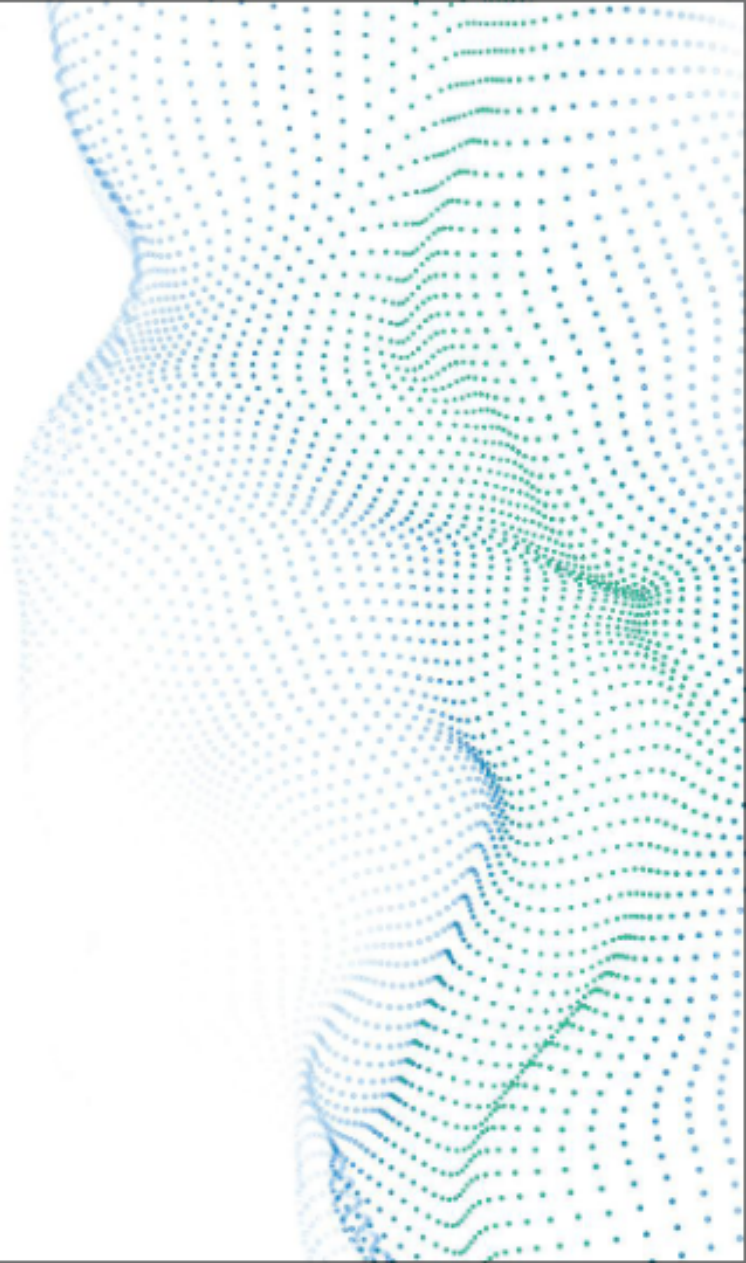
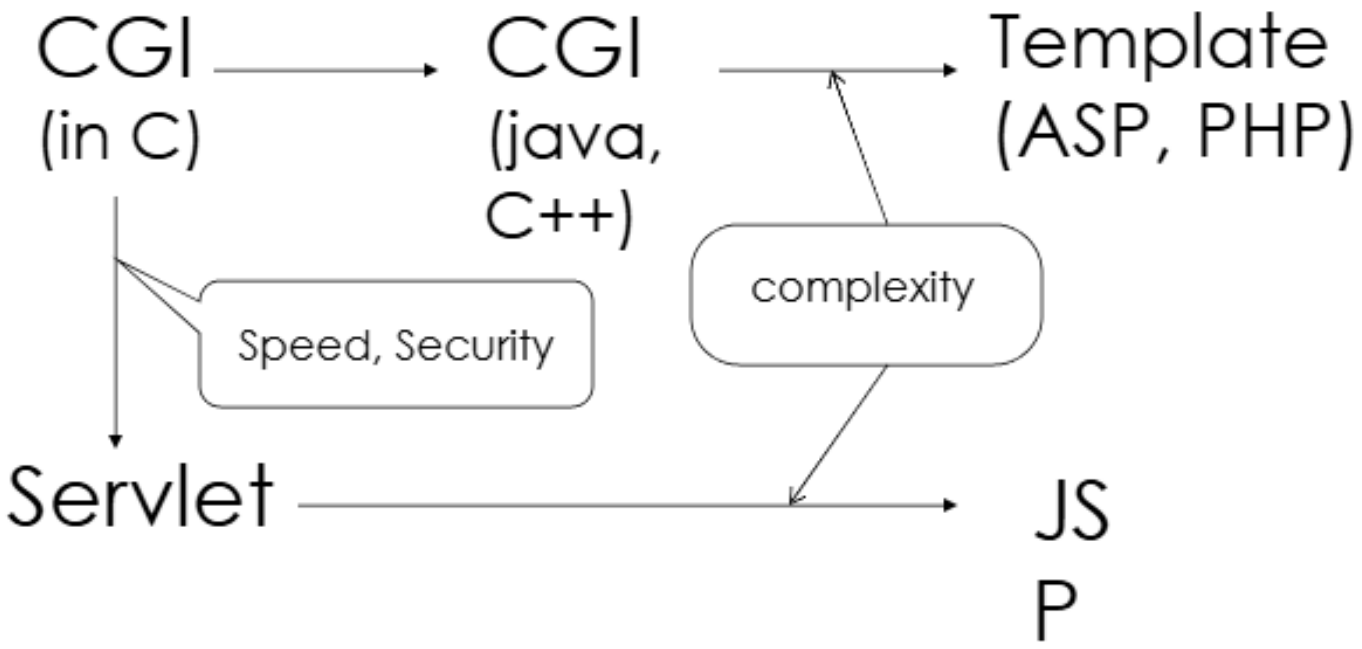
Advance Java



Java Server Pages

Overview of History



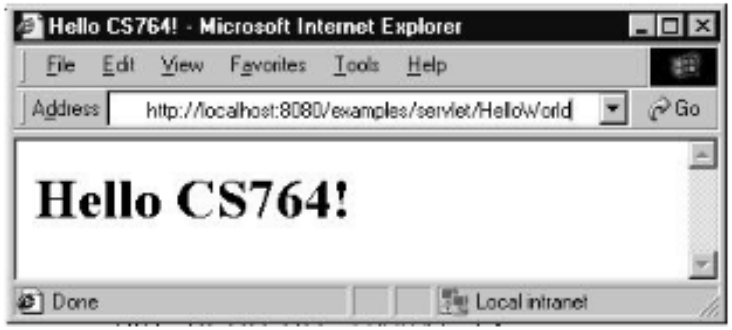
HelloWorld

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld 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("<body>");
      out.println("<head>");
      out.println("<title>Hello CS764!</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Hello CS764!</h1>");
      out.println("</body>");
      out.println("</html>");
                  out.close();
   }
}
```

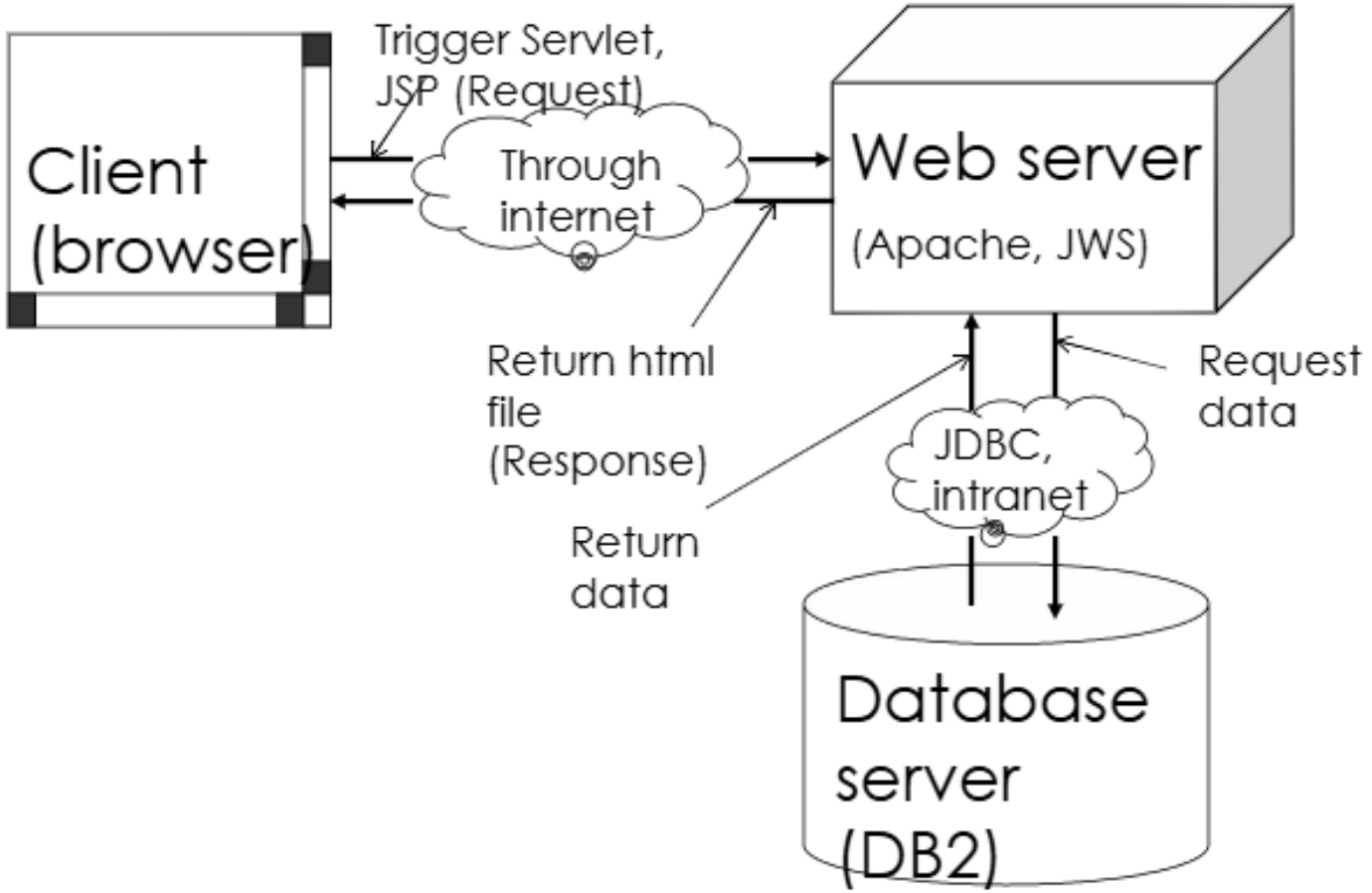
EduSkills

```
<html><head></head>
<body>
<a href="../servlet/HelloWorld">
<h1>Execute HelloWorld Servlet</h1>
</a>
</body>
</html>
```

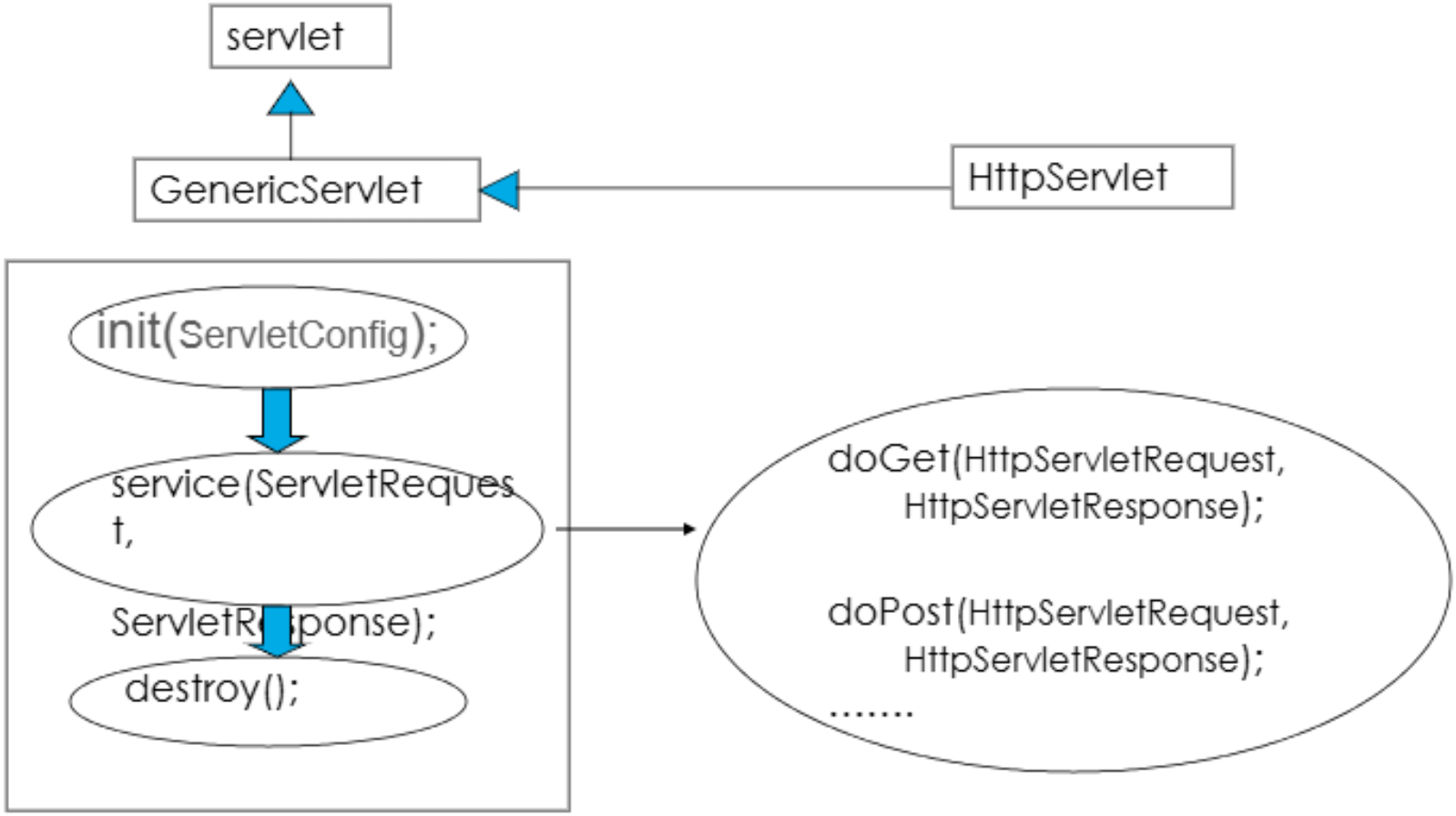

```
<html>
<head>
<title>Hello CS764!</title></head>
<body>
<h1>Hello CS764!</h1>
</body>
</html>
```



Client - Server - DB



Life Cycle of Servlet



Interaction with Client

HttpServletRequest

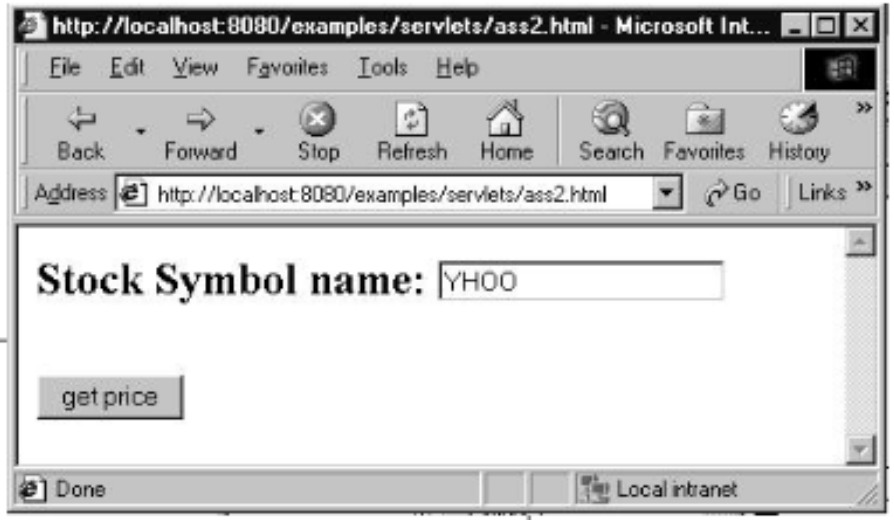
- String getParameter(String)
- Enumeration getParameters(String[])

HttpServletResponse

- Writer getWriter()
- ServletOutputStream getOutputStream()

Handling GET and POST Requests

Assignment 2: Get Stock Price



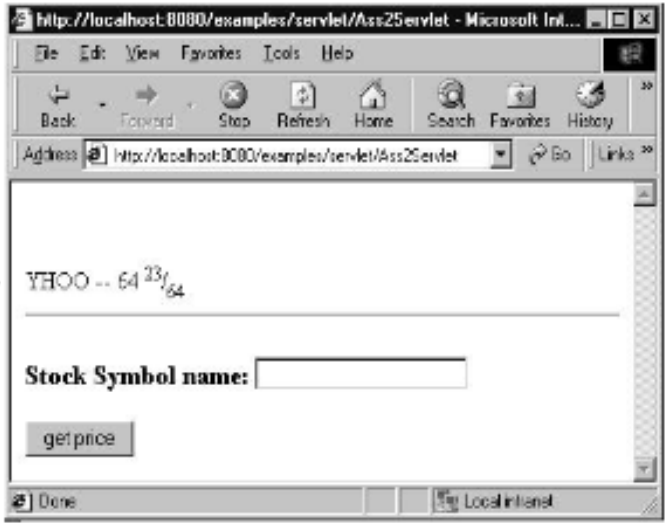
Ass2.htm

```
<html><head></head>
<body>
<form action="../servlet/Ass2Servlet"
method=POST>
<h2>Stock Symbol name:
<input type=text name="stockSymbol"></h2><br>
<input type="submit" value = "get price">
</form>
</body></html>
```

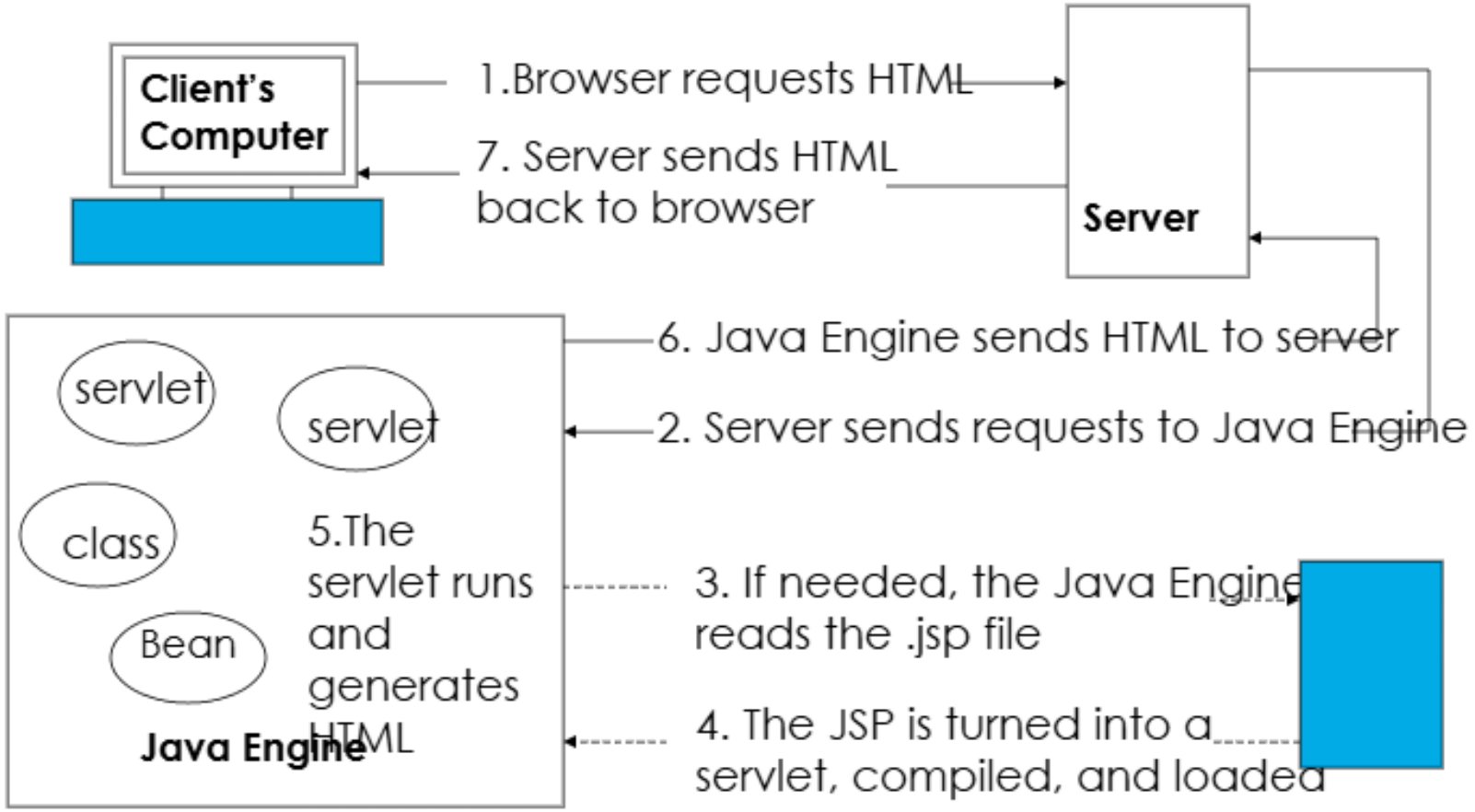
Client Side

Ass2Servlet

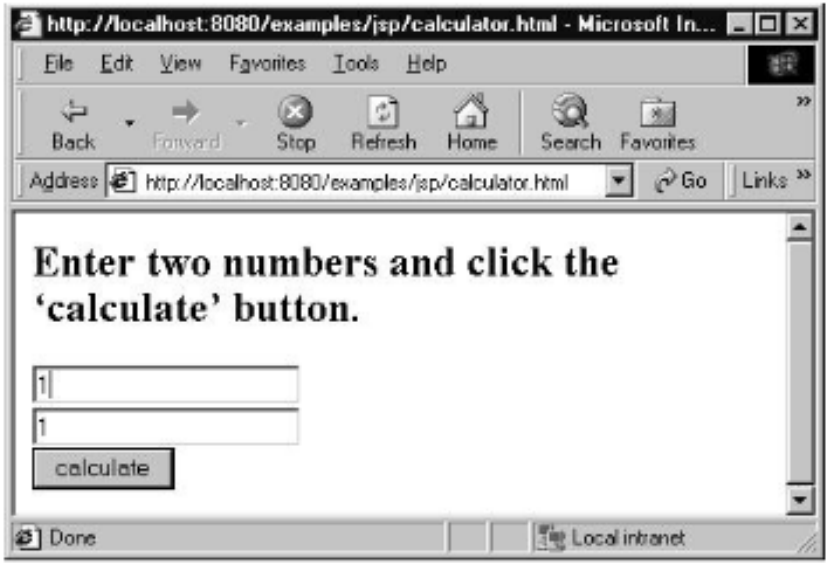
```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Ass2Servlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
    HttpServletResponse res)
    throws IOException, ServletException
    {
    res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        String stockSymb = request.getParameter("stockSymbol");

    StockGrabber sg = new StockGrabber();
    sg.setStockSymbol(stockSymb); // Set the stock symbol as "input"
    String stockPrice = sg.getPrice();// Get the price of stock

    System.out.println("After StockGrabber.getPrice --"+stockPrice);// Debug
        out.println("<html><head></head><body><br><br>");
        out.println(stockSymb + " -- " + stockPrice);
    out.println("<hr>");
    out.println("<form action=\"../servlet/Ass2Servlet\" method=POST>");
    out.println("<h3>Stock Symbol name: <input type=text name=\"stockSymbol\"></h3>");
    out.println("<input type=submit value=\"get price\">");
    out.println("</form>");
    out.println("</body></html>");
    }
}
```



JAVA SERVER PAGES (JSP)

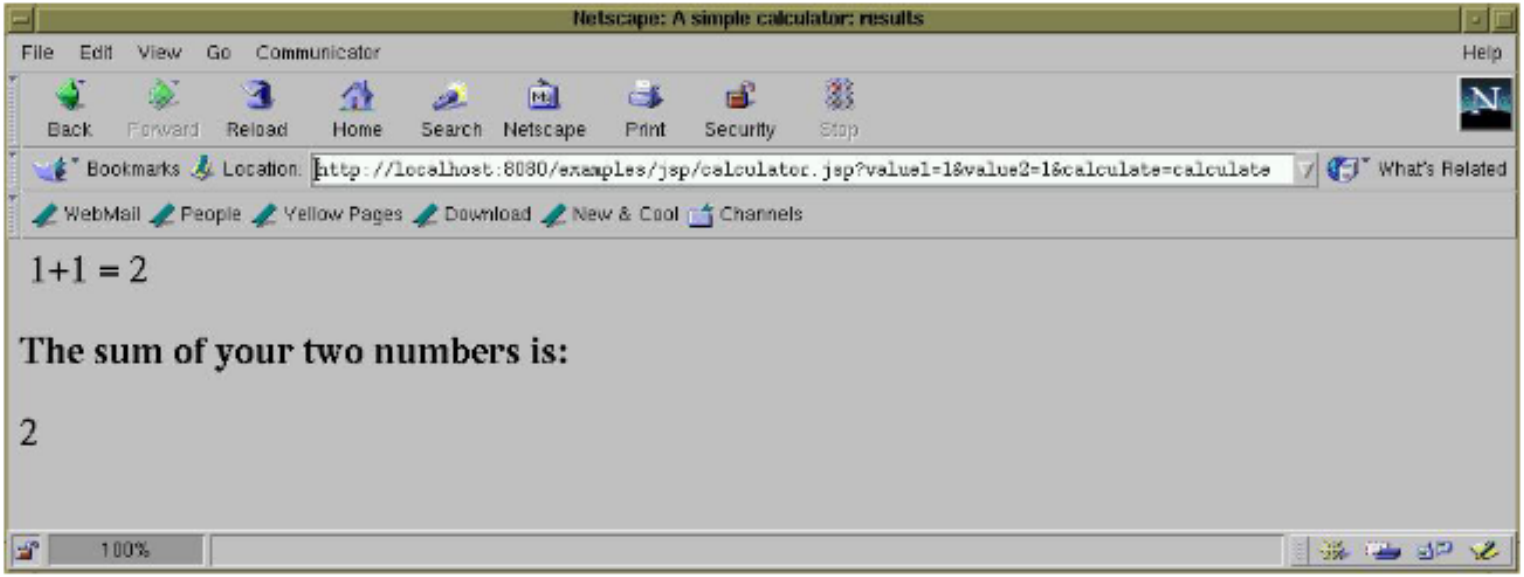


A FIRST JSP




```
<html>
<head></head>
<body>
<p>Enter two numbers and click the
'calculate' button.</p>

<form action="calculator.jsp" method="get">
<input type=text name=value1><br>
<input type=text name=value2 ><br>
<input type=submit name=calculate value=calculate>
</form>
</body>
</html>
```

Calculator.html




```
<html>
<head><title>A simple calculator: results</title></head>
<body>
<%-- A simpler example 1+1=2 --%>
1+1 = <%= 1+1 %>
<%-- A simple calculator --%>
<h2>The sum of your two numbers is:</h2>
<%= Integer.parseInt(request.getParameter("value1")) +
    Integer.parseInt(request.getParameter("value2")) %>
</body>
</html>
```

Calculator.jsp

JSP Tags

Comments	<%-- ……text…… --%>
Declaration	<%! int i; %>
	<%! int numOfStudents(arg1,..) {} %>
Expression	<%= 1+1 %>
Scriptlets	<% … java code … %>
include file	<%@ include file="*.jsp" %>
……	

Using Java Bean

Declaration

1. **<jsp:useBean id="*bean1*" class="*Bean1*"/>**
2. <jsp:useBean id="bean1" class="Bean1" name="serBean" type="SerBean1"/>

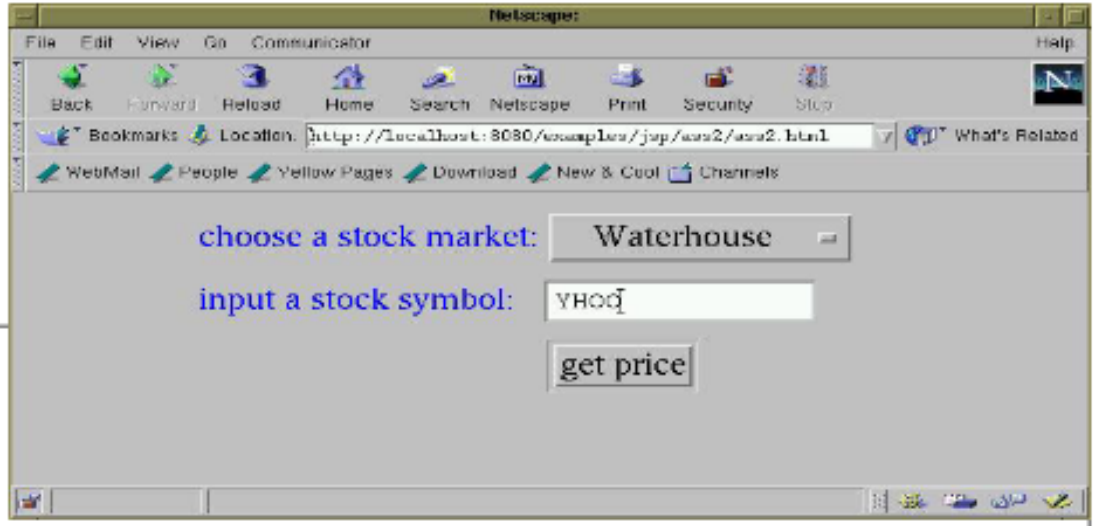
Setting property

1. **<jsp:setProperty name="*bean1*" property="*color*" value="*red*"/>**
2. <jsp:setProperty name="bean1" property="color"/>
3. <jsp:setProperty name="bean1" property="color" param="bgColor"/>
4. <jsp:setProperty name="bean1" property="*"/>

Getting property

1. **<jsp:getProperty name="*bean1*" property="*color*"/>**
2. <%=bean1.getColor() %>

Assg2 example




```
<html>
<head></head>
<body>
<center>
<table border = 0>
<form action=ass2.jsp method = POST>
<tr><td><font color=blue>choose a stock market:</font></td>
    <td><select name="stockMarket">
      <option value="Waterhouse">Waterhouse</option>
      <option value="Yahoo">Yahoo</option>
      <option value="ChicagoStockex">Chicago Stockex</option>
      <option value="Reuters">Reuters</option>
        </select></td>
</tr>
<tr><td><font color = blue>input a stock symbol:</font></td>
<td><input type="edit" name="stockSymbol" size=15></td>
</tr>
<tr><td></td><td><input type="submit" value = "get price"></td></tr>
</table>
</form></center>
</body></html>
```

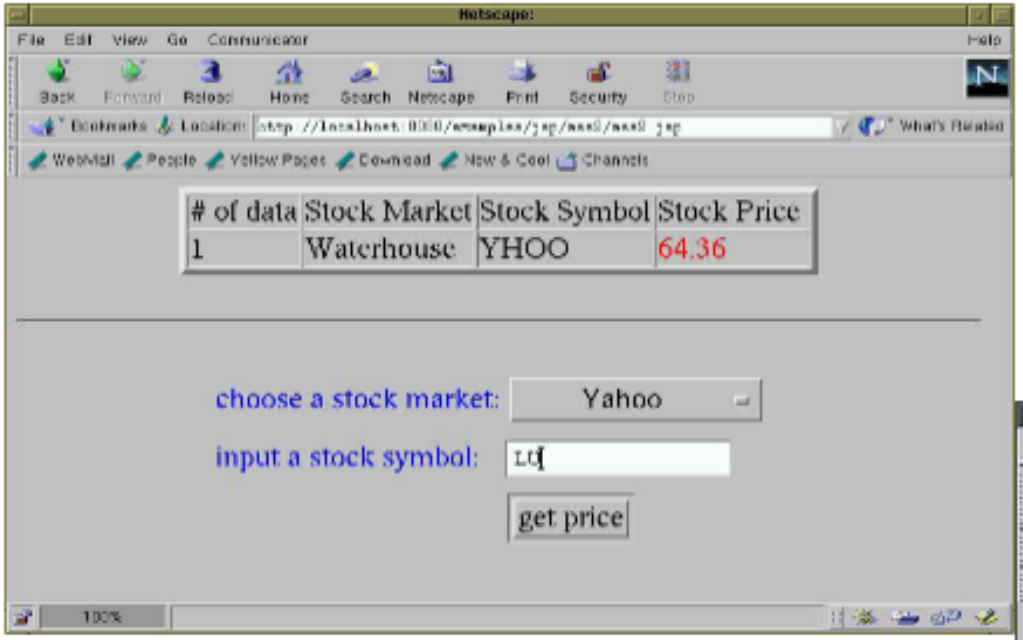
Client side
Ass2.html

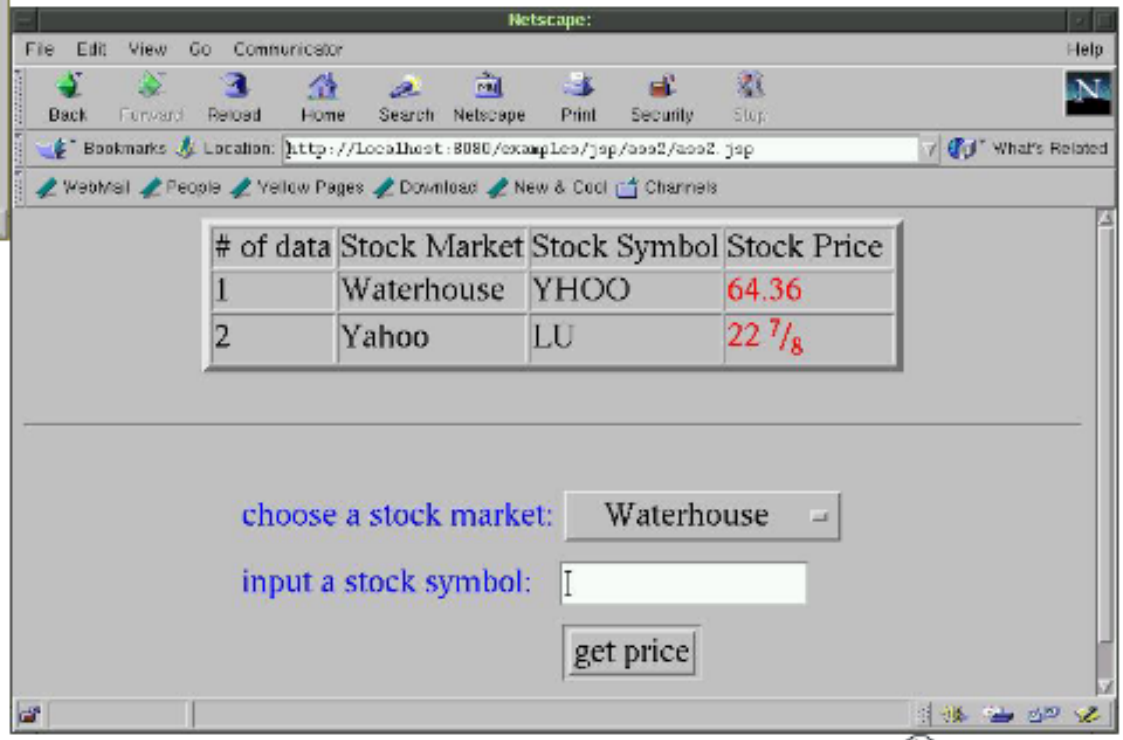
ass2.jsp

```
<html><head>
<jsp:useBean id="ass2" scope="session" class="ass2.StockGrabber" />
<jsp:setProperty name="ass2" property="*" />
</head>
<body><h2><%
            ass2.processInput();
            ass2.getPrice();
        %>
<center><table border=5>
<tr><td># of data</td>    <td>Stock Market</td>    <td>Stock Symbol</td>    <td>Stock
Price </td>
</tr><%
        String[] stockMarkets = ass2.getStockMarkets();
        String[] symbols = ass2.getSymbols();
        String[] prices  = ass2.getPrices();
        for(int i=0; i<prices.length; i++){
       %>
<tr><td>          <%= i+1 %>              </td>
    <td>    <%= stockMarkets[i] %>    </td>
    <td>    <%= symbols[i] %>          </td>
    <td><font color=red><%= prices[i] %></font></td>
</tr><%
       }
       %>
</table>
</center>
</h2>
<hr><%@include file="ass2.html" %></html>
```

```
<jsp:setProperty name="ass2" property="stockSymbol"/>
<jsp:setProperty name="ass2" property="stockMarket"/>
```

Server side





Without using JDBC

```
Public class StockGrabber {
   ...
   public void processInput(){
    if(stockMarket.compareTo("Waterhouse")==0){
           setPrePriceString("<!--Last-->");
            setPostPriceString("</FONT>");
           setUrlPrefix("http://research.tdwaterhouse.com/
              waterhouse/quote.asp?ticker=");
       }
       else if(stockMarket.compareTo("Yahoo")==0){
           setPrePriceString("<td nowrap><b>");
           setPostPriceString("</b></td>");
           setUrlPrefix("http://finance.yahoo.com/q?s=");
       }
       ...
       else if(...){}
       ...
       else{...}
   }
   ...
}
```

Using JDBC --> Database

```
import java.sql.*;
Public class StockGrabber {
    ...
    public void processInput(){
     try  {
          Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
              String sourceURL="jdbc:odbc:stockInfo";
          Connection databaseConnection=DriverManager.getConnection(sourceURL);
          Statement statement=databaseConnection.createStatement();
          ResultSet  info =statement.executeQuery(
                       "select tPrePriceStr, tPostPriceStr, tUrlPrefix
                  from stockMarketData
                       where tStockMarket = stockMarket");
          while(inf.next())
               {
              prePriceString  = info.getString("tPrePriceStr");
            postPriceString = info.getString("tPostPriceStr");
            urlPrefix       = info.getString("tUrlPrefix");
               }
          }
     catch(SQLException e){ ... }
     ...
    }
}
```

Java Server Faces Framework

Java Server Faces

- JSF stands for Java Server Faces. It is a server-side Java framework for web development.
- Our JSF tutorial includes all topics of JSF such as features, example, validation, bean validation, managed bean, referencing managed bean method, facelets etc.
- It is a server side component based user interface framework. It is used to develop web applications. It provides a well-defined programming model and consists of rich API and tag libraries. The latest version JSF 2 uses Facelets as its default templating system. It is written in Java.
- The JSF API provides components (inputText, commandButton etc) and helps to manage their states. It also provides server-side validation, data conversion, defining page navigation, provides extensibility, supports for internationalization, accessibility etc.
- The JSF Tag libraries are used to add components on the web pages and connect components with objects on the server. It also contains tag handlers that implements the component tag.

- With the help of these features and tools, you can easily and effortlessly create server-side user interface.

Benefits of JSF

It provides clean and clear separation between behavior and presentation of web application. You can write business logic and user interface separately.

2) JavaServer Faces API?s are layered directly on top of the Servlet API. Which enables several various application use cases, such as using different presentation technologies, creating your own custom components directly from the component classes.

3) Including of Facelets technology in JavaServer Faces 2.0, provides massive advantages to it. Facelets is now the preferred presentation technology for building JavaServer Faces based web applications.

Prerequisites

Java: You must have Java 7 or higher version.

Java IDE: In this tutorial, we have used NetBean IDE 8.2. Although you can also use other Java IDEs.

Server: We did not install server seperatly. All the examples are executed on the default server installed along with NetBeans IDE 8.2.

JSF FEATURES

Component Based Framework

Implements Facelets Technology

Integration with Expression Language

Support HTML5

Ease and Rapid web Development.

Support Internationalization

Bean Annotations

Default Exception Handling

Templating

Inbuilt AJAX Support

Security

FACELETS TECHNOLOGY

Facelets is an open source Web template system. It is a default view handler technology for JavaServer Faces (JSF). The language requires valid input XML documents to work. Facelets supports all of the JSF UI components and focuses completely on building the view for a JSF application.

EXPRESSION LANGUAGE

Expression Language provides an important mechanism for creating the user interface (web pages) to communicate with the application logic (managed beans). The EL represents a union of the expression languages offered by JavaServer Faces technology.

JAVA SERVER FACES LIFECYCLE

JavaServer Faces application framework manages lifecycle phases automatically for simple applications and also allows you to manage that manually. The lifecycle of a JavaServer Faces application begins when the client makes an HTTP request for a page and ends when the server responds with the page.

The JSF lifecycle is divided into two main phases:

Execute Phase

Render Phase

EXECUTE PHASE

In execute phase, when first request is made, application view is built or restored. For other subsequent requests other actions are performed like request parameter values are applied, conversions and validations are performed for component values, managed beans are updated with component values and application logic is invoked.

The execute phase is further divided into following subphases.

Restore View Phase

Apply Request Values Phase

Process Validations Phase

Update Model Values Phase

Invoke Application Phase

Render Response Phase

RESTORE VIEW PHASE

When a client requests for a JavaServer Faces page, the JavaServer Faces implementation begins the restore view phase. In this phase, JSF builds the view of the requested page, wires event handlers and validators to components in the view and saves the view in the FacesContext instance.

If the request for the page is a postback, a view corresponding to this page already exists in the FacesContext instance. During this phase, the JavaServer Faces implementation restores the view by using the state information saved on the client or the server.

APPLY REQUESTS VALUE PHASE

In this phase, component tree is restored during a postback request. Component tree is a collection of form elements.Each component in the tree extracts its new value from the request parameters by using its decode (processDecodes()) method. After that value is stored locally on each component.

If any decode methods or event listeners have called the renderResponse method on the current FacesContext instance, the JavaServer Faces implementation skips to the Render Response phase.

If any events have been queued during this phase, the JavaServer Faces implementation broadcasts the events to interested listeners.

If the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the FacesContext.responseComplete() method.

Process Validations phase

In this phase, the JavaServer Faces processes all validators registered on the components by using its validate () method. It examines the component attributes that specify the rules for the validation and compares these rules to the local value stored for the component. The JavaServer Faces also completes conversions for input components that do not have the immediate attribute set to true.

If any validate methods or event listeners have called the renderResponse method on the current FacesContext, the JavaServer Faces implementation skips to the Render Response phase.

If the application needs to redirect to a different web application resource or generate a response that does not contain any JavaServer Faces components, it can call the FacesContext.responseComplete method.

END MODULE-8 ADVANCE JAVA