one Servlet to do preliminary processing of a request and another resource to generate the response.
- public void include(ServletRequest, ServletResponse) throws ... : includes the content of a resource (Servlet, JSP page, HTML file) in the response. Invoking the method enables programmers to include results. The first Servlet is responsible to send the output to the client.



Source code of NetServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class NetServlet extends HttpServlet
{
  public void doGet(HttpServletRequest request,HttpServletResponse response)
      throws IOException,ServletException
  {
       response.setContentType("text/html");
       PrintWriter out=response.getWriter();
       Float gross=(Float)request.getAttribute("gross");
       float  net=gross.floatValue()-2000-1000;
       getServletContext().
            getRequestDispatcher("/caption.html").include(request,response);
       out.println("<HTML>");
       out.println("<BODY BGCOLOR=cyan>");
       out.println("YOUR NET SALARY IS :"+net);
       out.priotln("</BODY>");
       out.println("</HTML>");
       out.close();
  }//doGet()
}//class
```

Source code of caption.html

```
<HTML>
<BODY>
<MARQUEE><FONT  size=5  color=green>NO  SUBSTITUTE  FOR  HARD
WORK</FONT></MARQUEE>
</BODY>
</HTML>
```

APPLYING FILTERS TO WEB RESOURCES

A Filter is a container-managed object that is declaratively inserted within the request-response cycle of http client and http server. A filter instance acts as an interceptor. It is a preprocessor of the request before reaches a Servlet and a post processor of the response leaving a Servlet. A filter can perform the following.

- Intercept a servlet's invocation before a Servlet is called.
- Examine the request before a Servlet is called.
- Modify the request headers and request data.
- Modify the response headers and response data.
- Intercept a servlet's invocation after the Servlet is called.

A filter object implements javax.servlet.Filter interface. This interface provides 3 life cycle methods.

- public void init(FilterConfig config):-Called by the web container to indicate to a filter that it is being placed into service. The Servlet container calls the init method exactly once after instantiating the filter. The init method must complete successfully before the filter is asked to do any filtering work.
- public void doFilter(ServletRequest,ServletResponse, FilterChain):- It is called by the container each time a request/response pair is passed through the chain due to a client request for a resource at the end of the chain. The FilterChain passed in to this method allows the Filter to pass on the request and response to the next entity in the chain.
- public void destroy():- Called by the web container to indicate to a filter that it is being taken out of service. This method gives the filter an opportunity to clean up any resources that are being held.

Web application on the implementation of filter to a Servlet.

The directory structure of the application is as follows.

```
authenticatefilterapp
    login.html
    error.html
    WEB-INF
        web.xml
        classes
            AuthFilter.class
            LoginServlet.class
```

Source code of login.html

```
<HTML>
 <BODY>
   <FORM action="./nit" METHOD="POST">
          USERNAME<input type=text name=user><br>
          PASSWORD<input type=password name=password><br>
          <input type=submit value=login>
   </FORM>
   </BODY>
  </HTML>
```

Source code of error.html

```
<HTML>
<BODY>
<CENTER><H3>INVALID LOGIN OR PASSWORD</H3></CENTER>
<FORM action="./nit" METHOD=POST>
USERNAME<input type=text name=user><br>
PASSWORD<input type=password name=password><br>
<input type=submit value=login>
</BODY>
</HTML>
```

Source code of web.xml

```
<web-app>
   <filter>
     <filter-name>auth</filter-name>
     <filter-class>AuthFilter</filter-class>
   </filter>
   <filter-mapping>
      <filter-name>auth</filter-name>
      <url-pattern>/nit</url-pattern>
   </filter-mapping>
  <servlet>
     <servlet-name>Login</servlet-name>
     <servlet-class>LoginServlet</servlet-class>
  </servlet>
```

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

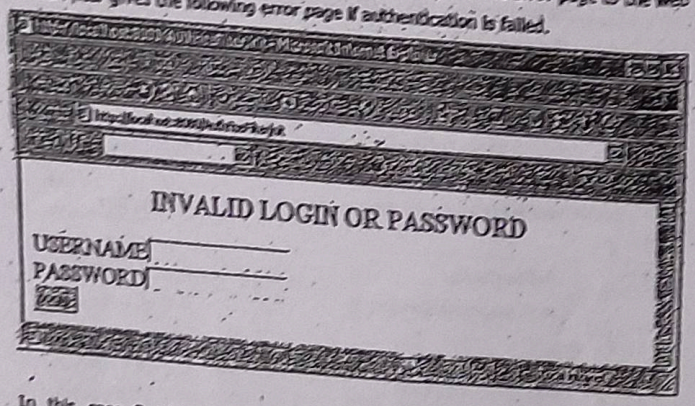
Source code of AuthFilter.class

```
import javax.servlet.*;
import java.io.*;
public class AuthFilter implements Filter
{
        FilterConfig f=null;
        ServletContext sc=null;
        public void doFilter(ServletRequest request, ... ServletResponse
response,FilterChain chain)
                throws IOException,ServletException
        {
                String user=request.getParameter("user");
                String password=request.getParameter("password");
                if( ! (user.equals(password)))
                  sc.getRequestDispatcher("/one.html").forward(request,response);
                else
                        chain.doFilter(request,response);
        }
        public void init(FilterConfig filterConfig) throws ServletException
        {
                f=filterConfig;
                sc=f.getServletContext();
        }
        public void destroy()
        {
                f=null;
        }
}
```

Source code of LoginServlet.class

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class LoginServlet extends HttpServlet
{
  public void doPost(HttpServletRequest request,HttpServletResponse response)
throws IOException,ServletException
      {
          response.setContentType("text/html");
          PrintWriter pw=response.getWriter();
          pw.println("WELCOME TO OUR WEBSITE");
          pw.close();
      }
}
```

In this application, filter is applied to the LoginServlet declaratively in the web.xml using the filter-mapping element. When the filter is applied to a resource, any request with the url name will not be directed to the resource first. The filter is activated. If the authentication gets failed, filter only gives the error page to the web client. Filter gives the following error page if authentication is failed.



In this case Servlet will not get the request there by disburdened from the authentication work.

VSC

STATE & SESSION MANAGEM

Http is a stateless protocol. A web client opens a connection and requests some resource. The server responds with the requested resource and closes the connection with client. After closing the connection, the http server does not remember any information about the client. The server considers the next request from the same client as a fresh request, with no relation to the previous request. This is what the stateless nature of the http protocol.

In enterprise web application it is mandatory that client and its associated data must be tracked at server side across multiple requests. There are four approaches to achieve this.

- Hidden form fields
- Cookies
- Session tracking with SERVLET API
- URL rewriting

Note: - For Hidden form fields mechanism there is no SERVLET API support available.

Hidden form fields: - a hidden field is similar to an ordinary input field in HTML. The only difference is that the hidden field doesn't have an associated user interface element. When the form that contains these hidden fields is submitted, the values of these fields are sent with the request. On the server side, these values are received as request parameters. This mechanism works only when form is submitted, not when you click the hyperlink. The key point is that client associated data travels invisibly from client to server and server to client in this mechanism. That is how state is managed across multiple requests. In J2EE environment this mechanism is not so widely used, as there is no SERVLET API support available.

Cookies Mechanism

This is a widely used mechanism for both state management and session management. A cookie is a single piece of information stored on the client, on behalf of the server. This information is returned to the server with every request, in addition to the requested document. It may choose to return some state information to the browser. This information includes a URL range within which the information should be returned to the server. The URL range comprises the domain name and some path within the domain. Whenever the browser requests a resource, it checks the URL against the URL range of all available cookies. If a match is found, the

cookie → object → name value age

→ cookie is simply an object

Two types of cookies, < non persistent cookie, Persistent cookies

cookie is also returned with the request. This helps the server overcome the stateless nature of the Http protocol.

Using the cookie mechanism in a Servlet has the following steps:

Step 1: creating the instance of javax.Servlet.http.Cookie class.

```
Cookie c=new Cookie ("user", "rama");
```

Step 2: setting other attributes of the cookie if required.

```
c.setMaxAge(3600);//It stores in the browser for one hour.
```

Step 3: Sending the cookie to the client with the response

```
Response. AddCookie ()
```

Step 4: To retrieve the cookie we call the following method.

```
Cookie c[] = request.getCookies();
```

List of Important methods in cookie class:

- Public String getName():- Returns the name of the cookie.
- Public String getValue ():- Returns the value of the cookie.
- Public void setMax Age(int expiry):- Sets the maximum age of the cookie in seconds. A positive value indicates that the cookie will expire after that many seconds have passed. Note that the value is the maximum age when the cookie will expire, not the cookie's current a A negative value means that the cookie is not stored persistently and will be deleted when the Web browser exits. A zero value causes the cookie to be deleted.
- Public void setDomain(String pattern):- Specifies the domain within which this cookies should be presented. A domain name begins with a dot (.foo.com) and means that the cookie is visible to servers in a specified Domain Name System (DNS) zone (for example, www.foo.com, but not a.b.foo.com). By default, cookies are only returned to the server that sent them.

Example web application on Cookie mechanism

Directory structure

```
cookiesapp
    shop.html
    WEB-INF
        web.xml
        classes
            CookieExample.class
```

Application URL: - http://localhost:8080/cookieapp/shop.html

Source code of shop.html

```
<HTML>
<BODY BGCOLOR="cyan">
<CENTER>
   <FORM ACTION="./cook" METHOD="post">
      <B>Select The Shirt</B>
      <select name="shirt"> <option value="Peter England">PeterEngland
       <option value="Arrow">Arrow<option value="Van Husslan">VanHussain
      </select><BR><BR><B>Select Book</B>
     <select name="book"> <option value="JCR">JCR
       <option value="RMI">RMI<option value="XML">XML
     </select><BR><BR>
     <input type="submit" value="ADD ITEMS TO CART">
     </FORM>
</CENTER>
</BODY>
</HTML>
```

Source code of web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
   <servlet>
      <servlet-name>create</servlet-name>
      <servlet-class>CookieExample</servlet-class>
   </servlet>
   <servlet-mapping>
      <servlet-name>create</servlet-name>
      <url-pattern>/cook</url-pattern>
   </servlet-mapping>
</web-app>
```

Source code of CookieExample.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class  CookieExample extends HttpServlet
{
        static int i=1;
        static int j=1;
        public void doPost(HttpServletRequest req, HttpServletResponse res)
        throws ServletException,IOException
        {
                String shirt=req.getParameter("shirt");
                String book=req.getParameter("book");
                res.setContentType("text/html");
                PrintWriter  pw=res.getWriter();
                Cookie c1=new Cookie("shirt"+i,shirt);
                Cookie c2=new Cookie("book"+j,book);
                res.addCookie(c1);   res.addCookie(c2);   i++;j++;
                pw.println("<h3>Items in the shopping cart are</h3>");
                pw.println("<h3>"+c1.getName()+" : "+shirt+"</h3>");
                pw.println("<h3>"+c2.getName()+" : "+book+"</h3>");
                Cookie c[]=req.getCookies();
                if(c !=null)
                {
                        for(int i=0;i<c.length;i++)
                pw.println("<h3>"+c[i].getName()+" : "+c[i].getValue()+"</h3>");
                }
                RequestDispatcher rd=req.getRequestDispatcher("/shop.html");
                rd.include(req,res);
                pw.close();
        }
}
```

SESSION TRACKING WITH HttpSession

Session tracking means that the container is able to associate a request with a client, and HttpSession object represents that association. The web container maintains this object for the duration of the client session, or a configurable timeout period. Since many clients interact with container, the container maintains individual HttpSession objects for each client. By using methods of HttpSession we can associate the client state also with the session object.

The Servlet API for session tracking is as follows.

- Javax.Servlet.http.HttpServletRequest for the creation and retrieval of HttpSession objects.
- Javax.Servlet.http.HttpSession interface for representing sessions. This interface provides methods for state maintenance also.

HttpServletRequest interface provides the following method to create or access the session object.

HttpSession getSession():-

The above method returns the reference to a container created object that implements HttpSession. If there is no session associated with the current request, the above method creates one. If already a session is associated with the client, it gives the reference to the existing session object.

List of Important methods in HttpSession

Methods for session lifetime:-

- public long getCreation Time():- Returns the time that the session was created, in milliseconds since Jan 1, 1970 00.00 hrs.
- public String getId():- Returns a String containing a unique identifier assigned to this session.
- public long getLastAccessedTime():- Returns the time that the session was last accessed by the client, in milliseconds since Jan1, 1970 00.00hrs. this method can be used to determine the period of inactivity between two consecutive requests from a client.
- public int getMaxInactiveInterval():-Returns the length of time in seconds that the session will remain active between requests before expiring.
- public void setMaxInactiveInterval(intseconds):-This methods sets the length of time in seconds that the session will remain active between requests before expiring. The web container makes sure that the session is automatically invalidated after expiry of this interval.

- public boolean isNew():- Returns true if the client does not yet know about the session or if the client chooses not to join the session. A client is considered to join a session when the client returns session tracking information previously sent by the server.
- public void invalidate():- Invalidates this session then unbinds any objects bound to it.

Methods for associating state with session:-

- Public void setAttribute(String name,Object value):- Stores a value in the session object.
- Public Object getAttribute(String name):- Returns the object bound with the specified name in this session, or null if no object is bound under the name.
- Public void removeAttribute(String name):- Removes the object from this session.
- Public Enumeration getAttributeNames():- Returns an Enumeration of String objects bound to the session.

Web application that implements session tracking using HttpSession

Directory Structure

```
sessionapp
    shop.html
    WEB-INF
        web.xml
        classes
            ShopServlet.class
```

application URL: http://localhost:8080/sessionapp/shop.html

step-1

Source code of shop.html

```
<HTML>
    <BODY BGCOLOR="wheat">
```

```
<CENTER>
        <H2>WELCOME TO SHOPPING MALL</H2>
 <FORM ACTION="./shopsession">
        SELECT PRODUCT CODE
        <SELECT name="pCode">
            <option value=101>101
            <option value=102>102
            <option value=103>103
            <option value=104>104
            <option value=105>105
        </select>
        <BR><BR>
 PRODUCT QNTY<INPUT TYPE="text"NAME="qty" VALUE=""><br><br>
 <INPUT TYPE="submit" NAME="submit" VALUE="ADD ITEM" >
 <INPUT TYPE="submit" NAME="submit" VALUE="REMOVE ITEM" >
 <INPUT TYPE="submit" NAME="submit" VALUE="SHOW ITEMS" >
 <INPUT TYPE="submit" NAME="submit" VALUE="PAY AMOUNT">
 <INPUT TYPE="submit" NAME="submit" VALUE="LOGOUT" >
 </CENTER>
 </FORM>
</BODY>
</HTML>
```

Source code of web.xml

```
<web-app>
  <servlet>
    <servlet-name>Shopping</servlet-name>
    <servlet-class>ShopServlet</servlet-class>
    </servlet>
    <servlet-mapping>
    <servlet-name>Shopping</servlet-name>
    <url-pattern>/shopsession</url-pattern>
  </servlet-mapping>
</web-app>
```

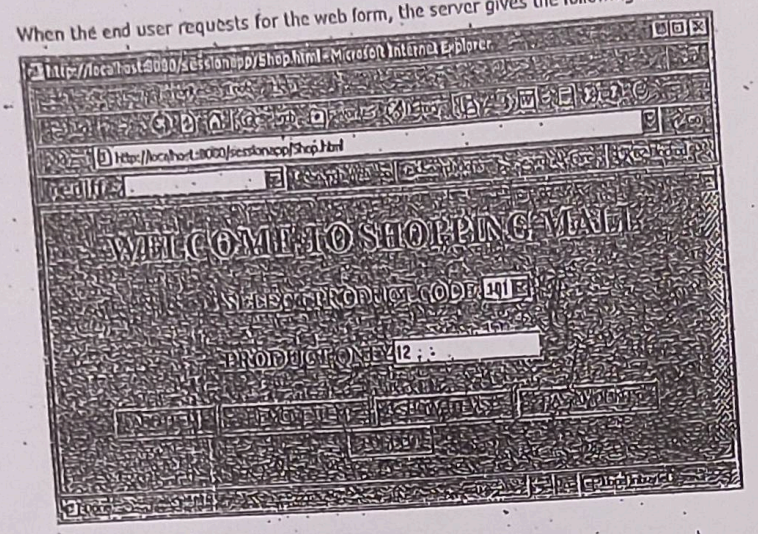
Source code of ShopServlet.java

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class  ShopServlet extends HttpServlet
{
        HttpSession session;
        String pCode,qty,clickButton;
        public void doPost(HttpServletRequest req,HttpServletResponse res)throws
        ServletException,IOException
        {
                res.setContentType("text/html");
                PrintWriter pw=res.getWriter();
                session=req.getSession(true);
                clickButton=req.getParameter("submit");
                if(clickButton.equals("ADD ITEM"))
                {
                        pCode=req.getParameter("pCode");
                        qty=req.getParameter("qty");
                        if(!(pCode.equals("") ||qty.equals("")))
                        {
                        session.setAttribute(pCode,qty);
                        res.sendRedirect("./shop.html");
                        }
                }
                else if(clickButton.equals("REMOVE ITEM"))
                {
                        pCode=req.getParameter("pCode");
                        session.removeAttribute(pCode);
                        res.sendRedirect("./shop.html");
                }
                else if(clickButton.equals("SHOW ITEMS"))
                {
                        Enumeration e=session.getAttributeNames();
                        if(e.hasMoreElements())
                        {
```

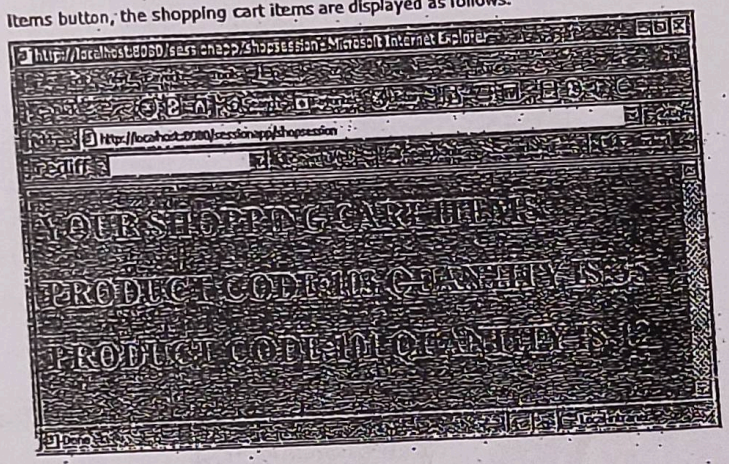
```
                    out.println("<H2><FONT COLOR=BLUE>YOUR
                        SHOPPING CART ITEMS</FONT></H2>");
                while(e.hasMoreElements())
                {
                    out.println("<BODY BGCOLOR=cyan>");
                    String code=(String)e.nextElement();
                    out.println("<H2>PRODUCT CODE:"+code);
                out.println("QUANTITY "+session.getAttribute(code)+
                                "</H2>");
                }
            }//end of if
            else
            {
                out.println("<BODY BGCOLOR=cyan>");
                out.println("<H2><FONT COLOR=red >YOU
                        ITEMS PLEASE</FONT></H2>");
            }

        }//close if
        else if(action.equals("CHECKOUT"))
        {
            session.invalidate();
            res.setContentType("text/html");
        }
        else if(action.equals("PAY AMOUNT"))
        {
            out.println("<BODY BGCOLOR=yellow>");
            out.println("<H2><FONT COLOR=red >PAYMENT LOGIC
            GOES HERE </FONT></H2>");
        }
        out.close();
    }//doPost
}//class
```

Control transfer to next page

When the end user requests for the web form, the server gives the following form.



After a series of interactions end user selects items. Finally when clicked on the show items button, the shopping cart items are displayed as follows.



URL REWRITING MECHANISM

When we are implementing session tracking using SERVLET API, container implicitly uses cookie mechanism to exchange the session id between the client and the server. If cookies are disabled in the client, session tracking gets failed. In such a case, web containers make use of URL rewriting mechanism to track sessions. In order for this mechanism to work, all URLs in the page should be encoded using the encodeURL() method of HttpServletResponse interface. This method appends a path string of the form sessionid=A81234LKDM3421A098 to the input URL.

Public String encodeURL(String url):- Encodes the specified URL by including the session ID in it; or, if encoding is not needed, returns the URL unchanged. The implementation of this method included the logic to determine whether the session ID needs to be encoded in the URL. For example, if the browser supports cookies, or session tracking is turned off, URL encoding is unnecessary. For robust session tracking, all URLs emitted by a Servlet should be run through this method.