11.1 FUNDAMENTALS OF WEB PROGRAMMING

11.1.1 Review of HTTP

Web Server and Web Client are two separate software, so there should be some common language for communication. HTML is the common language between server and client and stands for Hypertext Markup Language. Web server and client needs a common communication protocol, HTTP (Hypertext Transfer Protocol) is the communication protocol between server and client. HTTP runs on top of TCP/IP communication protocol.

HTTP stands for hypertext transfer protocol. A HTTP Servlet runs under the HTTP protocol. It is important to understanding the HTTP protocol in order to understand server-side programs (servlet, JSP, ASP, PHP, etc) running over the HTTP. In brief, HTTP is a request-response protocol. The client sends a request message to the server. The server, in turn, returns a response message. The messages consist of two parts: header (information about the message) and body (contents). Header provides information about the messages. The data in header is organized in name-value pairs.

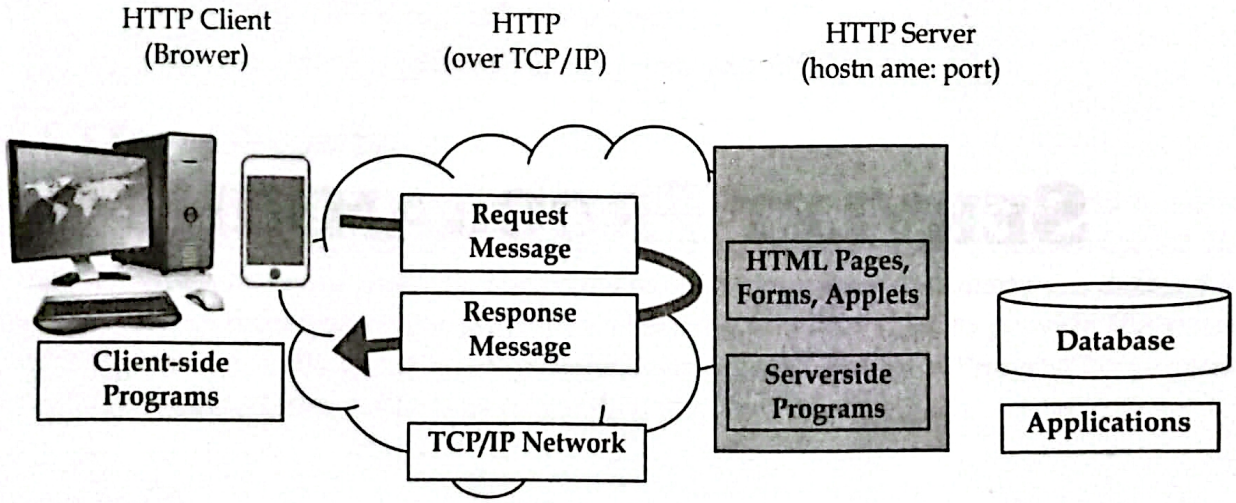


Fig 10.1: HTTP Request-Response

Some of the important parts of HTTP Response are:

- **Status Code** – an integer to indicate whether the request was success or not. Some of the well known status codes are 200 for success, 404 for Not Found and 403 for Access Forbidden.
- **Content Type** – text, html, image, pdf etc. Also known as MIME type
- **Content** – actual data that is rendered by client and shown to user.

11.1.2 GET and POST Request

Two request methods, GET and POST, are available for submitting form data, to be specified in the <form>'s attribute "method=GET | POST". GET and POST performs the same basic function. That is, gather the name-value pairs of the selected input elements, URL-encode, and pack them into a query string. However, in a GET request, the query string is appended behind the

URL, separated by a '?'. Whereas in a POST request, the query string is kept in the request body (and not shown in the URL). The length of query string in a GET request is limited by the maximum length of URL permitted, whereas it is unlimited in a POST request. Thus The "post" method is more robust and secure than "get". Hence, never use the "get" method to pass sensitive information! (password or other sensitive information will be visible in the browser's address bar)

11.2 SERVER SIDE OF THE WEB APPLICATION

Java servlets are the foundation of the Java based server-side technologies such as JSP (Java Server Pages), JSF (Java Server Faces), Struts, Spring, Hibernate, and others, are extensions of the servlet technology.

11.2.1 Web Container

Web container is runtime environment for a web application. The web applications run within web container. Web container may be built-in with web server or sometimes it can be separate system. When a request is made from Client to web server, it passes the request to web container and it's web container job to find the correct resource to handle the request (servlet or JSP) and then use the response from the resource to generate the response and provide it to web server. Then web server sends the response back to the client. Some of the important works done by web container are:

- **Communication Support** – Container provides easy way of communication between web server and the servlets and JSPs. Because of container, we don't need to build a server socket to listen for any request from web server, parse the request and generate response. All these important and complex tasks are done by container and all we need to focus is on our business logic for our applications.
- **Lifecycle and Resource Management** – Container takes care of managing the life cycle of servlet. Container takes care of loading the servlets into memory, initializing servlets, invoking servlet methods and destroying them.
- **Multithreading Support** – Container creates new thread for every request to the servlet and when it's processed the thread dies. So servlets are not initialized for each request and saves time and memory.
- **JSP Support** – JSPs doesn't look like normal java classes and web container provides support for JSP. Every JSP in the application is compiled by container and converted to Servlet and then container manages them like other servlets.
- **Miscellaneous Task** – Web container manages the resource pool, does memory optimizations, run garbage collector, provide security configurations, support for multiple applications, hot deployment and several other tasks behind the scene that makes our life easier.

11.2.2 Structure of Web Applications

A Web application exists as a structured hierarchy of directories. The root of this hierarchy serves as the document root for files that are part of the application. The first step in creating a web application is creating this structure. Each one of these directories should be created from the root directory of the servlet container. An example of a root directory, using Tomcat, would be */jakarta-tomcat-x.0/webapps*. The figure given below shows structure of web applications:

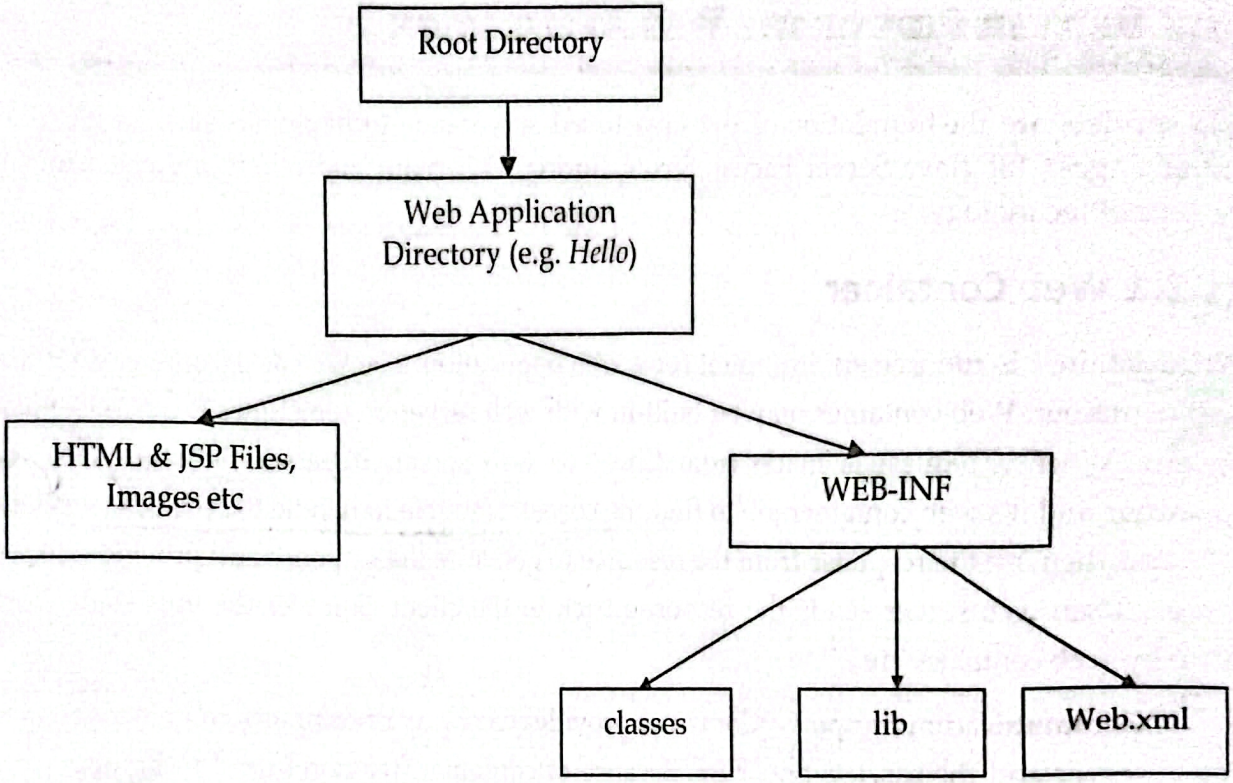


Fig 10.2: Hierarchical Structure of Web Application

In above structure, *classes* is the directory is where servlet and utility classes are located. The *lib* directory contains Java Archive files that the web application depends upon. For example, this is where we would place a JAR file that contained a JDBC driver. And Web.xml is the deployement descriptor.

11.2.3 Deployment Descriptors

The deployment descriptor is an XML file named web.xml located in the *root/application-name/WEB-INF* directory. It describes configuration information for the entire web application. The information that is contained in the deployment descriptor includes the following elements: ServletContext Init Parameters, Session Configuration, Servlet / JSP Definitions, Servlet / JSP Mappings, Welcome File list, Error Pages, Security etc. The following code snippet contains a limited example of a web application deployment descriptor.

```
<web-app>
 <display-name>The OnJava App</display-name>
   <session-timeout>30</session-timeout>
   <servlet>
         <servlet-name>TestServlet</servlet-name>
         <servlet-class>com.onjava.TestServlet</servlet-class>
```

```
        <load-on-startup>1</load-on-startup>
        <init-param>
            <param-name>name</param-name>
           <param-value>value</param-value>
        </init-param>
    </servlet>
</web-app>
```

11.3 SERVLET AND SERVLET TECHNOLOGY

Java servlets are server-side program that runs inside a web server and handles client requests and return dynamic response for each request. The dynamic response could be based on user's input (e.g., search, online shopping, online transaction) with data retrieved from databases or other applications, or time-sensitive data (such as news and stock prices). It acts as a middle layer between requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server (Web server). Following diagram shows the position of Servlets in a Web Application.

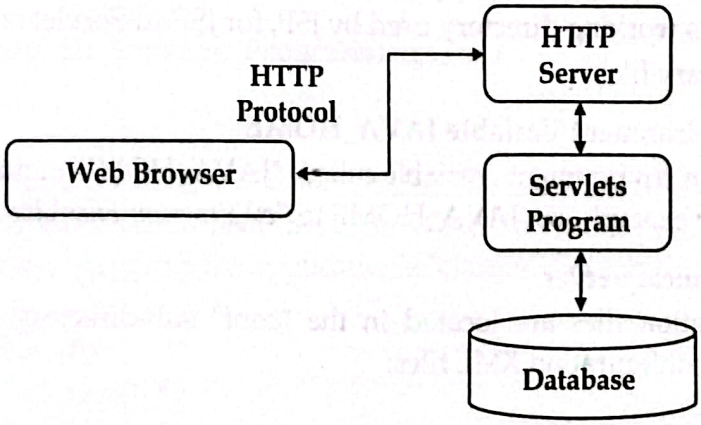


Fig 10.3: Position of Servlets

Servlets perform the following major tasks

- **Read the explicit data sent by the clients**: This includes an HTML form on a Web page or it could also come from an applet.
- **Read the implicit HTTP request data sent by the clients**: This includes cookies, media types and compression schemes the browser understands, and so forth.
- **Process the data and generate the results**: This process may require talking to a database, invoking a Web service, or computing the response directly.
- **Send the explicit data to the clients**: This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- **Send the implicit HTTP response to the clients**: This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

11.4 Environment Setup for Servlets

A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them. Like any other Java program, we need to compile a servlet by using the Java compiler **javac** and after compilation the servlet application, it would be deployed in a configured environment and can be executed. This development environment setup involves following steps:

STEP 1: Download and Install Tomcat

Go to http://tomcat.apache.org and download the file apache-tomcat-x.x.{xx}.zip. Then UNZIP it into a directory of your choice. (Say, C:\tomcat). It contains the following sub-directories:

- **bin**: contains the binaries; and startup script, shutdown script, and other binaries and scripts
- **conf**: contains the system-wide configuration files, such as server.xml, web.xml, context.xml, and tomcat-users.xml
- **lib**: contains the Tomcat's system-wide JAR files, accessible by all webapps
- **logs**: contains Tomcat's log files. We may need to check for error messages here
- **webapps**: contains the webapps to be deployed. we can also place the WAR (Webapp Archive) file for deployment here
- **work**: Tomcat's working directory used by JSP, for JSP-to-Servlet conversion.
- **temp**: Temporary files.

STEP 2: Create an Environment Variable JAVA_HOME

We need to create an environment variable called "JAVA_HOME" and set it to our JDK installed directory. For example, set JAVA_HOME to *"c:\Program Files\Java\jdk1.x.x_{xx}"*

STEP 3: Configure Tomcat Server

The Tomcat configuration files are located in the "conf" sub-directory of Tomcat installed directory. There are 4 configuration XML files:

1. server.xml
2. web.xml
3. context.xml
4. tomcat-users.xml

- **Set the TCP Port Number:** Open the configuration file "server.xml", under the "conf" sub-directory of Tomcat installed directory and set TCP port number. We may choose any number between 1024 and 65535, which is not used by an existing application. Default port number is 8080. We will use default port number.
- **Enabling Directory Listing:** Open the configuration file "web.xml", under the "conf" sub-directory of Tomcat installed directory and enable directory listing by changing "listings" from "false" to "true" for the "default" servlet.
- **Enabling Automatic Reload:** Open the configuration file "context.xml" and add the attribute reloadable="true" to the <Context> element to enable automatic reload after code changes. Locate the <Context> start element, and change it to *<Context reloadable="true">*.

STEP 4: Start Tomcat Server: The Tomcat's executable programs and scripts are kept in the "bin" sub-directory of the Tomcat installed directory. Go to the directory and execute *startup.bat* file. Then start a browser. Issue URL "http://localhost:8080" to access the Tomcat server's welcome page.

STEP 5: Develop and Deploy a WebApp: First of all, choose a *name* for your webapp. Let's call it "myproject". Goto Tomcat's "webapps" sub-directory and create the following directory structure:

1. Under Tomcat's "webapps", create your webapp *root* directory "myproject". We should keep all our HTML files and resources (e.g., HTMLs, CSSs, images, scripts, JSPs) in this directory.
2. Under "myproject", create a sub-directory "WEB-INF", directory name is case sensitive. This is where you keep your application's web descriptor file "web.xml".
3. Under "WEB-INF", create a sub-sub-directory "classes". This is where you keep all the Java classes such as servlet class-files.

We should restart our Tomcat server to pick up the newly created webapp. Issue the following URL to access the web application "myproject": *http://localhost:8080/myproject*. It should display directory listing of the web application.

Create the following HTML page and save as "Hello.html" in application's root directory "myproject".

```
<html>
 <head>
<title>First WebApp</title>
</head>
  <body>
      <h1>Well Come to Servlet Programming</h1>
</body>
</html>
```

We can browse this page by issuing this URL: *http://localhost:8080/hello/Hello.html*.

STEP 6: Write a "Hello-world" Java Servlet: Write the following source codes called "TestServlet.java" and save it under the application's "classes" directory.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class TestServlet extends HttpServlet

public void doGet(HttpServletRequest request, HttpServletResponse response)
throws IOException, ServletException
      {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          out.println("<html>");
          out.println("<head>");
          out.println("<title>Servlet Test</title>");
          out.println("</head>");
          out.println("<body>");
          out.println("<h1>This is From Servlet</h1>");
          out.println("</body>");
          out.println("</html>");
      }
}
```

We need the Servlet API to compile the servlet. Servlet API is NOT part of JDK. Tomcat provides Servlet API in <tomcat-home>/lib/servlet-api.jar. We need to include this JAR file in the CLASSPATH environment variable. Once the compilation succeeds, make a copy of web.xml file in *"webapps\hello\WEB-INF"* and make following entries in the file.

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

Restart the Tomcat server to refresh the "web.xml" file and Invoke the Servlet by issuing the URL *http://localhost:8080/myproject/TestServlet*.

11.5 Servlet Life Cycle (Steps for Writing a Servlet)

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the stages followed by a servlet lifecycle:

- Loaded and Instantiated
- Initialized by calling the **init ()** method
- Process a client's request by invoking **service()** method
- Terminated by calling the **destroy()** method
- Garbage collected by the JVM.

Three steps for writing servlets

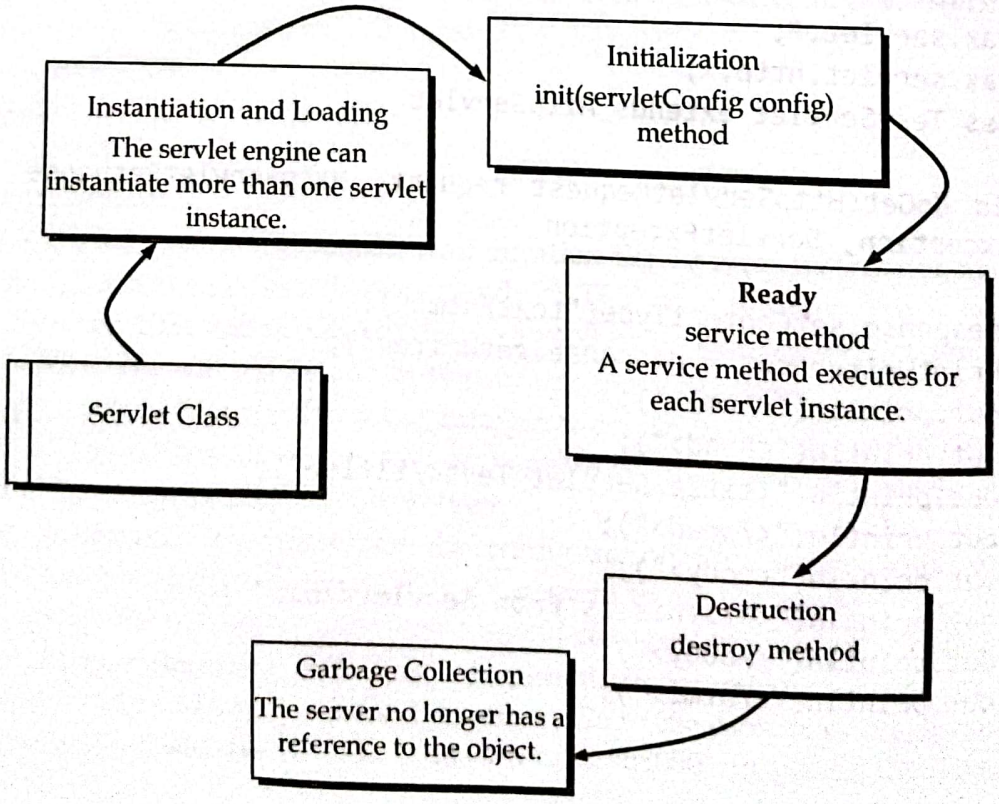


Fig.10.4 Servlet life cycle.

1. **Loaded and Instantiated**
   The class loader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container. Then it creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.
2. **Initialized by calling the init () method**
   The web container calls the **init()** method only once after creating the servlet instance. The init method is used to initialize the servlet. Syntax of the init method is given below:
   public void init(ServletConfig config) throws ServletException
3. **Process a client's request by invoking service() method**
   The web container calls the service method each time when request for the servlet is received. After servlet is initialized, it calls the service method. The syntax of the service method of the Servlet interface is given below:
   public void service(ServletRequest request, ServletResponse response)
   throws ServletException, IOException
   The doGet() and doPost() are most frequently used methods within each service request.
4. **Terminated by calling the destroy() method**
   The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:
   public void destroy()
5. **Garbage collected by the JVM**
   Once the servlet is destroyed, garbage collector component of JVM is responsible collecting the garbage's.

11.6 SERVLETS APIs

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet API. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

There are many interfaces in javax.servlet package. Some of them are: *Servlet, ServletRequest, ServletResponse, ServletConfig, ServletContext etc.* Besides this, there are many classes in javax.servlet package. Some of them are: *GenericServlet, ServletInputStream, ServletOutputStream, ServletRequestWrapper, ServletResponseWrapper, ServletException, UnavailableException etc.*

On the other hand, there are many interfaces in javax.servlet.http package. They are as follows: *HttpServletRequest, HttpServletResponse, HttpSession* etc. In addition to this, there are many classes in javax.servlet.http package. They are as follows: *HttpServlet, Cookie, HttpServletRequestWrapper, HttpServletResponseWrapper, HttpSessionEvent* etc.

11.6.1 Servlet Interface

Servlet interface provides common behavior to all the servlets. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods that are used to get servlet information and servlet configurations.

Example

```
import java.io.*;
import javax.servlet.*;
public class ServletInt implements Servlet
{
   ServletConfig config=null;
   public void init(ServletConfig config)
   {
      this.config=config;
      System.out.println("servlet is initialized");
   }
public  void  service(ServletRequest  req,ServletResponse  res)  throws
IOException, ServletException
   {
      res.setContentType("text/html");
      PrintWriter out=res.getWriter();
      out.print("<html><body>");
      out.print("<b>Hello simple servlet</b>");
      out.print("</body></html>");
   }
   public void destroy(){System.out.println("servlet is destroyed");}
   public ServletConfig getServletConfig(){return config;}
   public String getServletInfo(){return "Implementing Servlet Interface";}
}
```

Output

When we execute this program we will see the message *"Hello simple servlet"* in web browser.

11.6.2 GenericServlet Class

GenericServlet class implements Servlet, ServletConfig and Serializable interfaces. It provides the implementaion of all the methods of these interfaces except the service method. GenericServlet class can handle any type of request so it is protocol-independent. We can create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Example

```
import java.io.*;
import javax.servlet.*;
public class GenServlet extends GenericServlet
{
public void service(ServletRequest req,ServletResponse res)  throws
IOException,ServletException
```

```
  {
     res.setContentType("text/html");
     PrintWriter out=res.getWriter();
     out.print("<html><body>");
     out.print("<b>Gello Generic Servlet</b>");
     out.print("</body></html>");

  }
}
```

Output

When we execute this program we will see the message *"Hello Generic Servlet"* in web browser.

11.6.3 HttpServlet Class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost etc. doGet() method handles the GET request. And doPost() handles the POST request. Get request is the default request.

Example

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet
{
public void doGet(HttpServletRequest req,HttpServletResponse res)  throws
ServletException,IOException
  {
     res.setContentType("text/html");//setting the content type
     PrintWriter pw=res.getWriter();//get the stream to write the data

     //writing html in the stream
     pw.println("<html><body>");
     pw.println("Welcome to servlet");
     pw.println("</body></html>");
     pw.close();//closing the stream
  }
}
```

Output

When we execute this program we will see the message *"Welcome to Servlet"* in web browser.

11.7 PROCESSING HTML FORM DATA USING SERVLETS

We need to pass some information from your browser to web server and ultimately to our backend program. The browser uses two methods to pass this information to web server. These methods are GET Method and POST Method. The GET method sends the encoded user information appended to the page request. The page and the encoded information are separated by the character as follows:

http://www.test.com/hello?key1=value1&key2=value2

The GET method is the default method to pass information from browser to web server and it produces a long string that appears in browser's location box. Never use the GET method if we have password or other sensitive information to pass to the server. The GET method has size limitation: only 1024 characters can be in a request string. This information is passed using QUERY_STRING header and will be accessible through QUERY_STRING environment variable and Servlet handles this type of requests using **doGet()** method.

A generally more reliable method of passing information to a backend program is the POST method. This packages the information in exactly the same way as GET methods, but instead of sending it as a text string after a '?' in the URL it sends it as a separate message. This message comes to the backend program in the form of the standard input which we can parse and use for your processing. Servlet handles this type of requests using **doPost()** method.

Servlets reads parameters in a form using the following methods depending on the situation:

- ✓ **getParameter():** We call request.getParameter() method to get the value of a form parameter.
- ✓ **getParameterValues():** Call this method if the parameter appears more than once and returns multiple values, for example checkbox.
- ✓ **getParameterNames():** Call this method if we want a complete list of all parameters in the current request.

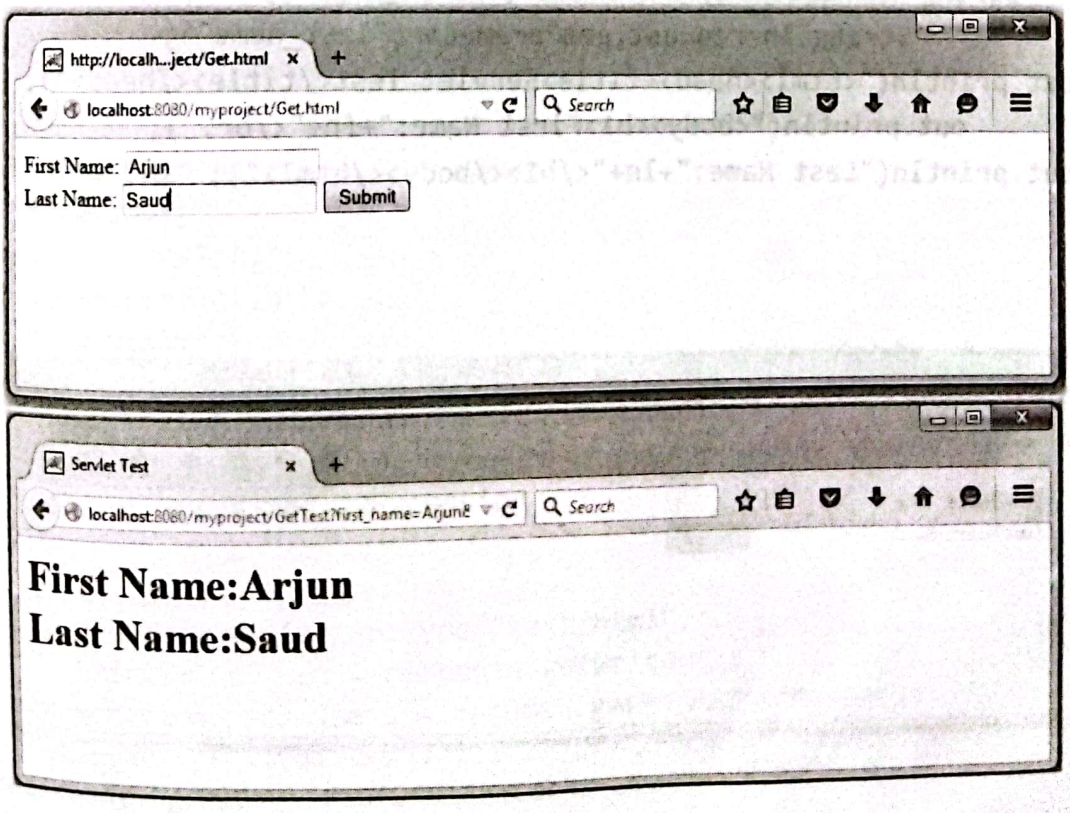
Example

//Reading form data with GET Method

```
//Get.html file
<html>
<body>
<form action="GetTest" method="GET">
   First Name: <input type="text" name="first_name">
   <br />
   Last Name: <input type="text" name="last_name" />
   <input type="submit" value="Submit" />
</form>
</body>
```

```
</html>

//GetTest.java File
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class GetTest extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
  {
        response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       String fn=request.getParameter("first_name");
             String ln=request.getParameter("last_name");
    out.println("<html><head><title>Servlet Test</title></head>");
         out.println("<body><h1>First Name:"+fn+"</br>");
    out.println("Last Name:"+ln+"</h1></body></html>");
  }
}
```

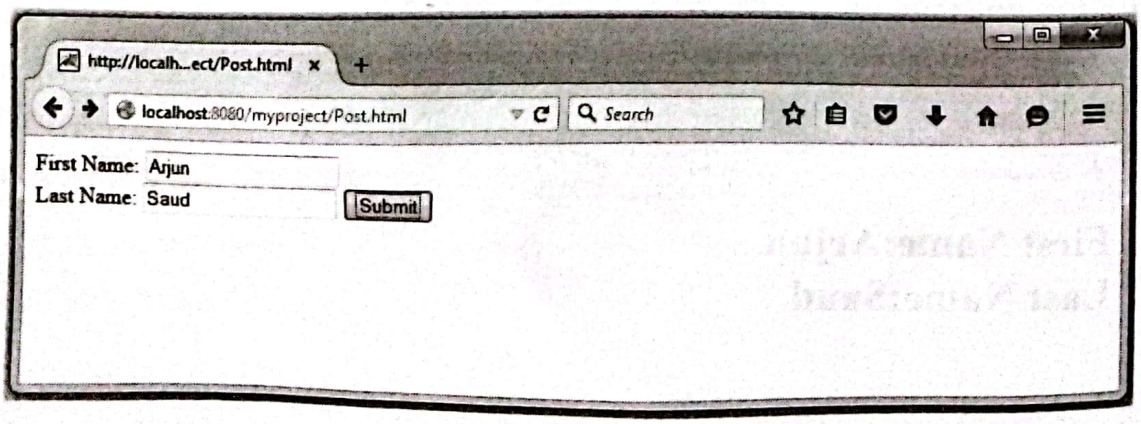
Output

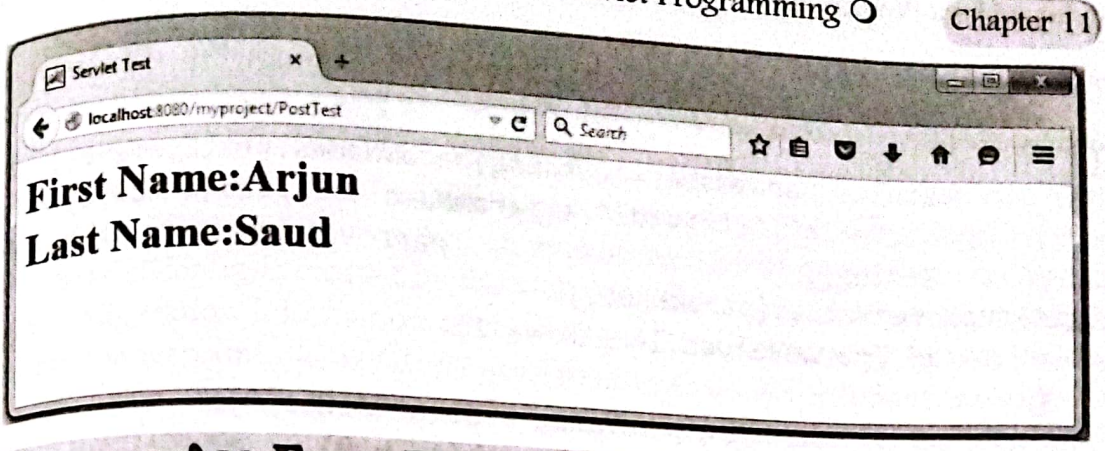


Example

```
//Reading form data with Post Method
//Post.html file
<html>
<body>
<form action="PostTest" method="POST">
   First Name: <input type="text" name="first_name">
   <br />
   Last Name: <input type="text" name="last_name" />
   <input type="submit" value="Submit" />
</form>
</body>
</html>

//PostTest.java File
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class PostTest extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
   {
          response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         String fn=request.getParameter("first_name");
               String ln=request.getParameter("last_name");
     out.println("<html><head><title>Servlet Test</title></head>");
          out.println("<body><h1>First Name:"+fn+"</br>");
     out.println("Last Name:"+ln+"</h1></body></html>");
   }
}
```

Output





11.8 READING ALL FORM PARAMETERS

Following is the generic example which uses *getParameterNames()* method of *HttpServletRequest* to read all the available form parameters. This method returns an Enumeration that contains the parameter names in an unspecified order. Once we have an Enumeration, we can loop down the Enumeration in the standard manner, using *hasMoreElements()* method to determine when to stop and using *nextElement()* method to get each parameter name.

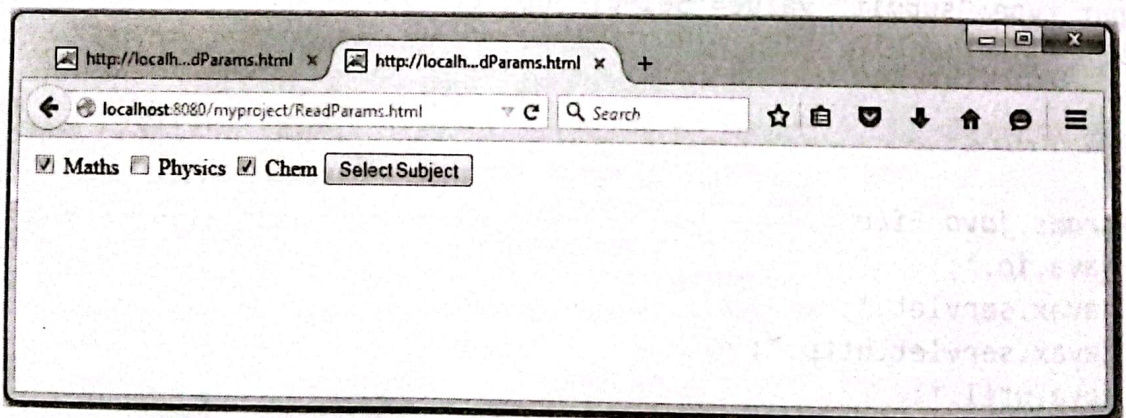
Example

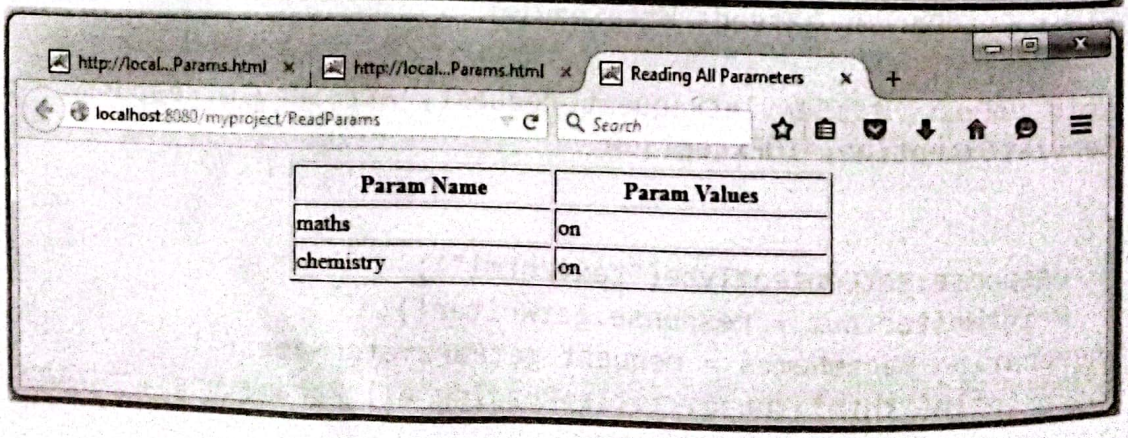
```
//ReadParams.html file
<html>
<body>
<form action="ReadParams" method="POST" target="_blank">
  <input type="checkbox" name="maths" checked="checked" /> Maths
  <input type="checkbox" name="physics"  /> Physics
  <input type="checkbox" name="chemistry" checked="checked" /> Chem
  <input type="submit" value="Select Subject" />
</form>
</body>
</html>
```

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

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
    Enumeration paramNames = request.getParameterNames();
    out.println("<html><head><title>Reading All Parameters </title>
</head><body bgcolor=\"#f0f0f0\"><table width=\"50%\"
```

```
border=\"1\"align=\"center\"><tr><th>Param Name</th><th>Param Values</th></tr>");
        while(paramNames.hasMoreElements())
      {
              String paramName = (String)paramNames.nextElement();
              out.print("<tr><td>" + paramName + "</td>\n<td>");
              String[]                          paramValues
request.getParameterValues(paramName);                                    =
              if (paramValues.length == 1)
        {
                    String paramValue = paramValues[0];
                    if (paramValue.length() == 0)
                          out.println("<i>No Value</i>");
                    else
                          out.println(paramValue);
              }
        else
        {
                                out.println("<ul>");
                    for(int i=0; i < paramValues.length; i++)
                    out.println("<li>" + paramValues[i]);
              }
              out.println("</ul>");
          }
    out.println("</tr></table></body></html>");
    }
}
```

Output





11.9 READING INITIALIZATION PARAMETERS

The servlet container provides the initialization parameters for a servlet within the configuration object. The configuration object provides a *getInitParameter()* function that takes a string name as argument and returns the contents of the initialization parameter by that name. An object of ServletConfig is created by the web container for each servlet. This object can be used to get configuration information from web.xml file. If the configuration information is modified from the web.xml file, we don't need to change the servlet. So it is easier to manage the web application if any specific content is modified from time to time. Methods of ServletConfig interface are:

- ✓ **public String getInitParameter(String name):**Returns the parameter value for the specified parameter name.
- ✓ **public Enumeration getInitParameterNames():**Returns an enumeration of all the initialization parameter names.
- ✓ **public String getServletName():**Returns the name of the servlet.
- ✓ **public ServletContext getServletContext():**Returns an object of ServletContext.

Syntax to provide the initialization parameter for a servlet

```
<web-app>
  <servlet>
    ......
    <init-param>
      <param-name>parametername</param-name>
      <param-value>parametervalue</param-value>
    </init-param>
    ......
  </servlet>
</web-app>
```

```
//Example: Reading Initialization Parameters
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletDemo extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
  {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        ServletConfig config=getServletConfig();
        String driver=config.getInitParameter("driver");
        out.print("Driver is: "+driver);
      out.close();
      }
}
```

Output

When we execute this program we will see the message *"Driver is: sun.jdbc.odbc.JdbcOdbcDriver"* in web browser.

11.10 COOKIES IN SERVLET

A cookie is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number. There are three steps involved in identifying returning users:

- ✓ Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- ✓ Browser stores this information on local machine for future use.
- ✓ When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

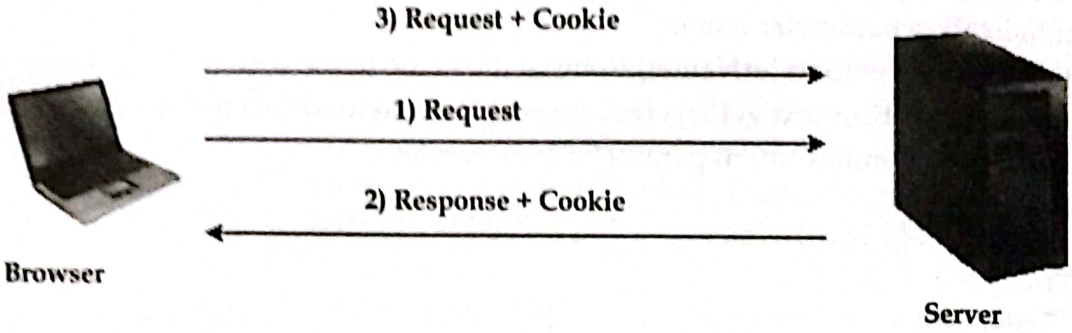


Fig. 10.5 Cookie handling.

Advantages of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	*Constructs a cookie*
Cookie(String name, String val)	*Constructs a cookie with a specified name and value*

Useful Methods of Cookie class

Method	Description
setMaxAge(int expiry)	*Sets the maximum age of the cookie in seconds*
String getName()	*Returns the name of the cookie*
String getValue()	*Returns the value of the cookie*
void setName(String name)	*Changes the name of the cookie*
void setValue(String value)	*Changes the value of the cookie*

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

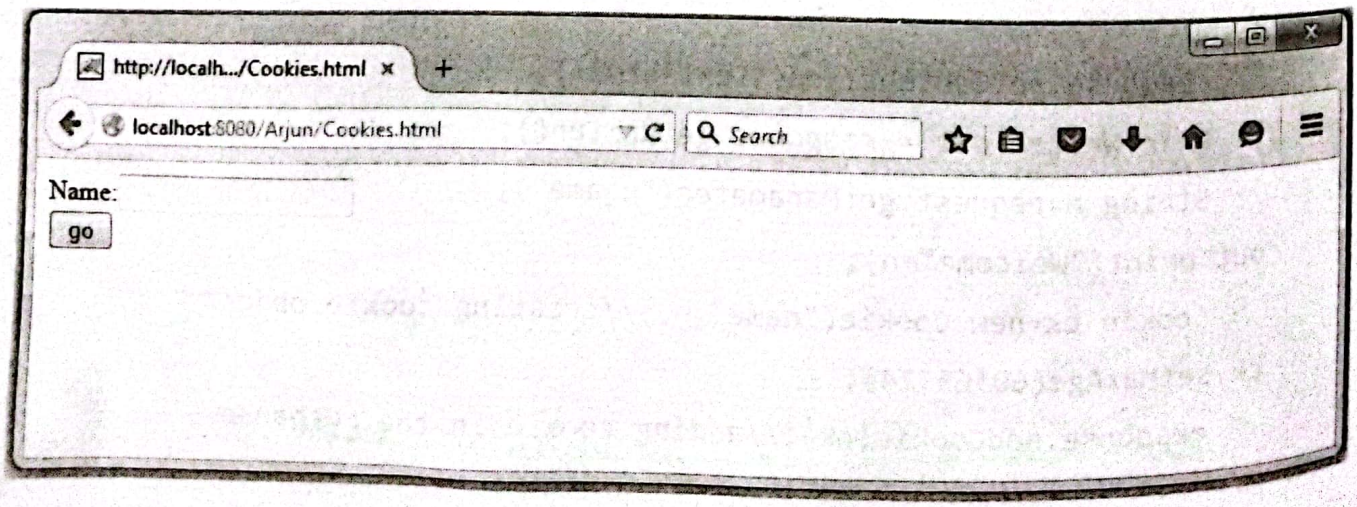
- **public void addCookie(Cookie ck):** method of HttpServletResponse interface is used to add cookie in response object.
- **public Cookie[] getCookies():** method of HttpServletRequest interface is used to return all the cookies from the browser.

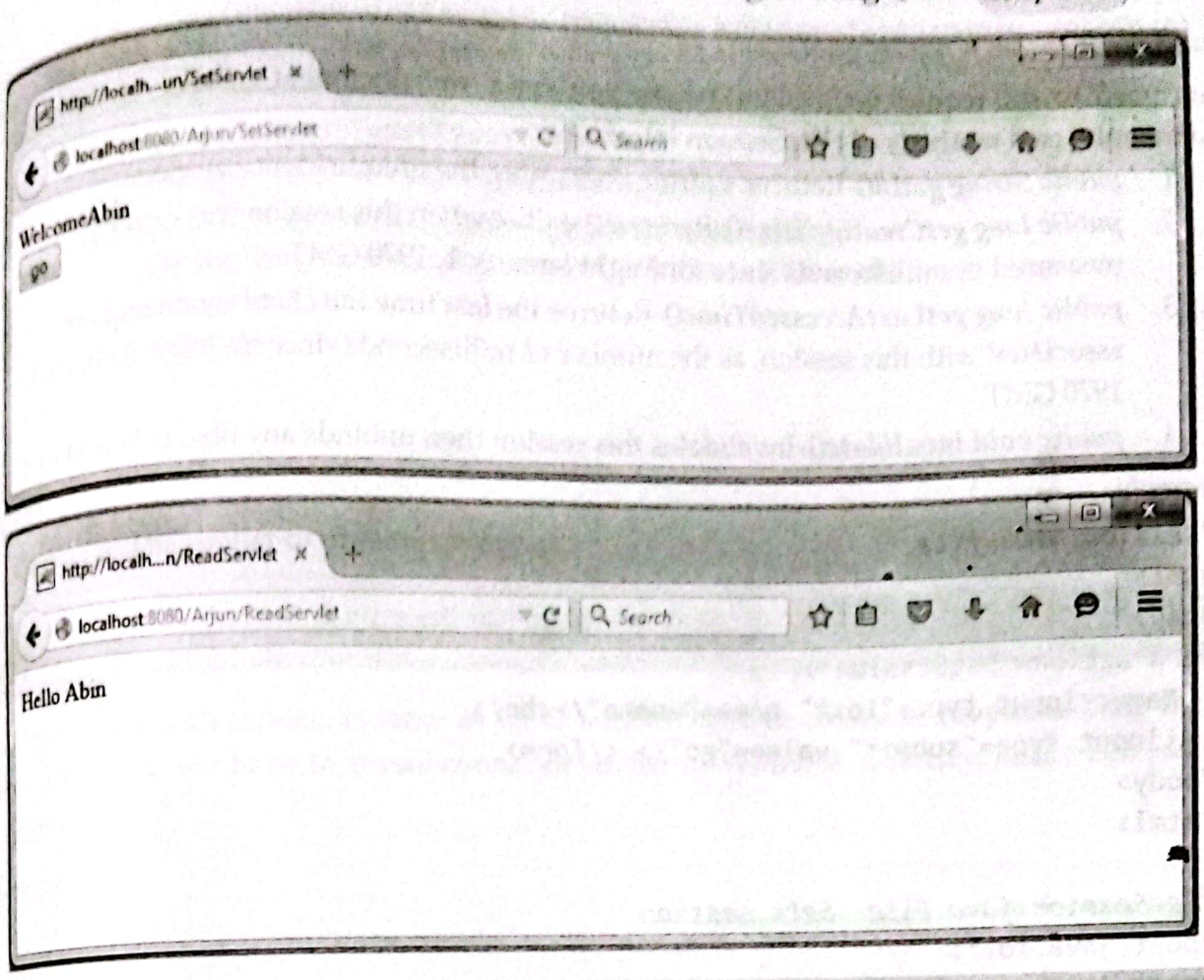
Example

```
//cookies.html file
<html>
<body>
<form action="SetServlet" method="POST">
  Name:<input type="text" name="uname"/><br/>
  <input type="submit" value="go"/> </form>
</body>
</html>

//SetServlet.java File–Sets cookies
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SetServlet extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
  {

      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      String n=request.getParameter("uname");
    out.print("Welcome"+n);
      Cookie ck=new Cookie("name",n);//creating cookie object
    ck.setMaxAge(60*60*24);
      response.addCookie(ck);//adding cookie in the response
```

```
        //creating submit button
            out.print("<form action='ReadServlet' method='POST'>");
            out.print("<input type=submit value='go'>");
            out.print("</form>");
            out.close();
    }
}

//ReadServlet.java File----Reads Cookies
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ReadServlet extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
   {
      response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         Cookie[ ] ck=request.getCookies();
      if(ck!=null)
         out.print("Hello "+ck[0].getValue());
         out.close();
         }
}
```

Output





11.11 SESSION TRACKING IN SERVLETS

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request. Thus, each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user. **Session** simply means a particular interval of time.

Session Tracking is a way to maintain state (data) of an user. It is also known as **session management** in servlet. There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

Cookies is already discussed and we will discuss only Session tracking using HttpSession interface here.

HttpSession Object

Servlt provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user. The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. We can get HttpSession object by calling the public method *getSession()* of HttpServletRequest, as below:

HttpSession session = request.getSession();

You need to call request.getSession() before you send any document content to the client.

Commonly used methods of HttpSession interface

1. ***public String getId()***-Returns a string containing the unique identifier value.
2. ***public long getCreationTime()***-Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. ***public long getLastAccessedTime()***-Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. ***public void invalidate()***-Invalidates this session then unbinds any objects bound to it.

Example

```
//session.html file
<html>
<body>
<form action="SetSession">
   Name:<input type="text" name="uname"/><br/>
   <input type="submit" value="go"/> </form>
</body>
</html>

//SetSession.java File- Sets session
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SetSession extends HttpServlet
{
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
   {
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         String n=request.getParameter("uname");
      out.print("Welcome"+n);
         HttpSession session=request.getSession();
            session.setAttribute("uname",n);
            out.print("<a href='ReadServlet'>Visit Here</a>");
   }
}

//ReadSession.java File- Reads session
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ReadSession extends HttpServlet
{
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
  {
     response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        HttpSession session=request.getSession(false);
           String n=(String)session.getAttribute("uname");
           out.print("Hello "+n);
        out.close();
        }
}
```

Output

Output will be similar to previous program

11.12 DATABASE ACCESS WITH SERVLETS

Database access with servlets is same as JDBC that we have discussed in chapter 4. Only the difference is that we have to mysql-connector.jar file in WEB-INF directory inside our root directory.

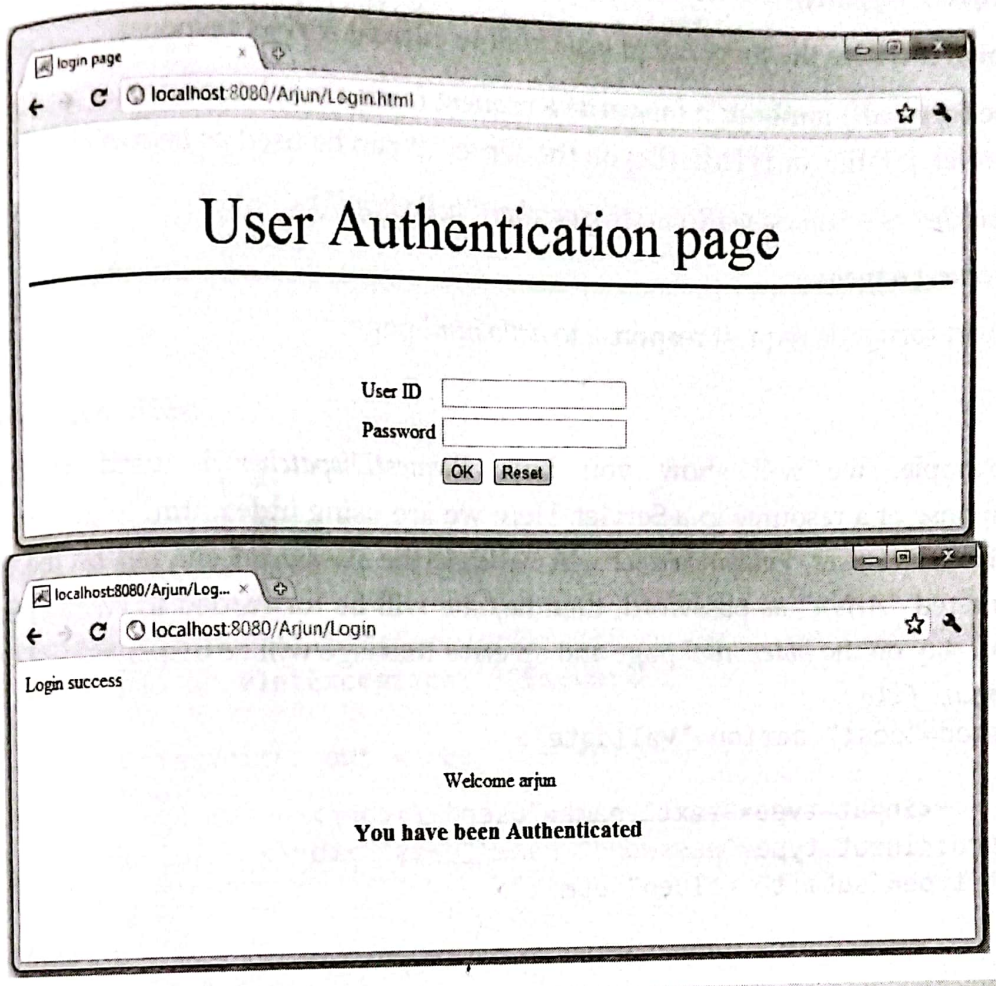
Example

```
//login.html file
<html>
<head>
   <Title> login page </Title>
</head>
<body>
   <br><br>
   <font color=blue size=12> <center> User Authentication page
</center></font>
   <hr color=red size=3>
   <br><br> <br>
   <form action="Login" method="Post">
      <center>
      <table>
      <tr> <td> User ID </td> <td> <input type=text name=txtid></td></tr>
      <tr> <td> Password </td> <td>
<input type=password name=txtpass></td></tr>
      <tr> <td colspan=2 align=center>
<input type=submit value=OK>
      <input type=reset></td></tr>
```

```
        </table>
    </form>
</body>
</html>

//Login.java File
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Login extends HttpServlet
{
   Connection con=null;
public void doPost(HttpServletRequest req, HttpServletResponse resp) throws
IOException, ServletException
   {
      PrintWriter out=resp.getWriter();
      resp.setContentType("text/html");
      String url="jdbc:mysql://localhost/Test";
      try
      {
         String n=req.getParameter("txtid");
         String p=req.getParameter("txtpass");
         Class.forName("com.mysql.jdbc.Driver");
         con=DriverManager.getConnection(url,"root","arjun");
         Statement stmt=con.createStatement();
String sql="select * from login where Name='"+n+"' and Password='"+p+"'";
         ResultSet rs=stmt.executeQuery(sql);
if(!rs.next())
         {
out.println("<html><head></Title> Login error </Title> </head><body>");
out.println("<br><br><br><br><b>Unknown User </b> <br><br>:");
               out.println("<h3>Access denied </h3></body></html>");
         }
         else
         {
out.println("<html><head></Title>  Login  success  </Title>  </head>
<body><center>");
               out.println("<br><br><br> Welcome  ");
               out.println(n+"</b><br>");
out.println("<h3>You  have  been  Authenticated  </h3>  </center>
</body></html>");
         }
         con.close();
   }
```

```
catch(SQLException se)
{out.println("Error!!!!"+se);}
catch(ClassNotFoundException cne)
{out.println("Error!!!!"+cne);}
}
```

Output



1.13 REQUEST DISPATCHER

While developing web applications we need to distribute the request processing and response generation to multiple servlet objects. So we need to dispatch requests from one component to another component. This can be done by using *RequestDispatcher* interface is implemented by servlet container to dispatch or to pass the request to a web resource such as Servlet, HTML page or JSP page. To dispatch the request from Servlet or JSP to web resource using *RequestDispatcher* we need to perform following steps:

- Get a *RequestDispatcher* object reference
- Using *include ()* and *forward ()* methods of *RequestDispatcher.*

We can get object of *RequestDispatcher* by using *getRequestDispatcher()* method of *ServletRequest* as below:

RequestDispatcher rs = request.getRequestDispatcher(resource_name);

12.1 Basic of Java Server Pages

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. It can be thought of as an extension to servlet because it provides more functionality than servlet. A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than servlet because we can separate designing and development.

Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands. JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages and sharing information between requests, pages etc.

12.1.1 Advantages of JSP

Following is the list of other advantages of using Java Server Pages over other technologies:

JSP vs ASP

The advantages of JSP are twofold

- The dynamic part is written in Java, not Visual Basic or other MS specific language, so it is more powerful and easier to use.
- It is portable to other operating systems and non-Microsoft Web servers.

JSP vs Servlets

There are many advantages of JSP over servlet. They are as follows:

- **Extension to Servlet:** JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, which makes JSP development easy.
- **Easy to maintain:** JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
- **Fast Development:** If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
- **Less code than Servlet:** In JSP, we can use a lot of tags such as action tags, custom tags etc. that reduces the code.

12.1.2 JSP Access Model

When a user goes to a web site that is developed by using JSP and tries to access JSP page by using a web browser, following steps are performed to process the request and return result to the user:

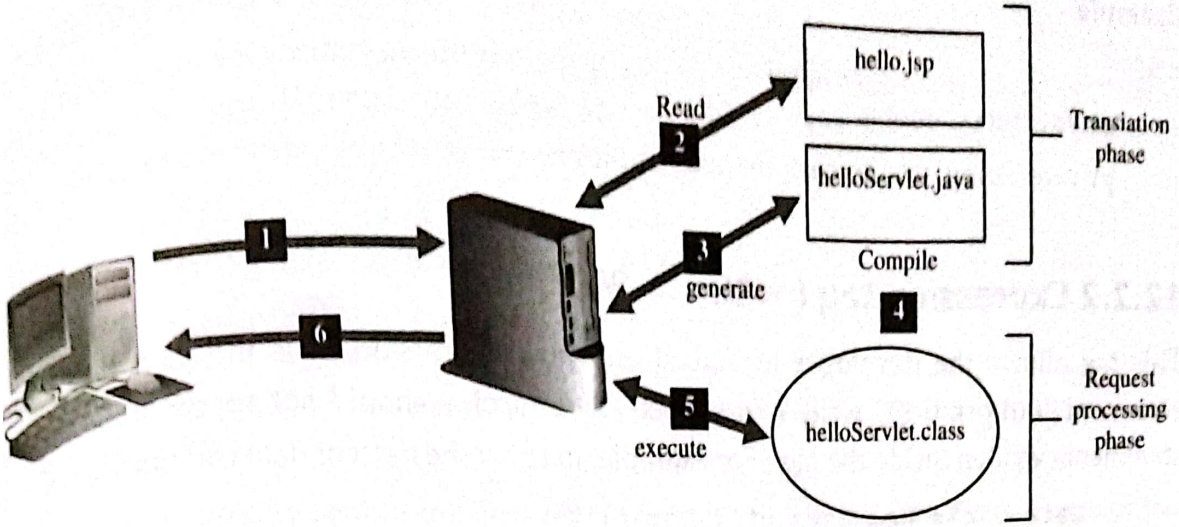


Fig. 11.1 JSP Architecture.

User sends JSP request to the Web server through web browser to the web server with JSP engine.

The Web server recognizes that the file required is JSP file, therefore passes the JSP file to the JSP Engine.

If the JSP file has been called the first time, the JSP file is parsed, otherwise go to step 5. The JSP engine generates a special servlet file from the JSP file.

The servlet file is compiled into a executable class file. JSP engine then forwards the original request to a servlet engine and the servlet is instantiated.

5. The servlet engine loads the servlet class and executes it by calling init and service methods. During execution, the servlet produces an output in HTML format, which is passed to an HTTP response.

6. The web server forwards the HTTP response to browser in terms of static HTML content. Finally web browser handles the dynamically generated HTML page inside the HTTP response and results are displayed on the web browser.

Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet? If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that servlet file is not generated. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

12.2 JSP SYNTAX

There are five main tags that all together form JSP syntax. These tags are: Declaration tags, Expression tags, Directive tags, Scriptlet tags, and Action tags.

12.2.1 Declaration tag (<%! %>)

This tag allows the developer to declare variables or methods. Before the declaration we must have <%! and at the end of the declaration, we must have %>. Statements placed in this tag must end with a semicolon. Declarations do not generate output so are used with JSP expressions or scriptlets.

Example

```
<%!
    private int counter = 0 ;
    private String getAccount( int accountNo) ;
%>
```

12.2.2 Expression tag (<%= %>)

This tag allows the developer to embed any Java expression. It is the short hand form of statement "out.println()" written in servlets. A semicolon should not appear at the end of the statements wriiten inside the tag. For example, to show the current date and time, we can write

```
Date : <%= new java.util.Date() %>
```

12.2.3 Directive tag (<%@ directive ... %>)

It is a JSP directive that gives special information about the page to the JSP Engine. It usually has the following form: <%@ directive attribute="value" %>. There are three main types of directives:

- Page Directive - processing information for this page.
- Include Directive- files to be included.
- Tag library Directive- tag library to be used in this page.

Directives do not produce any visible output when the page is requested but change the way the JSP Engine processes the page. For example, we can make session data unavailable to a page by setting a page directive (session) to false.

Page directive: This directive has 11 optional attributes that provide the JSP Engine with special processing information. Some of the attributes are:

```
<%@ page language = "java" %>
<%@ page extends = "com.taglib… %>
<%@ page import = "java.util.*" %>
```

Include Directive: It allows a JSP developer to include contents of a file inside another. File is included during translation phase. Examples are given below:

```
<%@ include file = "include/privacy.html" %>
<%@ include file = "navigation.jsp" %>
```

The first example includes the *privacy.html* file residing in include directory into the JSP page and the second example includes *navigation.jsp* file in current directory into the JSP page

Tag Lib directive: A tag lib is a collection of custom tags that can be used by web developers in the current. It uses following syntax:

```
<%@ taglib uri = "tag library URI" prefix = "tag Prefix" %>
```

Example

```
<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
```

12.2.4 Scriptlet tag (<% ... %>)

In JSP, java code can be written inside the jsp page using the scriptlet tag. Code is written between <% and %> tags. This code can access any variable or bean declared. For example, to print a variable, we can write scriptlet as below:

```
<%
        String username = "Suman" ;
        out.println ( username ) ;
%>
```

12.2.5 Action tag

The action tags are used to control the flow between pages and to use Java Bean. Some of the action tags supported by JSP are: jsp:forward, jsp:include, jsp:param etc. An example of action tag that forwards to another JSP page is given below:

```
<jsp:forward page="printdate.jsp" />
```

12.2.6 JSP Comment

JSP comments marks text or statements that the JSP container should ignore. A JSP comment is useful when we want to hide or "comment out" part of our JSP page. Following is the syntax of JSP comments:

```
<%-- This is JSP comment --%>
```

12.3 JSP Implicit Objects

There are several objects that are automatically available in JSP called implicit objects. These objects are available to developers in each page and developer can call them directly without being explicitly declared. JSP Implicit Objects are also called pre-defined variables. There are nine implicit objects defined in JSP container. These objects are described below:

- **out object–** This object is used for writing any data to the buffer. It is the object of PrintWriter class.
- **request object-** This is the object of HttpServletRequest class. This object is normally used in looking up parameter values and cookies.
- **response object-** This is the object of HttpServletResponse class. Through this object the JSP programmer can add new cookies or date stamps, HTTP status codes etc.
- **config object-** The config object is an instantiation of ServletConfig class. This object can be used to get initialization parameter for a particular JSP page.
- **exception object-** exception is an implicit object of Throwable class. This object can be used to print the exception.
- **Application object-** It is an instance of ServletContext class and is created only once by the web container when application or project is deployed on the server. This object can be used to get initialization parameter from configuration file (web.xml).
- **Session object-** session is an implicit object of type HttpSession. The Java developer can use this object to set,get or remove attribute or to get session information.

- **Page object-** page is an implicit object of type Object class. This object is assigned to the reference of auto generated servlet class.
- **pageContext object-** pageContext is an implicit object of type PageContext class. The pageContext object can be used to set, get or remove attribute from object scopes.

12.4 Scope of JSP Objects

The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object. Every object created in a JSP page will have a scope. Object scope in JSP is segregated into four parts and they are page, request, session and application.

- **Page Scope-** page scope means, the JSP object can be accessed only from within the same page where it was created. JSP implicit objects out, exception, response, pageContext, config and page have page scope.

```
//Example of JSP Page Scope
<jsp:useBean id="employee" class="EmployeeBean" scope="page" />
```

- **Request Scope-** A JSP object created using the request scope can be accessed from any pages that serves that request. More than one page can serve a single request. Implicit object request has the request scope.

```
//Example of JSP Request Scope
<jsp:useBean id="employee" class="EmployeeBean" scope="request" />
```

- **Session Scope-** session scope means, the JSP object is accessible from pages that belong to the same session from where it was created. Implicit object session has the session scope.

```
//Example of JSP Session Scope
<jsp:useBean id="employee" class="EmployeeBean" scope="session" />
```

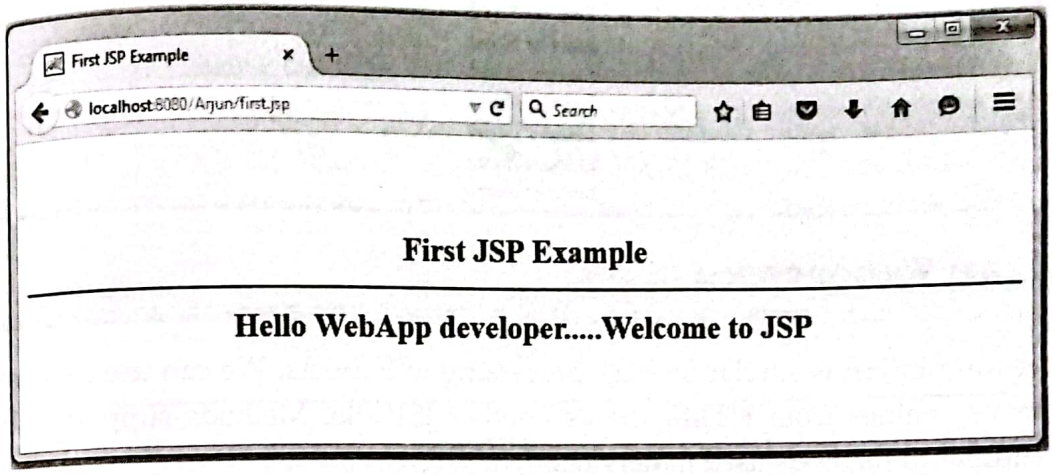
- **Application Scope-** A JSP object created using the application scope can be accessed from any pages across the application. The. Implicit object application has the application scope.

```
//Example of JSP application Scope
<jsp:useBean id="employee" class="EmployeeBean" scope="application" />
```

```
//Example---First JSP Example---first.jsp File
<html>
<head>
<title>First JSP Example</title>
</head>
<body>
<font size=3>
<br><br><center><h2>
<%-- Display by using expression tag --%>
<%= "First JSP Example" %>
<hr color=blue size=3>
```

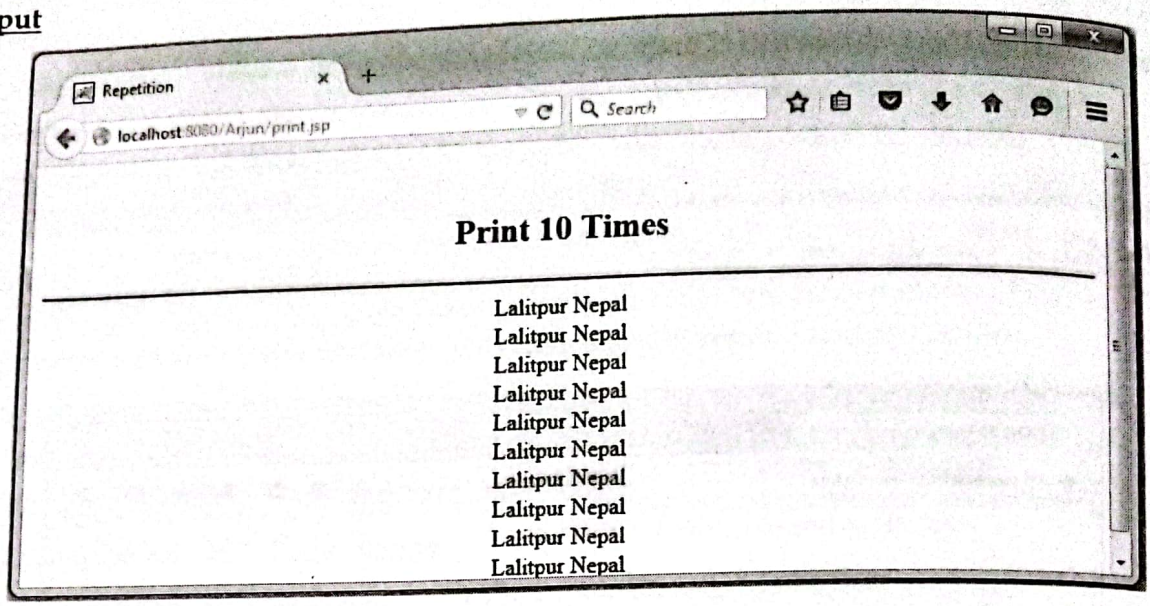
```
<%-- Display by using scriptlet tag --%>
<%
out.println("Hello WebApp developer.....Welcome to JSP");
%>
<h2>
<center>
</font>
</body>
</html>
```

Output




```
//Second Example-Printing Message 10 times
//print.jsp File
<html>
<head>
<title>Repetition</title>
</head>
<body>
<font size=3>
<br><center><h2>
<%-- Display by using expression tag --%>
<%= "Print 10 Times" %>
</h2>
<hr color=blue size=3>
<%-- Display by using scriptlet tag --%>
<%
   for (int i=0;i<10;i++)
   out.println("Lalitpur Nepal<br>");
%>
<center>
</font>
</body>
</html>
```

Output

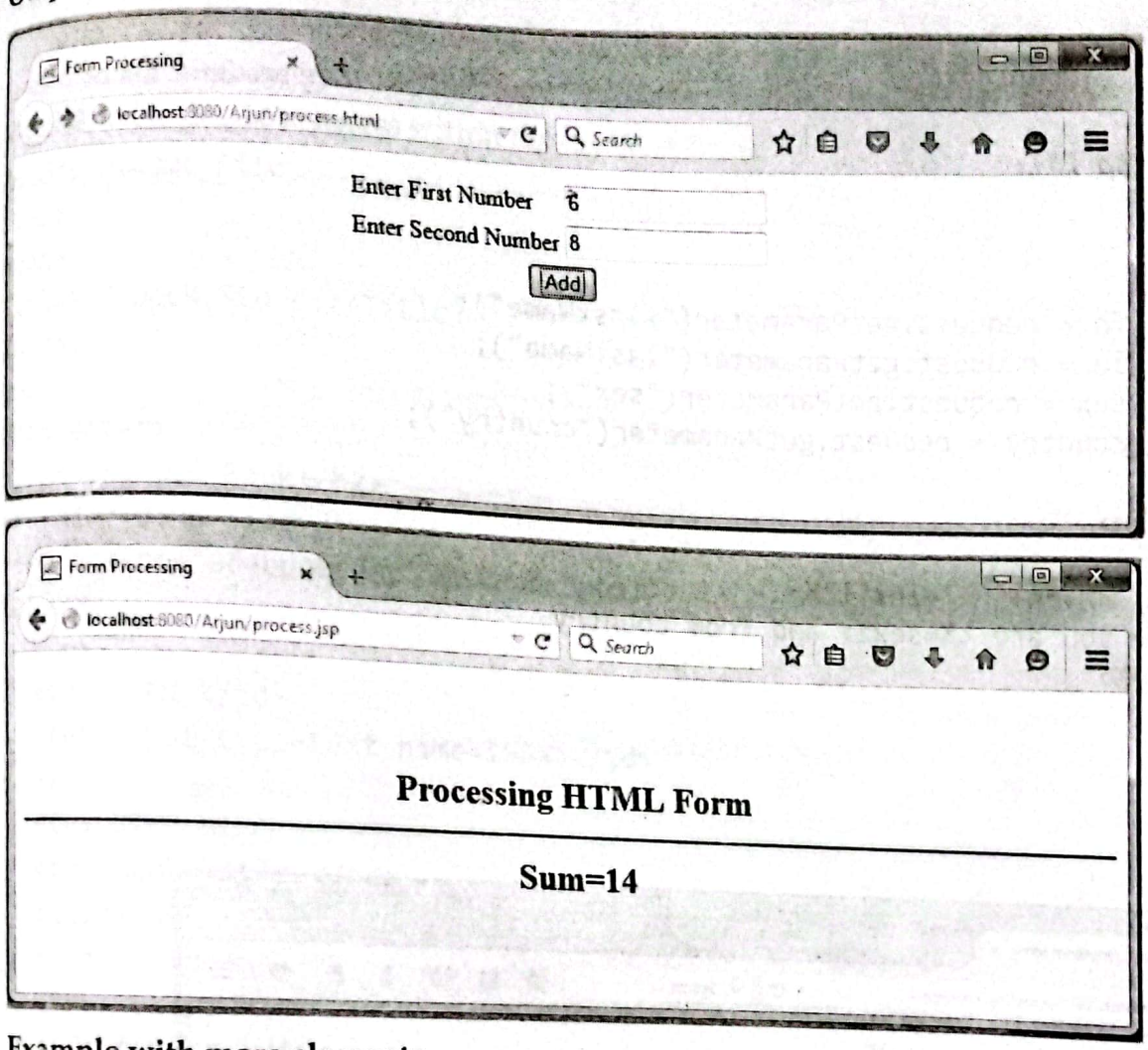


12.5 Form Processing in JSP

Form processing in JSP is similar to form processing in Servlets. We can use request object to read parameter values from HTML file or another JSP file. Methods supported by request object are similar to methods used in dervlets.

```
//process.jsp File
<html>
<head>
<title>Form Processing</title>
</head>
<body>
<font size=3>
<br><br><center><h2>
Processing HTML Form
<hr color=red size=3>
<%
   int x=Integer.parseInt(request.getParameter("first"));
   int y=Integer.parseInt(request.getParameter("second"));
   int z=x+y;
   out.println("Sum="+z);
%>
<h2>
<center>
</font>
</body>
</html>
```

Output

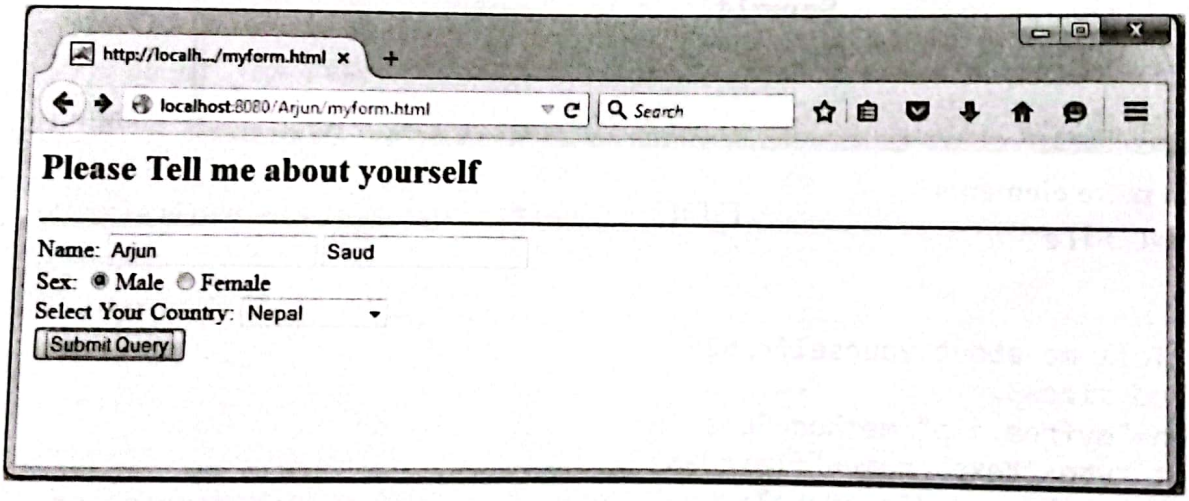


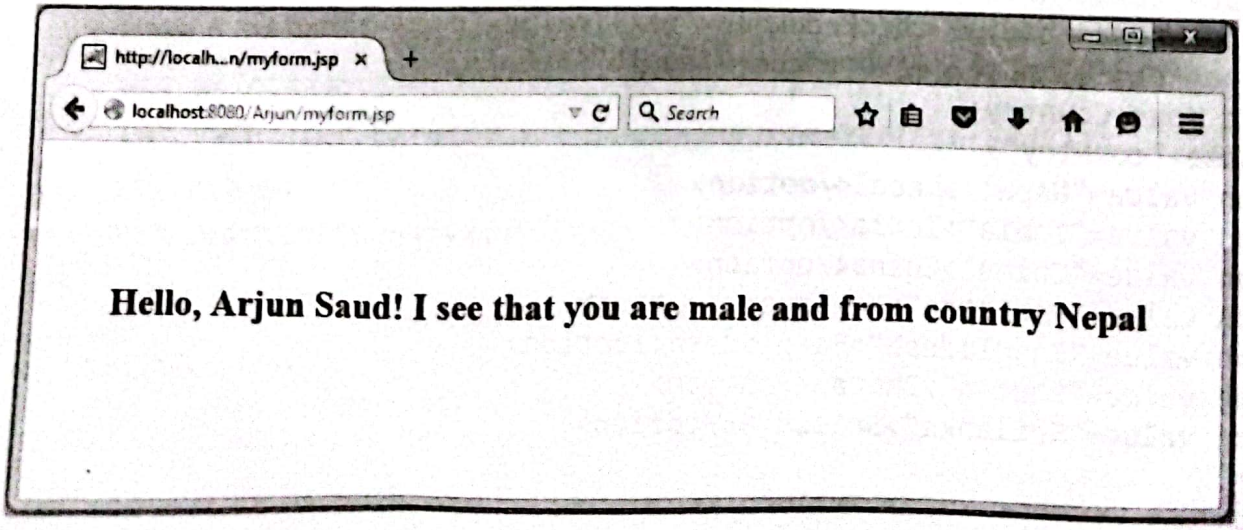
Example with more elements

```
//myform.html File
<html>
<body>
<h2>Please Tell me about yourself</h2>
<hr color=red size=3>
<form action="myfrom.jsp" method="post">
Name: <input type="text" name="firstName">
<input type="text" name="lastName"><br>
Sex:  <input type="radio" checked name="sex" value="male">Male
<input type="radio" name="sex" value="female">Female
<br>Select Your Country:
<select name="country">
   <option value="Nepal">Nepal</option>
   <option value="India">India</option>
   <option value="China">China</option>
   <option value="Pakistan">Pakistan</option>
   <option value="Bangladesh">Bangladesh</option>
   <option value="Bhutan">Bhutan</option>
   <option value="Srilanka">Srilanka</option>
</select>
<br>
<input type="submit">
```

```
</form>
</body>
</html>

//myform.jsp File
<html>
<body>
<%
   String fn = request.getParameter("firstName");
   String ln = request.getParameter("lastName");
   String sex = request.getParameter("sex");
   String country = request.getParameter("country");
%>
<br><br><br>
<center>
<h2>Hello, <%=fn+"  "+ln+"!"%>
I see that you are <%=sex%> and from country
<%=country%>
</h2>
</center>
</body>
</html>
```

Output





12.6 DATABASE ACCESS WITH JSP

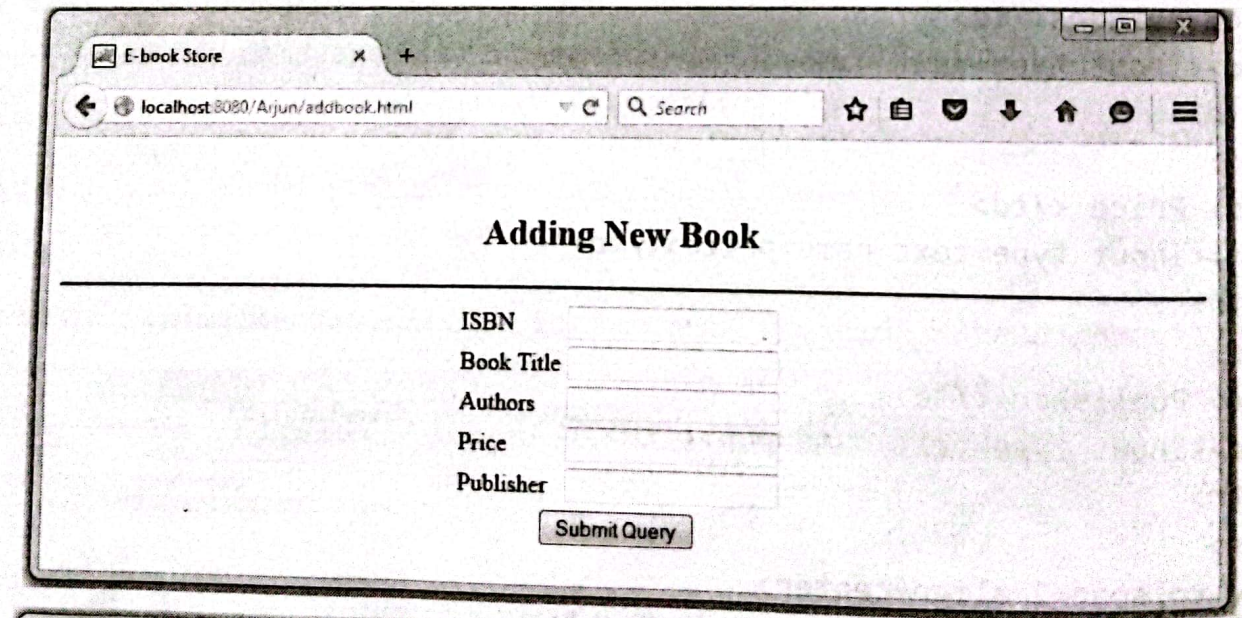
We also access database by using JSP. All JDBC APIs can be accessed from JSP. We have put all codes related to database access within the expression tag.

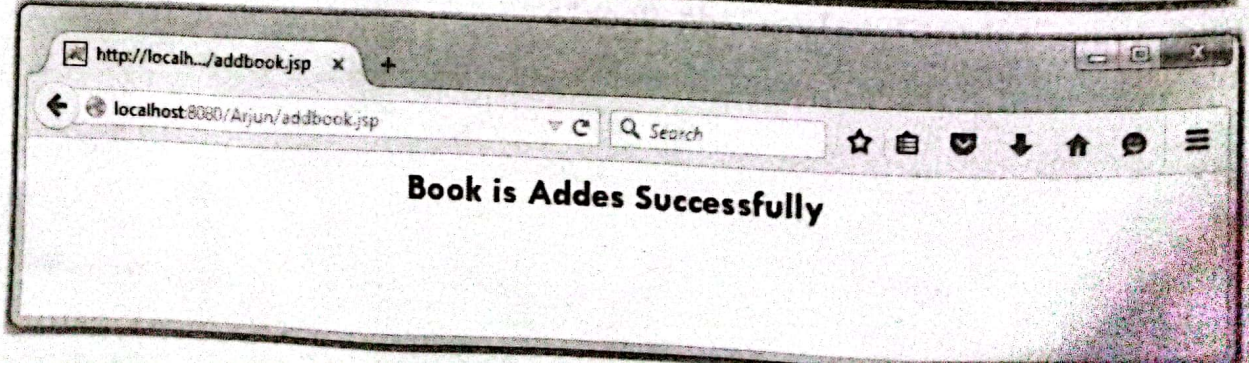
//addbook.html File

```
<html>
<head>
<Title> E-book Store </Title>
</head>
<body>
<br><Center>
<h2>Adding New Book</h2>
<hr color=blue size=3>
<form action="addbook.jsp" method="post">
<table>
   <tr>
   <td> ISBN </td>
   <td><input type=text name=ISBN></td>
   </tr>
   <tr>
   <td> Book Title </td>
   <td><input type=text name=title></td>
   </tr>
   <tr>
   <td> Authors</td>
   <td><input type=text name=authors></td>
   </tr>
   <tr>
   <td> Price </td>
   <td><input type=text name=price></td>
   </tr>
   <tr>
   <td> Publisher </td>
   <td><input type=text name=pub></td>
   </tr>
   <tr>
   <td colspan=2 align=center>
   <input type="submit" value="Add Book">
   </td>
   </tr>
</table>
</form>
</body>
```

```
//addbook.jsp File
<html>
<body>
<%
    String ISBN = request.getParameter("ISBN");
    String title = request.getParameter("title");
    String auth = request.getParameter("authors");
    int price = Integer.parseInt(request.getParameter("price"));
    String pub=request.getParameter("pub");
%>
<h2><center>
<font face="Tw Cen MT" color=blue>
<%@ page import = "java.sql.*" %>
<%
 Connection con =
DriverManager.getConnection("jdbc:mysql://localhost/ebook", "root",
"arjun");
 Statement stmt = conn.createStatement();
 String sql ="insert into books values ('"+ISBN+ "','"+title+
"','"+auth+"'," +price+",'" +pub+"')";
stmt.executeUpdate(sql);
out.println("Book is Addes Successfully");
%>
</font><center><h2>
</body>
</html>
```

Output





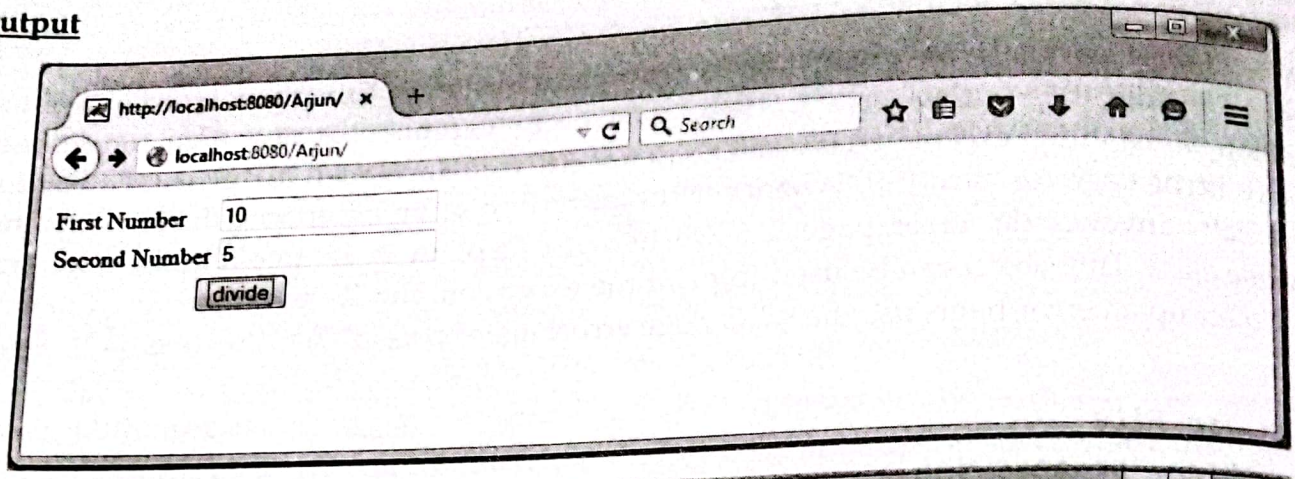
12.7 EXCEPTION HANDLING IN JSP

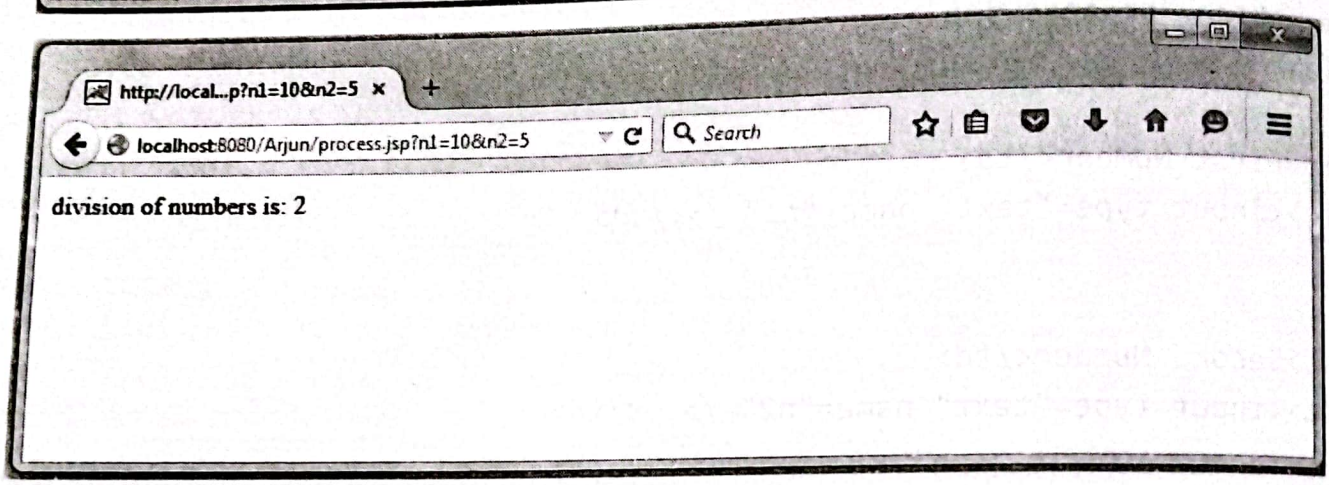
The exception is normally an object that is thrown at runtime. Exception Handling is the process to handle the runtime errors. There may occur exception at any time in our web application. So handling exceptions is a safer side for the web developer. JSP gives us an option to specify Error Page for each JSP. Whenever the page throws an exception, the JSP container automatically invokes the error page. As we know, exception is an implicit object of type Throwable class. This object can be used to print the exception. But it can only be used in error pages. To set up an error page, use the <%@ page errorPage="xxx.jsp" %> directive.

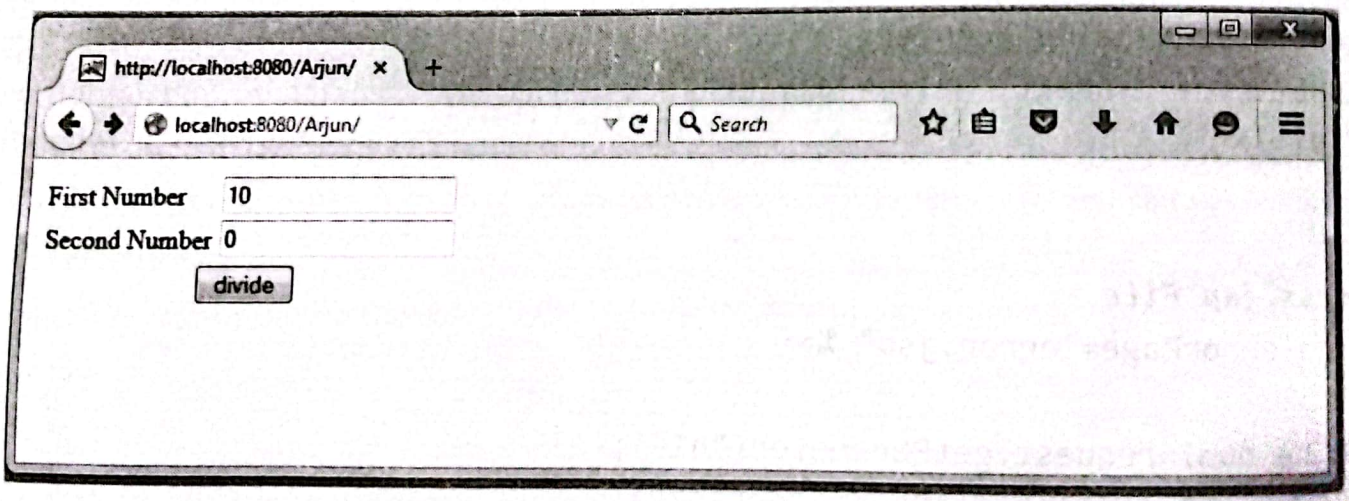
Example

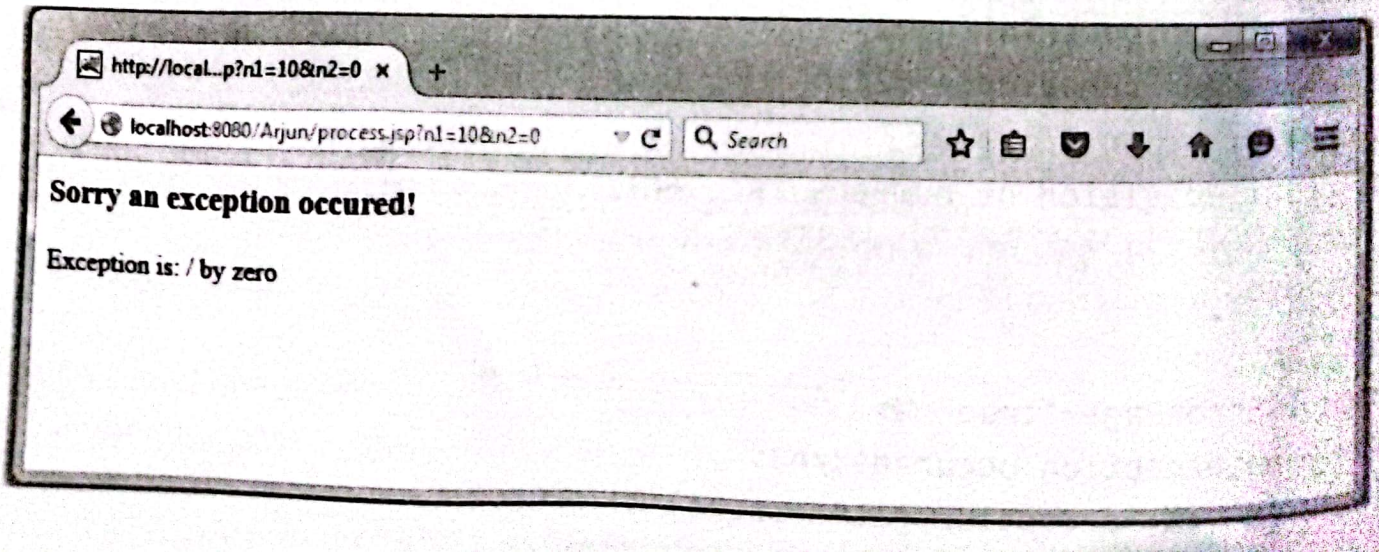
```
//index.jsp File
<form action="process.jsp">
<table>
<tr>
   <td>First Number</td>
   <td><input type="text" name="n1" /></td>
</tr>
<tr>
   <td>Second Number</td>
   <td><input type="text" name="n2" />  </td>
</tr>
<tr>
   <td colspan=2 align=center>
   <input type="submit" value="divide"/>
   </td>
</tr>
</form>
//process.jsp File
<%@ page errorPage="error.jsp" %>
<%
   String num1=request.getParameter("n1");
   String num2=request.getParameter("n2");
   int a=Integer.parseInt(num1);
   int b=Integer.parseInt(num2);
   int c=a/b;
   out.print("division of numbers is: "+c);
%>

//error.jsp
<%@ page isErrorPage="true" %>
<h3>Sorry an exception occured!</h3>
Exception is: <%= exception.getMessage() %>
```

Output









12.8 JSP SESSION TRACKING

In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information. By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows:

```
<%@ page session="false" %>
```

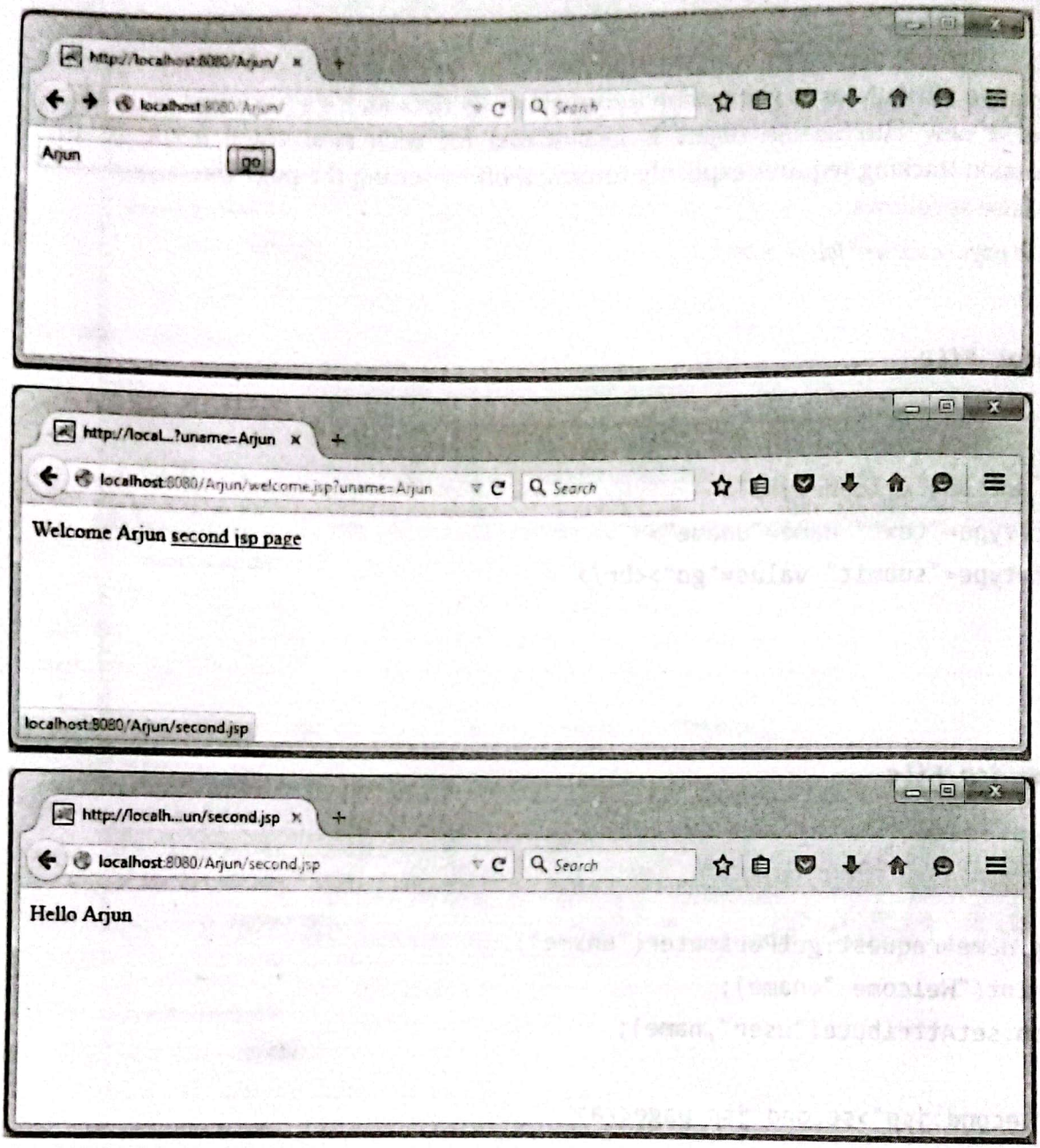
Example

```
//index.html File
<html>
<body>
   <form action="welcome.jsp">
   <input type="text" name="uname">
   <input type="submit" value="go"><br/>
</form>
</body>
</html>
```

```
// welcome.jsp File
<html>
<body>
<%
   String name=request.getParameter("uname");
   out.print("Welcome "+name);
   session.setAttribute("user",name);
%>
<a href="second.jsp">second jsp page</a>
</body>
</html>
```

```
//second.jsp File
<html>
<body>
<%
   String name=(String)session.getAttribute("user");
   out.print("Hello "+name);
%>
</body>
</html>
```

Output



12.9 INTRODUCTION TO JAVA WEB FRAMEWORKS

Frameworks are large bodies of pre-written code to which you add your own code in order to solve a problem. You make use of a framework by calling its methods, inheritance, and supplying callbacks, listeners, or other implementations of the patterns. A framework will often dictate the structure of an application. Some frameworks even supply so much code that you have to do very little to write your application. Three major java web frameworks are briefly discussed below.

- **Struts:** Struts is a Java-based open-sourced framework that helps in developing web application in J2EE. It extends the Java Servlet API and promotes the Model, View, Controller (MVC) architecture. This makes the web applications developed in standard technologies like JSP, JavaBeans, and XML, more maintainable, extensible,

and flexible. The framework is designed to streamline the full development cycle, from building, to deploying, to maintaining applications over time. Struts provide various other features to make web development easier for the developers such as POJO forms and POJO actions, Tag Support, AJAX support, easy Integration and many

- **Spring:** Spring is a powerful lightweight application development framework used for Java Enterprise Edition (JEE). In a way, it is a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF, etc. The framework in a broader sense can be defined as a structure using which you can solve many technical problems. You can say that the Spring Framework is a comprehensive tool for supporting applications using Java programming language.
- **Hibernate:** Hibernate is a framework in Java which comes with an abstraction layer and handles the implementations internally. The implementations include tasks like writing a query for CRUD operations or establishing a connection with the databases, etc. Hibernate develops persistence logic, which stores and processes the data for longer use. It is lightweight and an Objection-relational Mapping (ORM) tool, and most importantly open-source which gives it an edge over other frameworks. ORM is a technique that maps the object stored in the database. An ORM tool simplifies data creation, manipulation, and access. It internally uses the Java API to interact with the databases.

12.10 JAVA BEANS IN JSP

JavaBeans components are Java classes that can be easily reused and composed together into applications. Any Java class that follows certain design conventions is a JavaBeans component. Following are the unique characteristics that distinguish a JavaBean from other Java classes:

- It must provide a default, no-argument constructor.
- It should be serializable and implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "getter" and "setter" methods for the properties.

JavaServer Pages technology directly supports using JavaBeans components with standard JSP language elements. We can easily create and initialize beans and get and set the values of their properties.

jsp:useBean

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows:

```
<jsp: useBean id="beans name" class="class name" scope="beans scope" />
```

Here values for the scope attribute could be page, request, session or application based on our requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other useBean declarations in the same JSP.

Example

```
<html>
   <head>
      <title>useBean Example</title>
   </head>
   <body>
      <jsp:useBean id="date" class="java.util.Date" />
      <p>The date/time is <%= date %>
   </body>
</html>
```

Output

This would produce following result: *The date/time is Thu Sep 30 11:18:11 GST 2010*

jsp:setProperty and jsp:getProperty

Along with <jsp:useBean...>, we can use <jsp:getProperty/> action to access get methods and <jsp:setProperty/> action to access set methods. Here is the full syntax:

```
<jsp:useBean id="beans id" class="class name" scope="beans scope">
   <jsp:setProperty name="beans id" property="property name"
                    value="value"/>
   <jsp:getProperty name="beans id" property="property name"/>
   ...........
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the get or set methods that should be invoked.

Example

```
//Student.java File
public class Student implements java.io.Serializable
{
   private String firstName = null;
   private String lastName = null;
   private int age = 0;
   public Student() {     }
   public String getFirstName()
   {
         return firstName;
   }
   public String getLastName()
   {
         return lastName;
   }
   public int getAge()
```

```
{     return age;

}
public void setFirstName(String firstName)
{     this.firstName = firstName;

}
public void setLastName(String lastName)
{     this.lastName = lastName;

}
public void setAge(Integer age)
{     this.age = age;

}
}

//Student.jsp File
<html>
<head>
<title>get and set properties Example</title>
</head>
<body
  <jsp:useBean id="st" class="Student">
     <jsp:setProperty name="st" property="firstName" value="Aagaman"/>
     <jsp:setProperty name="st" property="lastName" value="Saud"/>
     <jsp:setProperty name="st" property="age" value="5"/>
  </jsp:useBean>
  <p>Student First Name:
  <jsp:getProperty name="st" property="firstName"/>
  </p>
  <p>Student Last Name:
  <jsp:getProperty name="st" property="lastName"/>
  </p>
  <p>Student Age:
  <jsp:getProperty name="st" property="age"/>
  </p>
</body>
</html>
```

Output

This would produce following result:

Note: *Make Students.class available in CLASSPATH and try to access above JSP.*

EXERCISE

1. How JSP is different from Servlets? Explain advantages of JSP over other server side scripting languages.
2. Explain JSP access model with suitable diagram.
3. What is meant by implicit objects? Explain different implicit objects provided by JSP briefly.
4. Write a JSP script that reads value from a textbox and identifies whether it is odd or even.
5. List the different tags available in JSP. Explain each of the tag with brief description and syntax.
6. What is meant by scope of objects? Explain different scopes of JSP objects briefly with example.
7. Write a JSP script that that demonstrates use of select and insert operations with suitable example.
8. How JSP can be used for enabling and disbling session? Explain with suitable JSP script.
9. How exceptions can be handled in JSP scripts? Explain with suitable JSP script.
10. What is meant by Java Bean? How it can be used in JSP scripts? Explain with suitable example.

❑❑❑