
Servlets-JSP

Table of Contents

Module 1: Getting Started	3
Module 2: The Servlet's Life Cycle	21
Module 3: Form Data ,Context and Configuration Parameters	27
Module 4: Analyzing the Request Headers	42
Module 5: Setting the Response Header	46
Module 6: Redirection in Servlets	52
Module 7: Request Dispatching	56
Module 8: The Cookies	61
Module 9: Session Tracking	72
Module 10: The JSP Technology	83
Module 11: Predefined Variables and Page Directives	94
Module 12: Including Files in JSP	97
Module 13: The 'useBean' tag in JSP	104
Module 14: The Expression Language (EL)	110
Module 15: The MVC Architecture	118
Module 16: The Custom Tags	121

Module 1 : Getting Started

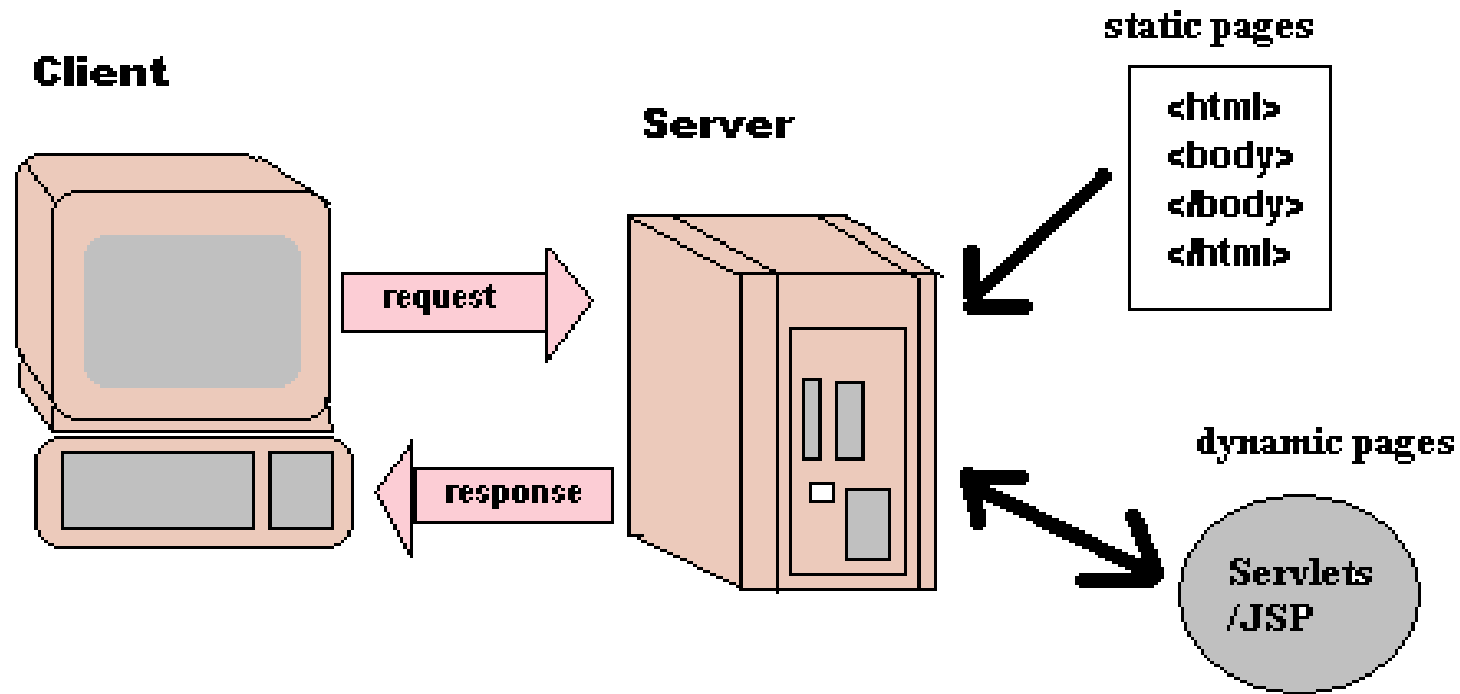
Overview :

- Introduction to Servlet Technology
- Setting up Servlet programming environment
- Writing and deploying first Servlet
- Role of web server

Flavours of Java

- J2SE(Standard Edition)
- J2EE(Enterprise Edition)
- J2ME(Micro Edition)

Client Server Technology



Need of Dynamic Web Page Building

- Data on web page depends on the client request
 - Ex : Search engines
- Data on web pages changes frequently
 - Ex : Weather reports
- Data on web pages uses data from corporate databases
 - Ex : Stock indexes

A Static HTML Page

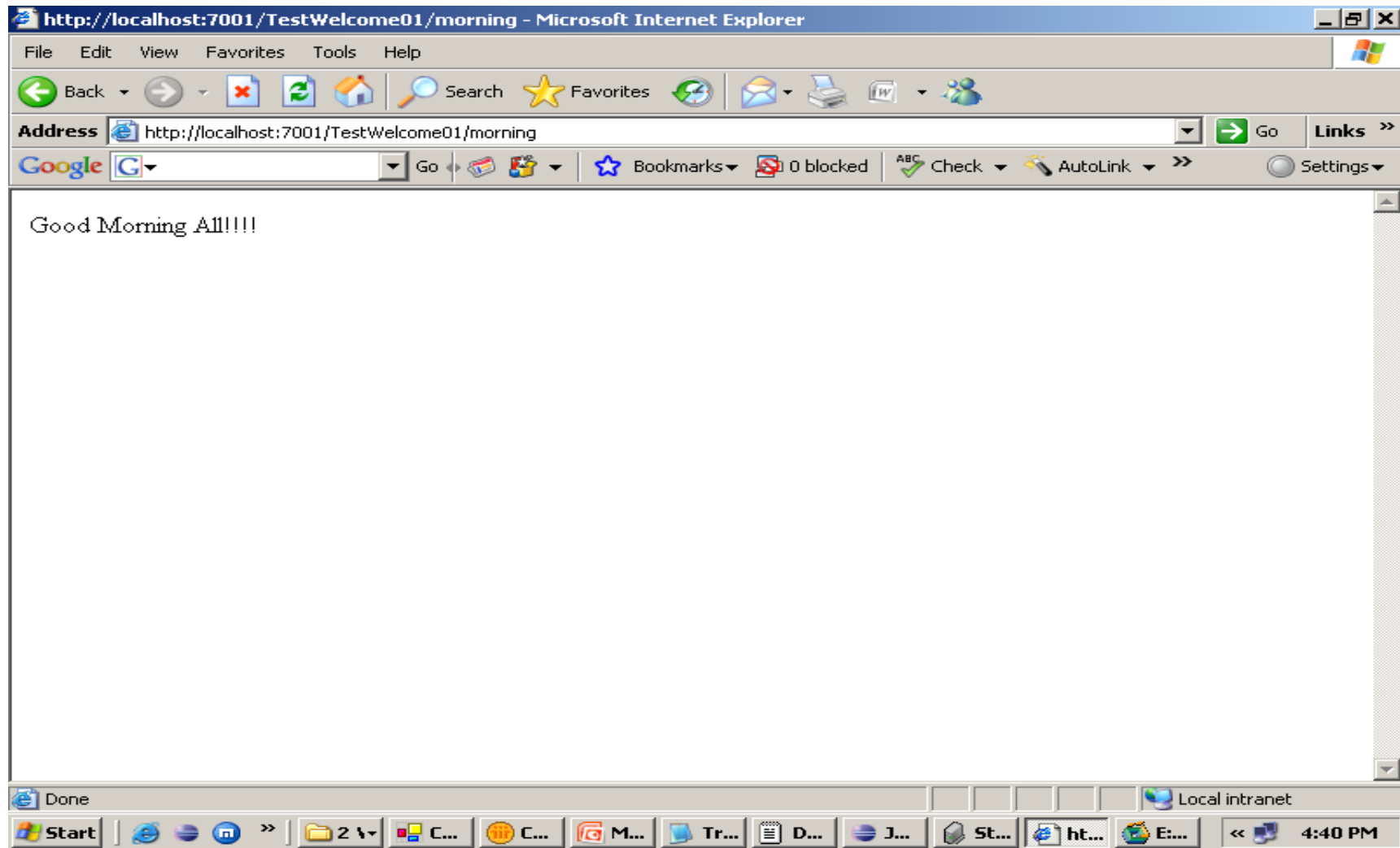
```
<html>
  <head>
    <title>
      Displaying Date
    </title>
  </head>
  <body>
    <script language="java script">
      <!-- document.write(Date( )); -->
    </script>
  </body>
</html>
```

Writing the First Servlet “GoodMorning.java”

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

public class GoodMorning extends HttpServlet {
    protected void doGet(HttpServletRequest request , HttpServletResponse response) throws
        IOException {
        response.setContentType( “text/html” );
        PrintWriter out = response.getWriter();
        out.println( “GOOD MORNING EVERYBODY !!!! ” );
    }
}
```


Accessing Servlet



Servlet Showing Dynamic Nature

```
public class DateFormat extends HttpServlet{
    public void doGet(HttpServletRequest request, HttpServletResponse
        response)throws IOException{

        PrintWriter out = response.getWriter( );
        out.println("<B>STATIC DATA</B><BR>");
        out.println("GOOD MORNING EVERYBODY!!!!!!");

        String [ ] monthNames = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec" };
        Calendar timeNow = Calendar.getInstance( );
        int date = timeNow.get(Calendar.DAY_OF_MONTH);
        int month = timeNow.get(Calendar.MONTH);

        out.println("<BR><BR><B>DYNAMICALLY GENERATED DATA</B><BR>");
        out.println(" Today is : "+date+" th"+monthNames[month]+" "+year+"<BR>");
    }
}
```

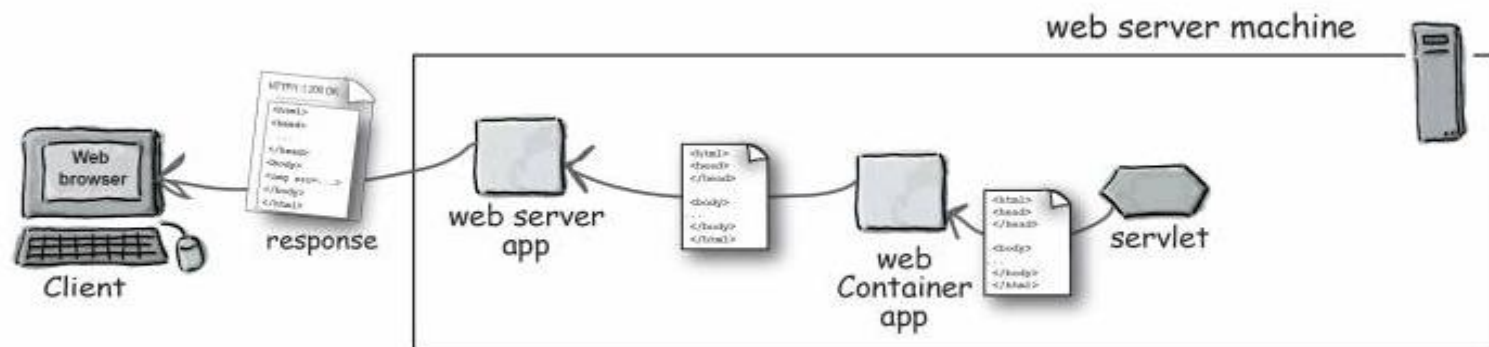
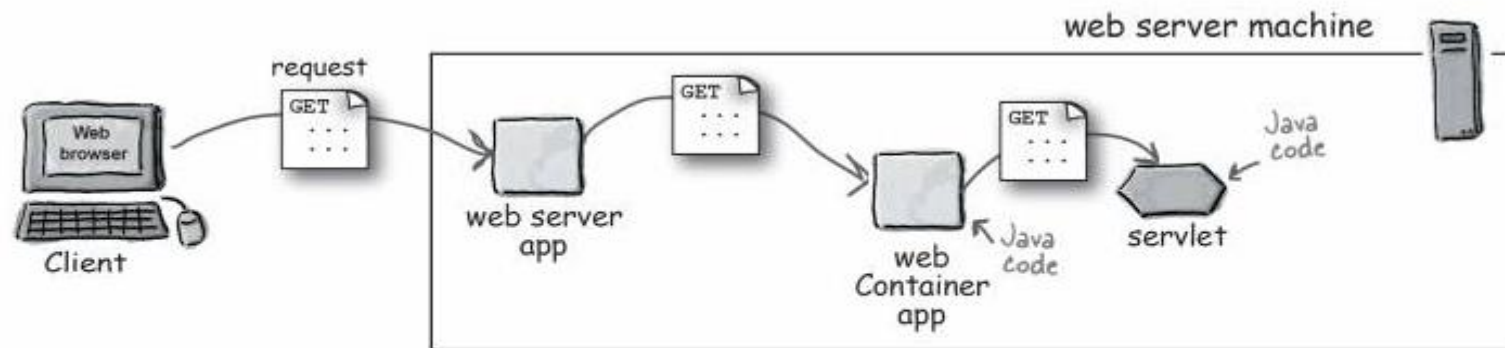
Other Competing Technologies

- HTML (HyperText Markup Language)
- CGI (Common Gateway Interface)
- ASP (Active Server Pages)
- JavaScript
- PHP (HyperText Pre-Processor)
- ColdFusion

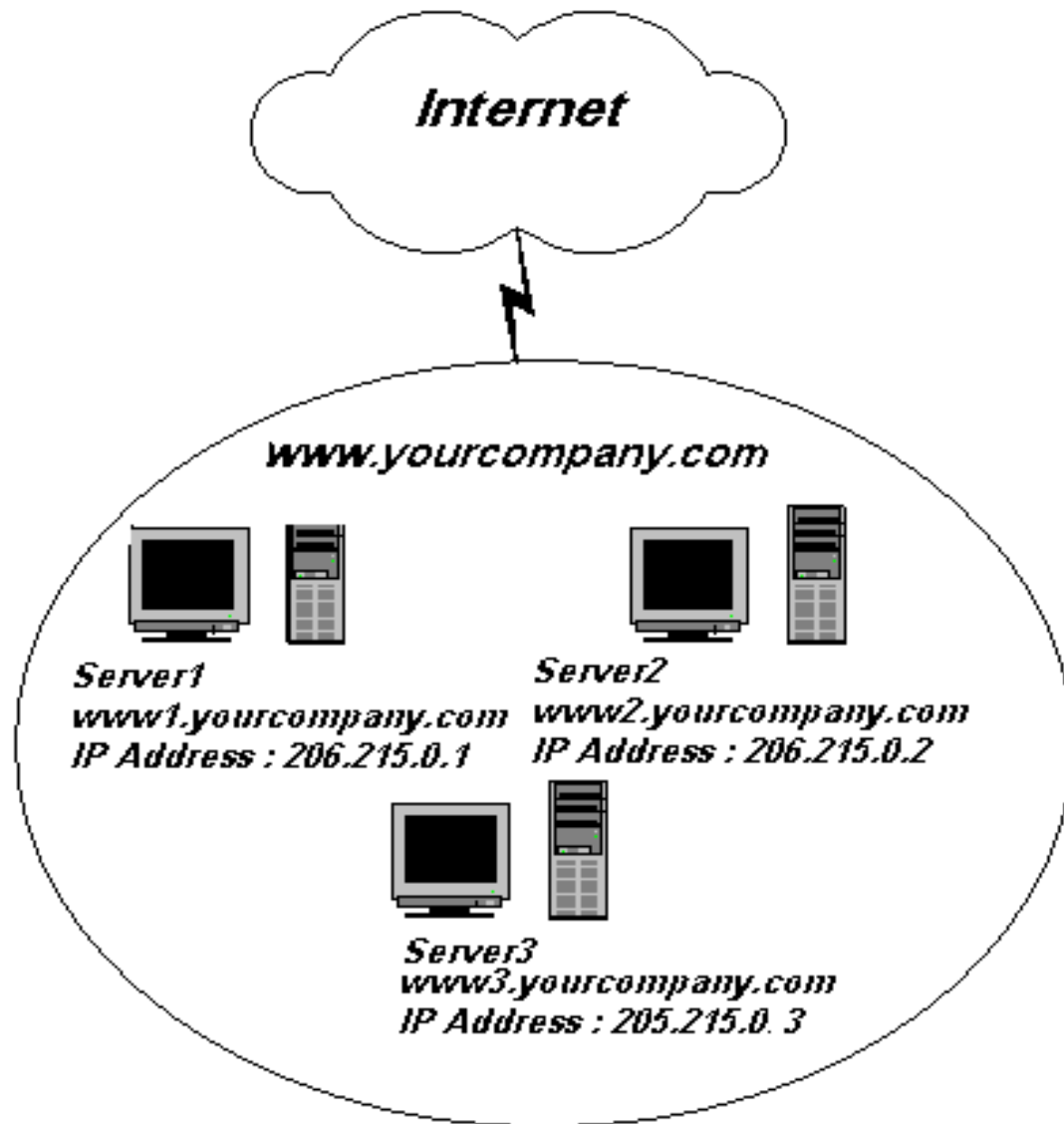
Servlets Features

- Portability
- Powerful
- Efficiency
- Safety
- Integration
- Extensibility
- Inexpensive

Role of a Server

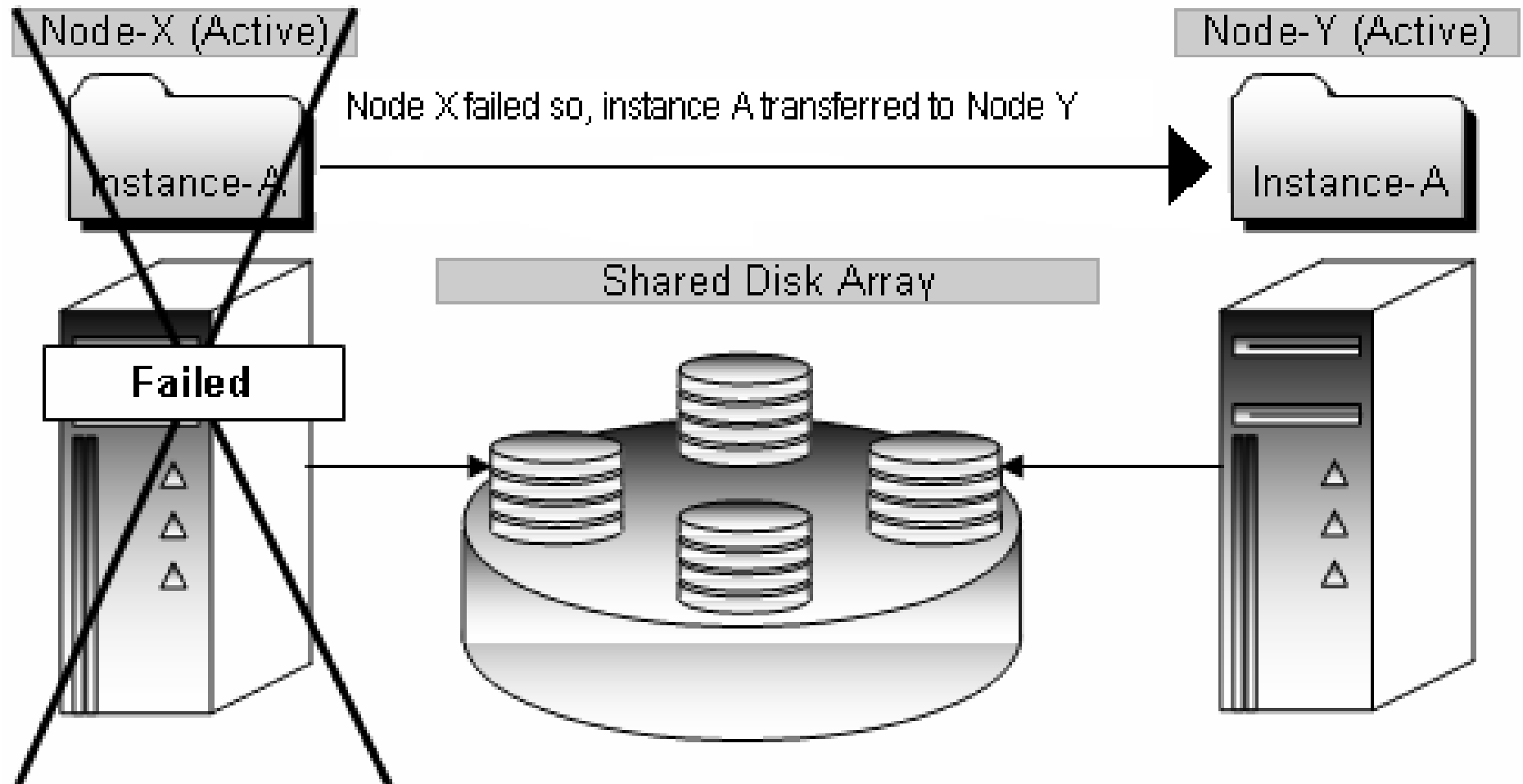


Features of Server



***Web Server Cluster for
www.yourcompany.com***

Features of Server (contd...)

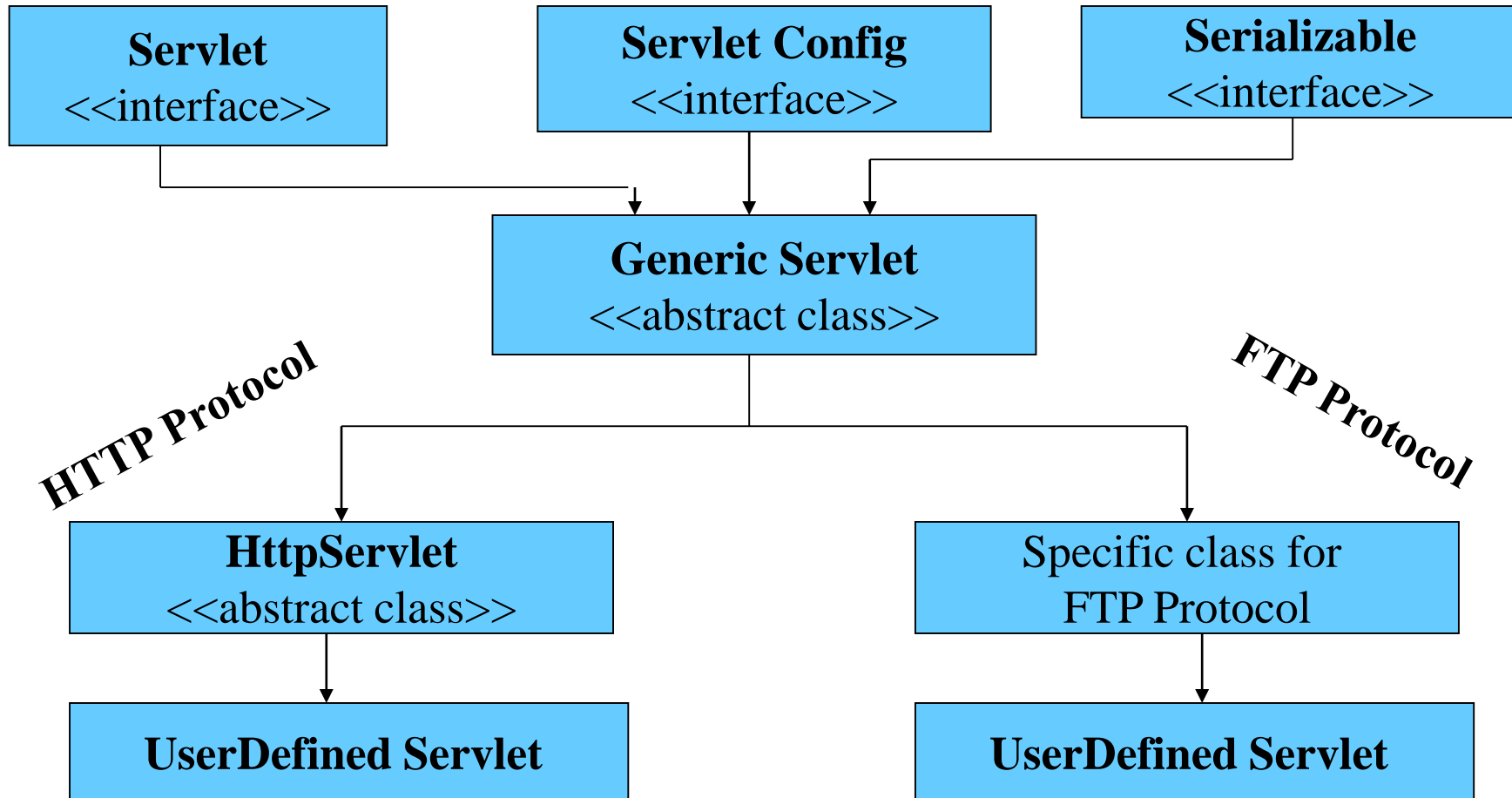


Web Servers

- Tomcat (Apache)
- Weblogic (Bea)
- Microsoft Internet Information Server (IIS)
- Websphere (IBM)
- WebStar (StarNine)
- Java Web Server (Sun's)

Servlet Architecture

Servlet Class Hierarchy



Servlet Architecture contd...

Directory Structure :

➤ **Working Directory**

- html
- jsp
- **WEB-INF**
 - classes
 - lib
 - src

Servlet Architecture contd...

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

```
public class GoodMorning extends HttpServlet {  
    protected void doGet(HttpServletRequest request , HttpServletResponse response) throws  
        IOException {  
        response.setContentType( "text/html" );  
        PrintWriter out = response.getWriter();  
        out.println( "GOOD MORNING EVERYBODY !!!!!" );  
    }  
  
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws  
        IOException {  
        doGet( request, response);  
    }  
}
```

Deployment Descriptor and Deployment

1. Write and add web.xml in WEB-INF.

```
<web-app>
  <servlet>
    <servlet-name>GOODMORNING</servlet-name>
    <servlet-class>Module1.packwelcome.GoodMorning</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>GOODMORNING</servlet-name>
    <url-pattern>/goodmorning</url-pattern>
  </servlet-mapping>
</web-app>
```

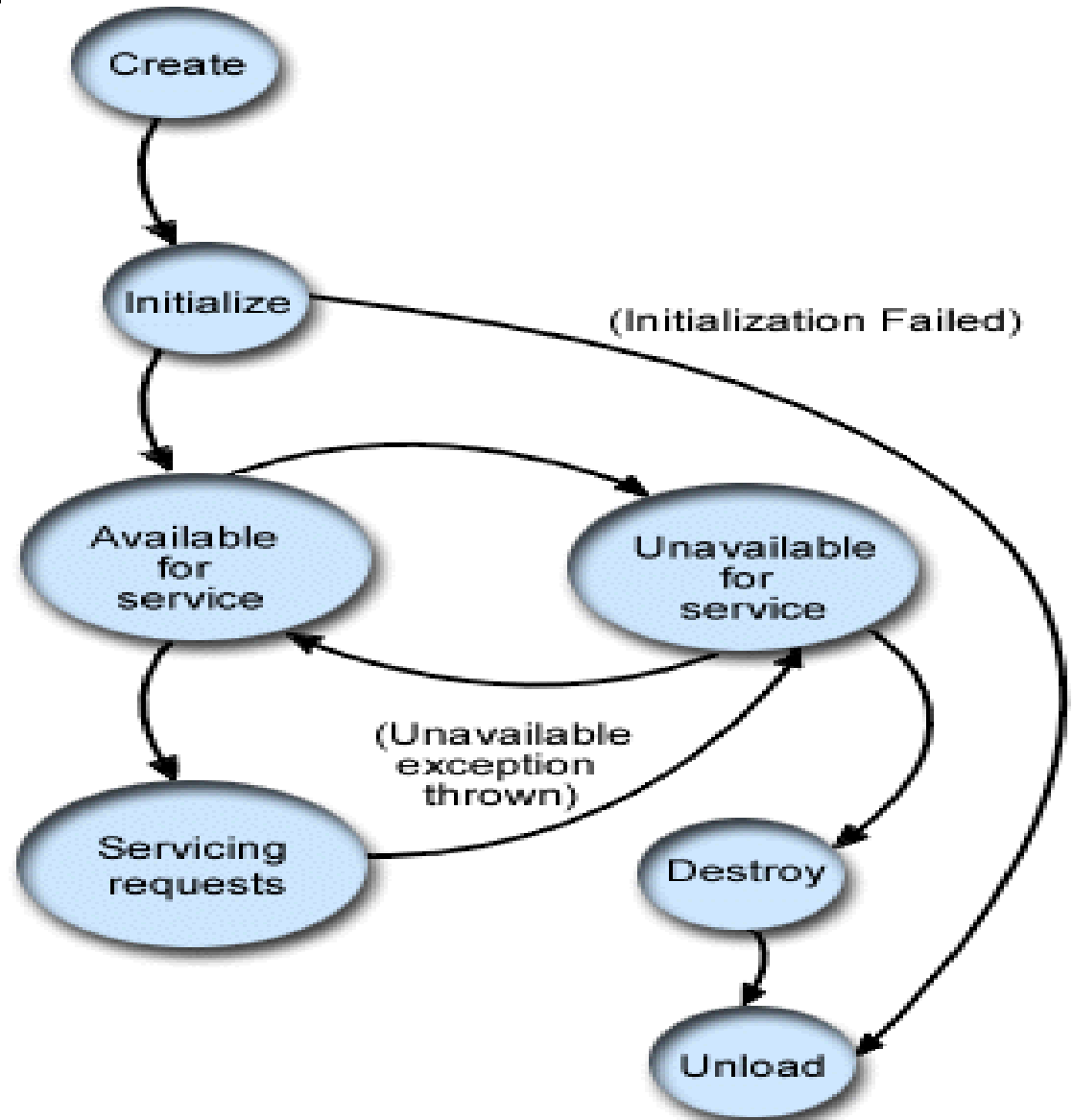
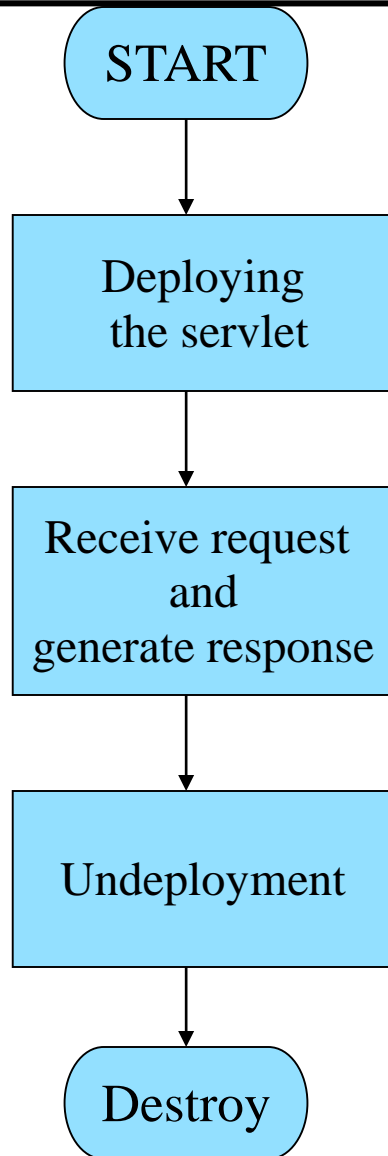
2. Optional : Convert an application into a JAR file.
3. Mandatory : Deploy application on the server.

Module 2 : The Servlet Life Cycle

Overview :

- Servlet Lifecycle
- init(), service() and destroy() methods
- The doGet() and doPost() methods
- The servlet packaging

Servlet Life Cycle



The init() method

```
public void init( ) throws ServletException{
```

```
    f= new File("C:/Count.txt");
```

```
    FileInputStream fi=null;
```

```
    try {
```

```
        if (f.exists( )){
```

```
            fi = new FileInputStream(f);
```

```
            hitCount = fi.read( );
```

```
            fi.close( );
```

```
        }
```

```
        else
```

```
            hitCount = 0;
```

```
    }
```

```
    catch(IOException ie){
```

```
        throw new ServletException("Input file corrupted.", ie);
```

```
    }
```

```
}
```

The service() method

```
protected void doGet ( HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    System.out.println("Servlet receiving subsequent requests" );
    PrintWriter out = response.getWriter();
    out.println(" Pragati Software Private Limited" );
    out.println(" This website is hit "+(hitCount++)+" no. of times." );
}
```

```
protected void doPost (HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException{
    doGet(arg0, arg1);
}
```


The destroy() method

```
public void destroy(){
    System.out.println( " Object being undeployed from server.." );
    try {
        FileOutputStream fo = new FileOutputStream(f);
        fo.write(hitCount);
        fo.close();
    }
    catch(IOException io){
        io.printStackTrace();
    }
}
```

Servlet Packaging

Various ways in which Packaging can be done.....

- .jar file : Java Archive
- .war file : Web Archive
- .ear file : Enterprise Archive

Module 3: FormData, Config and Context Parameters

Overview :

- The Form Data
- Collecting form data for different HTML components
- Roll of XML descriptor
- The 'context' and 'config' parameters
- The Context workspace

The Form Data

The HTML elements

- The FORM ACTION tag
- The INPUT like component tags
- The SUBMIT button tag

The method types

- The GET type of method

`http://localhost:7001/CollectParameters/collect?fname=abc&sname=xyz`

- The POST type of method

The form data is sent on the separate line

Different HTML Components

Creating Form

<FORM ACTION = "/CollectParameters22/param" METHOD = "GET">

Creating Text Box

First Name : <INPUT TYPE="TEXT" NAME = "fname" ">

Creating Password Box

PassWord : <INPUT TYPE="PASSWORD" NAME = "pass">

Creating Radio Buttons

Married<INPUT TYPE="RADIO" NAME="marital" VALUE="Married">

Unmarried<INPUT TYPE="RADIO" NAME="marital" VALUE="Unmarried">

Creating Text Area

<TEXTAREA name='address' COLS="50" ROWS="4"></TEXTAREA>

Different HTML Components (Contd...)

Creating Combo Box

```
<SELECT NAME="city" SIZE="1">  
  <OPTION VALUE="Mumbai">Mumbai</ OPTION >  
  < OPTION VALUE ="Pune">Pune</ OPTION >  
  < OPTION VALUE ="Nasik">Nasik</ OPTION >  
</Select>
```

Creating Check Box

```
Times <INPUT TYPE="CHECKBOX" NAME="news" VALUE="Times ">  
DNA   <INPUT TYPE="CHECKBOX" NAME="news" VALUE="DNA">
```

Creating Submit Button

```
<INPUT TYPE="RESET" VALUE = "RESET">  
<INPUT TYPE="SUBMIT" NAME="choice" VALUE = "Tie">  
<INPUT TYPE="SUBMIT" NAME="choice" VALUE = "Necklace">
```

The HttpServletRequest

The ServletRequest methods :

- String Request.getParameter(String)
- String [] request.getParameterValues(String)
- Enumeration.request.getParameterNames()
- BufferedReader request.getReader()
- ServletInputStream request.getInputStream()

The HttpServletRequest methods :

- String getHeader (String)
- Enumeration getHeaderNames()
- String getMethod()

The HttpServletResponse

The Servlet Response methods :

- `PrintWriter getWriter()`
- `void setContentType(String type)`
- `void setBufferSize(int size)`
- `ServletOutputStream getOutputStream()`

The HttpServletResponse methods :

- `void addHeader(String name, String value)`
- `void setHeader(String name, String value)`
- `void setStatus(int code)`

Collecting Form Data

```
private void doPost(HttpServletRequest arg0, HttpServletResponse arg1) {  
    String[] firstName = arg0.getParameter("fname");           // Text box  
    String surName = arg0.getParameter("sname");               // Text box  
    String passWord = arg0.getParameter("pass");               // Password box  
    String maritalStatus = arg0.getParameter("marital");       // Radio Buttons  
    String yourAddress = arg0.getParameter("address");         // Text Area  
    String yourCity = arg0.getParameter("city");               // Combo box  
    String yourChoice = arg0.getParameter("choice");           // Submit Button  
  
    String [ ]news = arg0.getParameterValues("news");         // Check boxes  
  
    if (news != null){  
        out.println("<TR><TD>News Paper/s");  
        out.println("<TD>"+news[0]);  
        for(int i=1; i<news.length; i++){  
            out.println(" & "+news[i]);  
        }  
    }  
}
```

Collecting Form data contd...

```
Enumeration e = arg0.getParameterNames( );  
while(e.hasMoreElements( )){  
    String field = (String)e.nextElement( );  
    String value = arg0.getParameter(field);  
    out.println(field+" : "+value+"<BR>");  
}
```

The ServletContext

- One per application
- Facilitates communication between server and the servlet
- Provides access to resources and facilities common to all servlets and JSPs in the application
- `getServletContext()` returns a reference to the `ServletContext`

The ServletConfig

- One per Servlet
- Used to store information specific to a particular servlet
- Carries servlet specific data, which is not accessible to any other servlet
- `getServletConfig ()` returns the reference to the ServletConfig

Initializing Parameters

```
<web-app>
```

```
  <!-- Declaring Context Parameters -->
```

```
  <context-param>
```

```
    <param-name>drv </param-name>
```

```
    <param-value>oracle.jdbc.driver.OracleDriver </param-value>
```

```
  </context-param>
```

```
  <!-- Declaring Servlet Tag -->
```

```
  <servlet>
```

```
    <servlet-name>BankEntry</servlet-name>
```

```
    <servlet-class>packbank02.BankEntry</servlet-class>
```

```
    <!-- Declaring Config Parameters -->
```

```
    <init-param>
```

```
      <param-name>user</param-name>
```

```
      <param-value>scott</param-value>
```

```
    </init-param>
```

```
  </servlet>
```

```
</web-app>
```

ServletContext methods

Accessing initial parameters and other server side information.

- `getInitParameter()`
- `getInitParameterNames()`
- `getMajorVersion()`
- `getMinorVersion()`
- `getServerInfo()`

To access server side file resources.

- `getMimeType()`
- `getResourceAsStream()`
- `getRequestDispatcher(String path)`

Handling server side log

- `log()`

ServletContext methods contd...

Accessing context workspace

- `setAttribute()`
- `getAttribute()`
- `getAttributeNames()`
- `removeAttribute()`

ServletConfig methods

Obtaining Config Parameters

- `getInitParameter()`
- `getInitParameterNames()`

Obtaining Context reference

- `getServletContext()`

Obtaining Servlet Name

- `getServletName()`

ServletConfig vs ServletContext

- **Accessibility :**

- **ServletContext** : One per Application , Context parameters are available across servlets under same application.
- **ServletConfig** : One per servlet. The config parameters are private to the servlet and cannot be accessed by any other servlet.

- **Getting the parameter values :**

- **ServletContext sct = this.getServletContext();**
String driverName = sct.getInitParameter("drv");
- **ServletConfig sc = this.getServletConfig();**
String passwd = sc.getInitParameter("pass");

- **Setting attributes :**

- **ServletConfig has only Parameters.**
 - Cannot set the config parameters via methods hence, only getter methods are available.
- **ServletContext has both Parameters and Attributes.**
 - Context Parameters can be set via the setter methods provided

Module 4 : Analysing Request Header

Overview :

- The GET and POST requests
- HTTP Request Parameter
- The Request Headers
- Analyzing request header

Typical Http Request

For GET method :

GET /requestheadersdemo?name=abc&surname=xyz http/1.1

Host : <http://localhost:8081/Myervlets/collectparameters>

User-Agent: Mozilla/4.0

Accept : */*

Accept Encoding : gzip,deflate

For POST method :

POST /requestheadersdemo http/1.1

Host : <http://localhost:8081/Myervlets/collectparameters>

User-Agent: Mozilla/4.0

Accept : */*

Accept Encoding : gzip,deflate

name=abc&surname=xyz

Reading Request Header

General :

getHeader("HeaderName")
getHeaderNames()

Specialized :

getCookies()
getRemoteUser()
getContentLength()
getDateHeader()
getIntHeader()

Related Info :

getMethod()
getRequestURI()
getQueryString()
getProtocol()

Analysing Request Header

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) {  
    PrintWriter out = response.getWriter( );  
    out.println("<B>Request Method : </B>" + request.getMethod( ) + "<BR>");  
    out.println("<B>Request Protocol : </B>" + request.getProtocol( ) + "<BR>");  
    out.println("<B>Request URI : </B>" + request.getRequestURI( ) + "<BR>");  
  
    Enumeration headerNames = request.getHeaderNames( );  
    while(headerNames.hasMoreElements( )){  
        String headerName =(String)headerNames.nextElement( );  
        out.println("<TR><TD>" + headerName);  
        out.println("    <TD>" + request.getHeader(headerName));  
    }  
}
```

Module 5 : Setting Response Header

Overview :

- HTTP Response Header
- Setting MIME types
- Status Code, Refreshing, Sending Error Page, Logging in
- Setting Encoding Type

Typical HttpResponse

HTTP Response Message

status line
(protocol
status code
status phrase)

header
lines

data, e.g.,
requested
HTML file

HTTP/1.1 200 OK

Connection: close

Date: Thu, 06 Aug 1998 12:00:15 GMT

Server: Apache/1.3.0 (Unix)

Last-Modified: Mon, 22 Jun 1998 ...

Content-Length: 6821

Content-Type: text/html

data data data data data ...

Setting Response Header

```
import javax.servlet.*;
import javax.servlet.http.*;
public class ResponseHeaders extends HttpServlet{
    public void doGet(HttpServletRequest req , HttpServletResponse res)throws
        ServletException , IOException{
        res.setContentType("text/html");
        PrintWriter out = res.getWriter( );
        res.addHeader("MyHeader" , "This is My custom header");
        out.println("<html>");
        out.println("<body>");
        out.println("My Custom Header Set");
        out.println("</body>");
        out.println("<html>");
    }
}
```


Various Response Headers

- Allow
- Cache-Control
- Connection
- Content-Encoding
- Content-Language
- Content-Length
- Content-Type
- Refresh

Setting MIME type

Method to set MIME type :

setContentTypes(Mime type)

List of all mime types :

- text/plain : Plain text
- text/html : HTML document
- text/xml : XML document
- audio/basic : Sound file in .snd format
- image/gif : GIF image
- image/jpeg : JPEG image
- application/msword : Microsoft Word Document
- application/pdf : Acrobat (.pdf) file .
- application/x-java-archive : JAR file
- video/mpeg : MPEG video clip

Setting Encoding Type

```
public class Gzipping extends HttpServlet {
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1)
        throws ServletException, IOException {
        String encodings = arg0.getHeader("Accept-Encoding");
        PrintWriter out;
        String title;
        arg1.setContentType("text/plain");
        if ((encodings != null) && (encodings.indexOf("gzip") != -1)) {
            title = "Page Encoded with GZip";
            OutputStream out1 = arg1.getOutputStream();
            out = new PrintWriter(new GZIPOutputStream(out1), false);
            arg1.setHeader("Content-Encoding", "gzip");
        }
        else {
            title = "Un-encoded Page.";
            out = arg1.getWriter();
        }
        for(int i = 0; i < 10000; i++)
            out.println("-foo-faa");
        out.close();
    }
}
```

Module 6 : The Redirection

Overview :

- Redirection
- Redirection by setting response header
- Linking to another websites

Redirection in HTML

```
public class Redirect extends HttpServlet{  
    protected void doGet(HttpServletRequest request, HttpServletResponse response)throws  
        ServletException, IOException{  
        String url1 = "http://localhost:8081/MyServlet/readword";  
        String url2 = "http://www.pragatisoftware.com";  
        String url3 = "/MyServlet/goodmorning";  
  
        // HTML Hyper Link Way of Redirection  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<html><head><title>301 Moved Permanently</title></head>");  
        out.println("<body>");  
        out.println("Website moved to <a href='\""+url1+"\">here.</a>");  
        out.println("</body></html>");  
        out.close();  
    }  
}
```

Redirection using Servlet API

```
protected void doGet(HttpServletRequest request , HttpServletResponse response)  
    throws ServletException, IOException{
```

```
    PrintWriter out = res.getWriter();  
    out.println("<B><CENTER> *****WELCOME*****");
```

```
// Redirection by specifying the absolute path of the site to whom the request is directed  
    res.sendRedirect(url2);
```

```
// Redirection by specifying the relative path of the site to whom the request is directed  
    res.sendRedirect(url3);
```

```
// Redirection by setting response header  
    res.setStatus(HttpServletResponse.SC_MOVED_TEMPORARILY);  
    res.setHeader("Location", url1);  
}
```

Setting the Refresh Header

```
public class RefreshHeader extends HttpServlet {
    String url = "www.pragatisoftware.com";

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {

        PrintWriter out = response.getWriter();
        out.println("<H1>HELLO !!!!!<H1>");

        //The Browser asks for an updated page after 30 seconds
        response.setIntHeader("Refresh",10);

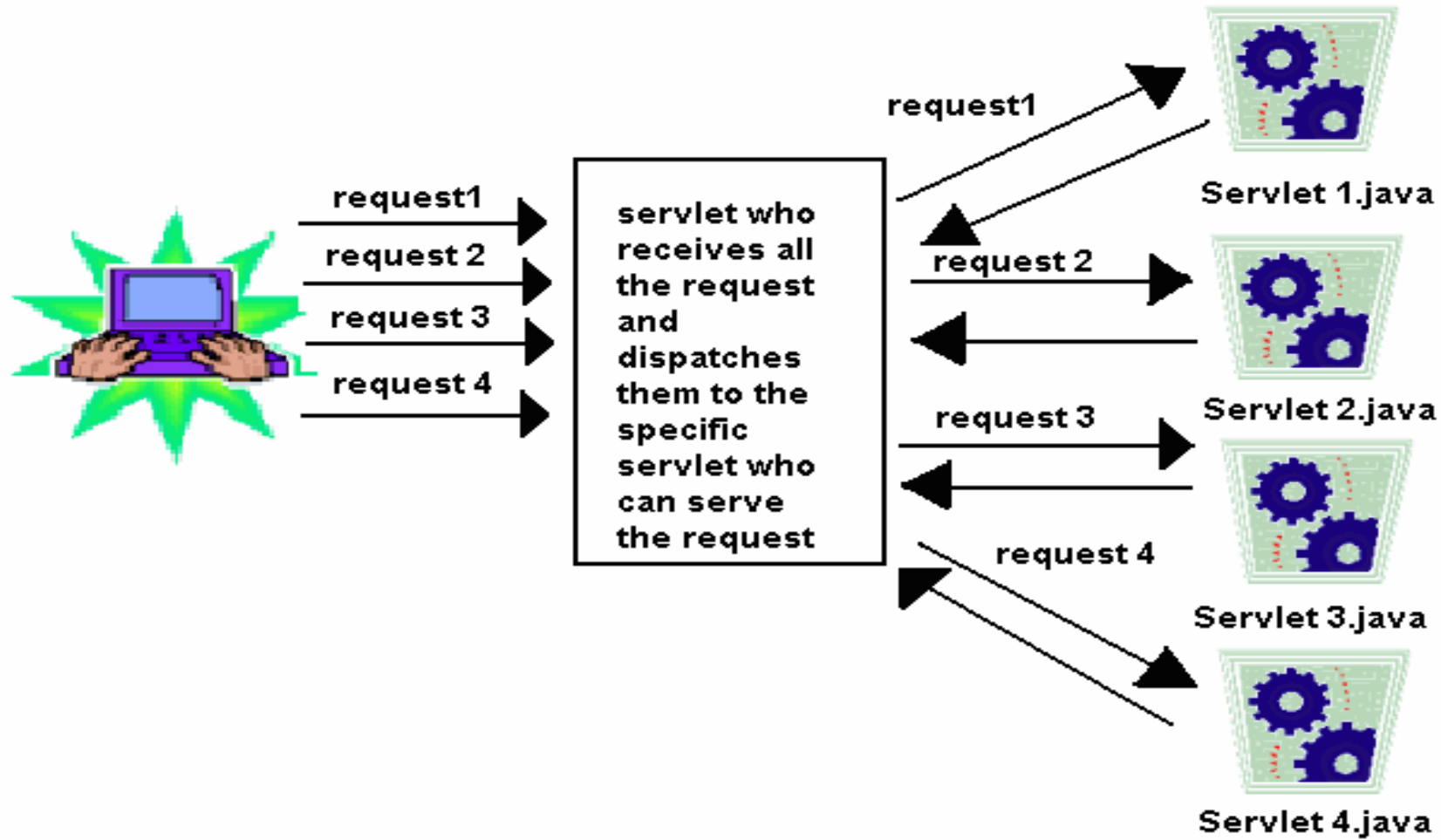
        //The browser goes to the specified page after specified time delay
        response.setHeader("refresh","5; URL=http://www.pragatisoftware.com");
    }
}
```

Module 7 : The Request Dispatching

Overview

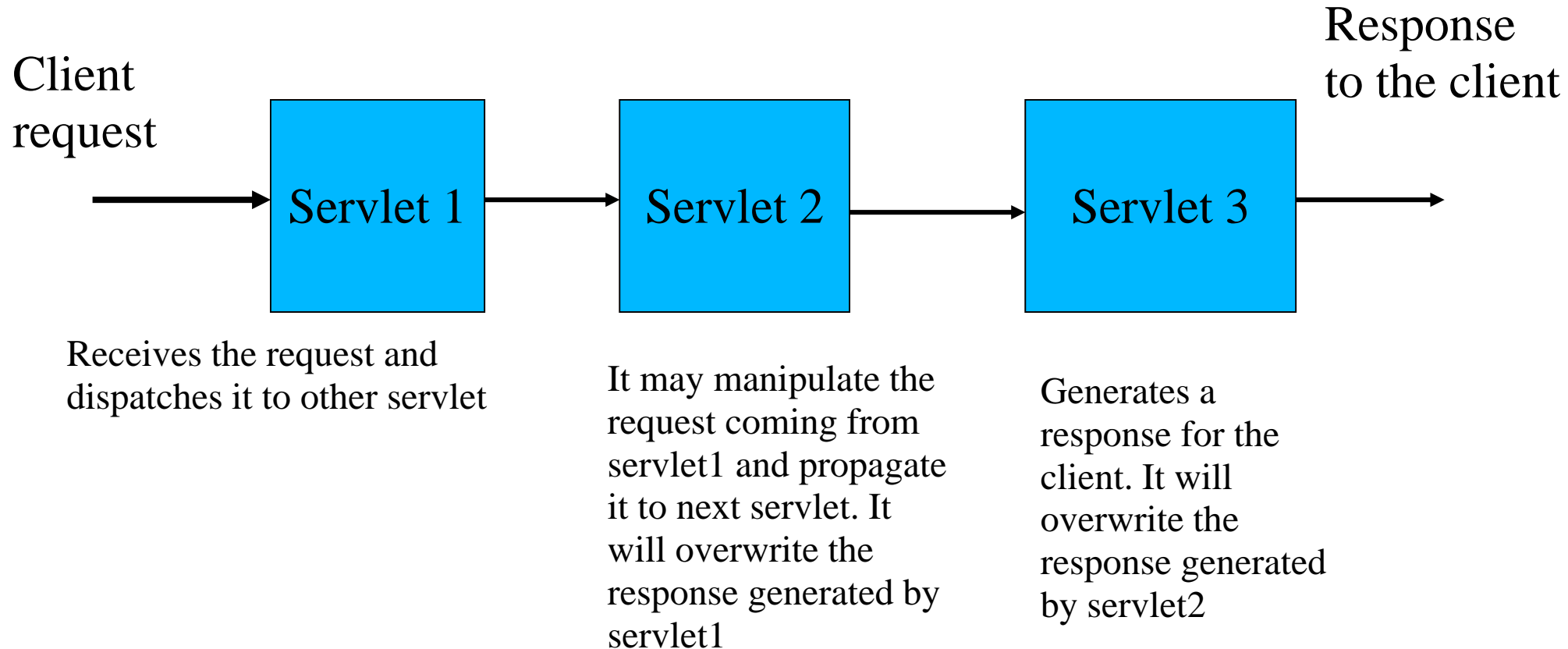
- The Request Dispatcher
- The forward and include requests
- Sending request between servlets
- The request workspace

Request Dispatcher



Dispatching using forward() method

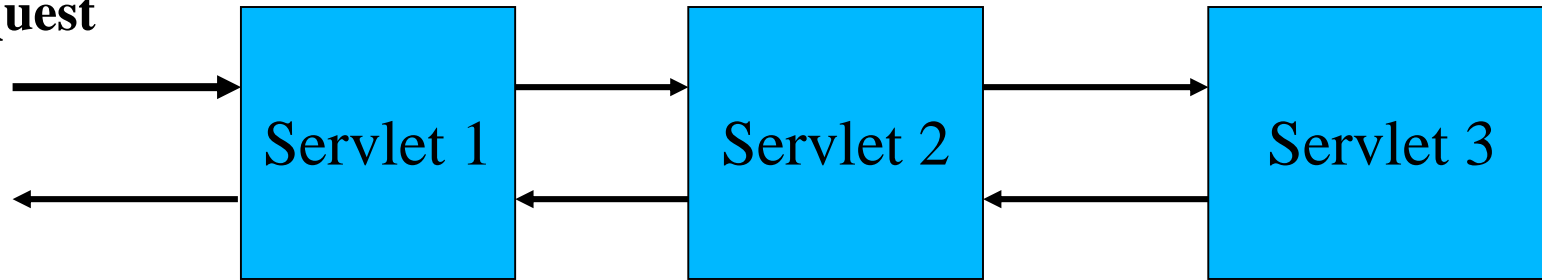
The request is received by one servlet and response is generated by another.



Dispatching using include() method

The request is received by one servlet and response is generated by same after including responses from other servlets .

**Client
request**



**Response
generated**

Receives the request and
dispatches it to other servlet

Generates an response
and dispatches to servlet
3

Adds some more
data to the
response and
sends back to
servlet2

MainPage.java

```
protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1) throws
    ServletException, IOException {
    PrintWriter out = arg1.getWriter();
    out.println( "Pragati Software Pvt Ltd." );
    String accounttype = arg0.getParameter("accounttype");
    RequestDispatcher rd;
    if (accounttype.equalsIgnoreCase("Current")){
        rd = arg0.getRequestDispatcher("/current");
        rd.forward(arg0, arg1);
    }
    else {
        rd = arg0.getRequestDispatcher("/saving");
        rd.include(arg0, arg1);
    }
    out.println("Lok Center , Marol - Maroshi Road , Marol , Andheri(E) .");
}
```

Module 8 : The Cookies

Overview :

- Pros and Cons of using Cookies
- Session and Persistent Cookies
- Sending and Receiving Cookies
- Application of Cookies

Preserving client data

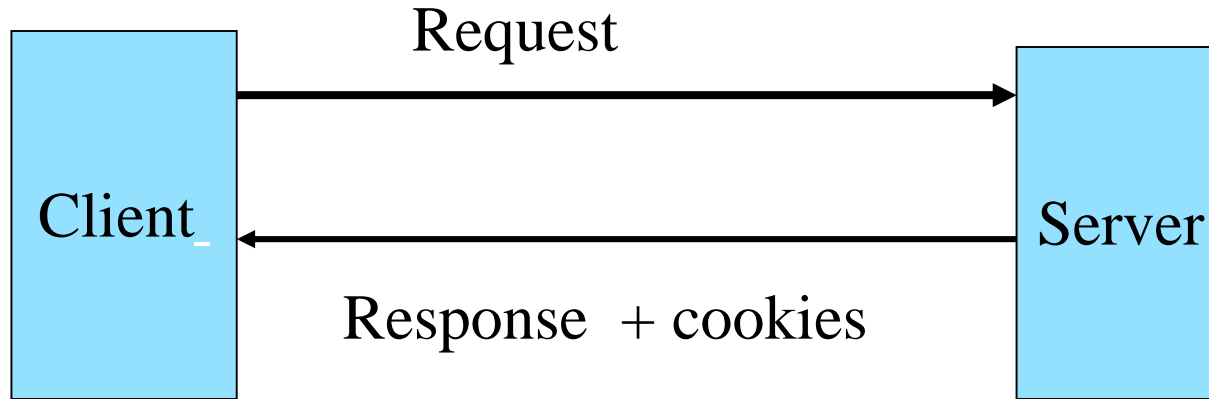
- At the server side in database
- At the server side in any other storage
 - Text files
 - Serialized form
- At the client side in the form of cookies

More about Cookies

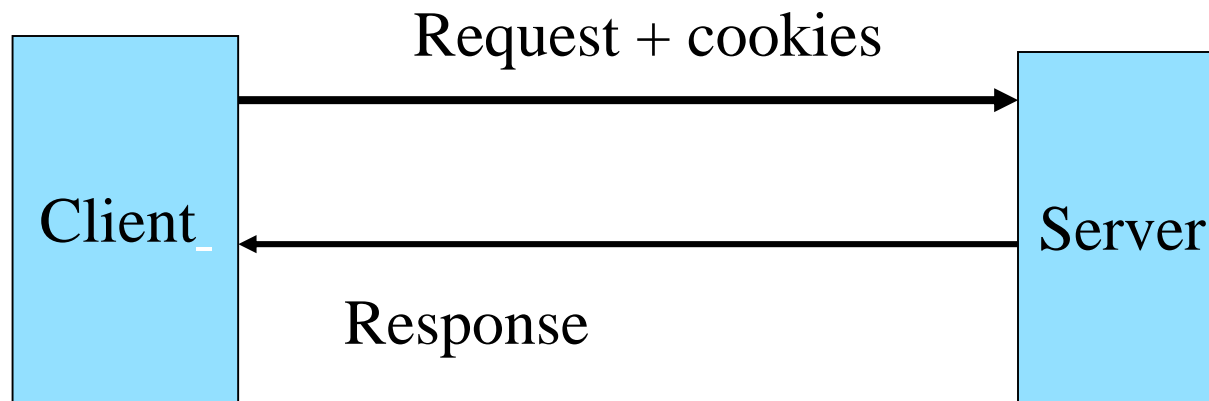
- Cookies is:
 - a textual information stored at client's machine by the server.
 - neither a virus nor carrier of a virus.
 - cannot contain Executable code.
 - carries less confidential client specific information.
- Browser restricts:
 - 20 cookies for a website.
 - 300 cookies for a browser instance.
 - 4kb size for a single cookie.

Working of Cookies :

A cookie generation:



The next request:



Creating Cookies

- Three steps to create a new cookie:
 - 1) Create a new Cookie Object
 - **Cookie cookie = new Cookie (name, value);**
 - 2) Set cookie attributes
 - **cookie.setMaxAge (60);**
 - 3) Add your cookie to the response object:
 - **Response.addCookie (cookie)**

Creating Cookies (contd...)

```
public class CreateCookies extends HttpServlet{
    protected void doGet(HttpServletRequest arg0, HttpServletResponse arg1) throws
        ServletException, IOException{
        for(int i=0; i<3; i++){
            Cookie cook = new Cookie("Session"+i, "Value "+i);
            arg1.addCookie(cook);// This is a cookie with age for a session only.
            Cookie cook1 = new Cookie("Persistent"+i, "Value "+i);
            if (i==0)
                cook1.setMaxAge(60);
            if (i==1)
                cook1.setMaxAge(120);// Age set for two minutes.
            if (i==2)
                cook1.setMaxAge(60*60);
            arg1.addCookie(cook1);// This is a cookie persistent for an hour.
        }
        PrintWriter out = arg1.getWriter();
        out.println("To see the cookies, visit the");
        out.println("<AHREF = \"/MyServlets/showcookies\">SHOWCOOKIES</A>");
    }
}
```

Reading Cookies

```
public class ShowCookies extends HttpServlet {  
    protected void doGet ( HttpServletRequest arg0, HttpServletResponse arg1) throws  
        ServletException, IOException {  
        PrintWriter out = arg1.getWriter( );  
        Cookie [ ] cookies = arg0.getCookies( );  
        for(int i= 0 ; i < cookies.length; i++){  
            out.println( "<TR>"+ " <TD>"+ cookies[i].getName()+  
                " <TD>"+ cookies[i].getValue() ) ;  
        }  
    }  
  
    protected void doPost(HttpServletRequest arg0, HttpServletResponse arg1) throws  
        ServletException, IOException{  
        doGet(arg0, arg1);  
    }  
}
```

Applications of cookies

- Customizing the sites
- Focused advertising
- Storing information about the client

Applications of cookies (contd...)

```
public class RepeatVisitor extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        boolean newVisitor = true;
        Cookie[ ] cookies = request.getCookies();
        if(cookies != null){
            for(int i =0;i<cookies.length;i++){
                Cookie c = cookies[i];
                if(c.getName().equals("repeatVisitor") && c.getValue().equals("yes")){
                    newVisitor = false;
                    break;
                }
            }
        }
    }
}
```

Applications of cookies (contd...)

```
String title , color;
if(newVisitor) {
    Cookie returnVisitorCookie = new Cookie("repeatVisitor","yes");
    returnVisitorCookie.setMaxAge(60*60*24*365); // persistent for 1 year
    response.addCookie(returnVisitorCookie);
    title = "WELCOME !!!!!!!";
    color = "skyblue";
}
else {
    title = "Nice to see you again !!!!!!!";
    color = "pink";
}
PrintWriter out = response.getWriter();
out.println("<html>\n" + " <head><title>" + title + "</title></head>\n" +
    "<body bgcolor = \"" + color + ">\n" + " <h1 align=\"center\">" + title + "</h1>\n" +
    "</body></html>");
}
```

Disadvantages of cookies

- Browsers block cookies
- Threat to privacy

Module 9 : The Session Tracking

Overview :

- Session Object
- Setting and getting attributes
- Session Tracking API
- Browser and Server sessions

Need of Session Tracking

- To identify a user if he re-visits your site.
- To maintain the state , when there are series of requests from the same user.

Session Handling

- Accessing session object associated with current request :
 - `request.getSession();`
- Looking up information associated with a session :
 - `session.getAttribute();`
- Storing information in a session :
 - `session.setAttribute();`
- Discarding Session :
 - `session.invalidate();`

Various Ways of Session Tracking

- Cookies
- URL Rewriting
- Hidden Form Fields

The Hidden Form Fields

The HTML :

```
<html> <body>
<form action = "/MyServlet/hiddenfield" Method = POST align =Center>
USERNAME : <input type = text name = "user" align = center>
<input type = submit value = "Login" align = center>
</form> </body> </html>
```

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

```
        String username = request.getParameter("user");
        PrintWriter out = response.getWriter();
        out.println("Hello !! click Submit to proceed further");
        out.println("<form action =\"/MyServlet/secondservlet\">");
        out.println("<input type =\"hidden\" name =\"user\" value =\""+username+">");
        out.println("SURNAME : <input type =\"text\" name = \"surname\" );
        out.println("<input type =\"Submit\" value = \"Submit\"</form>");
```

```
}
```

The Hidden Form Fields (contd...)

```
public class SecondServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        doPost(request,response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        String uname = request.getParameter("user");
        String surname = request.getParameter("surname");
        PrintWriter out = response.getWriter();
        out.println("Hello !!! "+uname);
    }
}
```

The URL Rewriting

```
public class URLRewriting extends HttpServlet{
```

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

```
        response.setContentType("text/html");
```

```
        PrintWriter out = response.getWriter();
```

```
        String str = "/MyServlet/SessionTracking";
```

```
        HttpSession ses = request.getSession();
```

```
        String encodedURL = response.encodeURL(str);
```

```
        out.println("test session");
```

```
        out.println("<html><body>");
```

```
        out.println("<form action =\"+encodedURL+\">");
```

```
        out.println("<input type =\"Submit\" value = \"Submit\"</form>");
```

```
        out.println("</body></html>");
```

```
    }
```

```
}
```

The Cookies

- Can pass information:
 - For both GET and POST type of requests.
 - Even if application has static web pages.
- Cannot pass information:
 - If Browser blocks cookies.
 - If web page is bookmarked

The Session Tracking API

Provides a simple interface to manage the session automatically.

- **The API manages session through:**
 - Cookies
 - URL rewriting
- **The API can do:**
 - Provides implicit session object
 - Creates unique session ID
 - Tracks session using session ID
 - Provides listeners for implicit session objects
- **API Hierarchy :**
 - public interface HttpSession

Methods of HttpSession

- `isNew()`
- `invalidate()`
- `getAttribute(String name)`
- `setAttribute(String name)`
- `getMaxInactiveInterval()`
- `setMaxInactiveInterval()`

The Session creation

```
public class SessionTracking extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Test session");
        HttpSession ses = request.getSession();
        if(ses.isNew()){
            out.println("This is a new Sesion");
        }
        else{
            out.println("Welcome back");
        }
    }
}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    doGet(request , response);
}
}
```

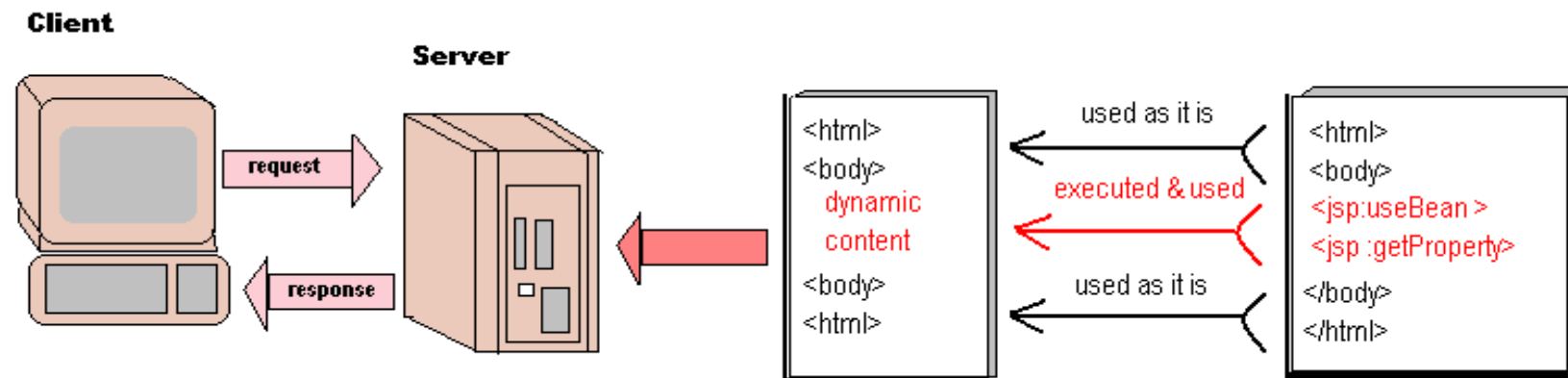
Module 10 : The JSP Technology

Overview :

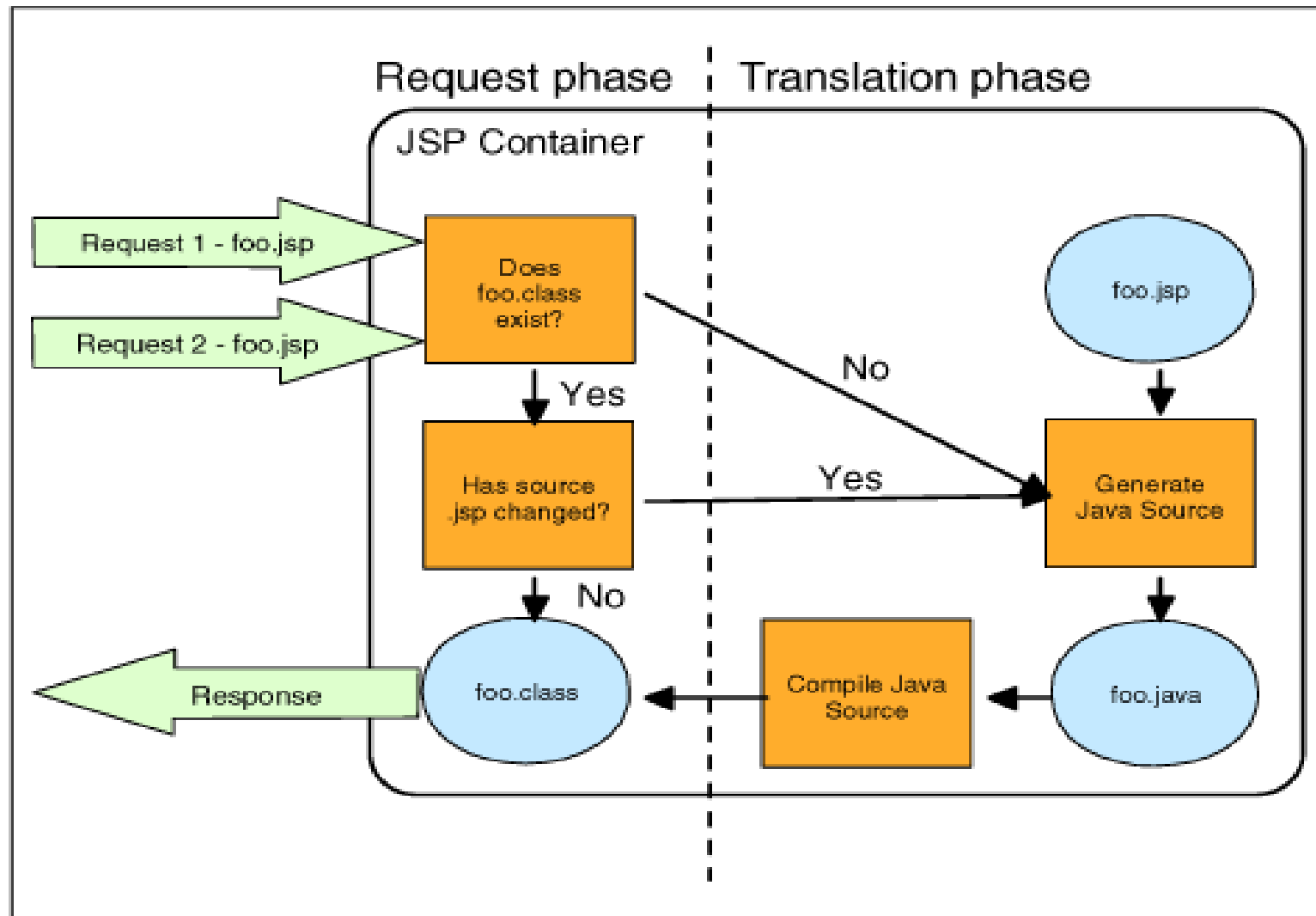
- Introduction to JSP Technology
- Writing and deploying first .jsp program
- Basic Syntax of JSP Elements
- JSP Life Cycle
- JSP scripting elements: Declarations, Scriptlets and Expressions

Introduction to JSP

JSP (Java Server Pages)



How JSP works ?



The First.jsp

```
<html>
```

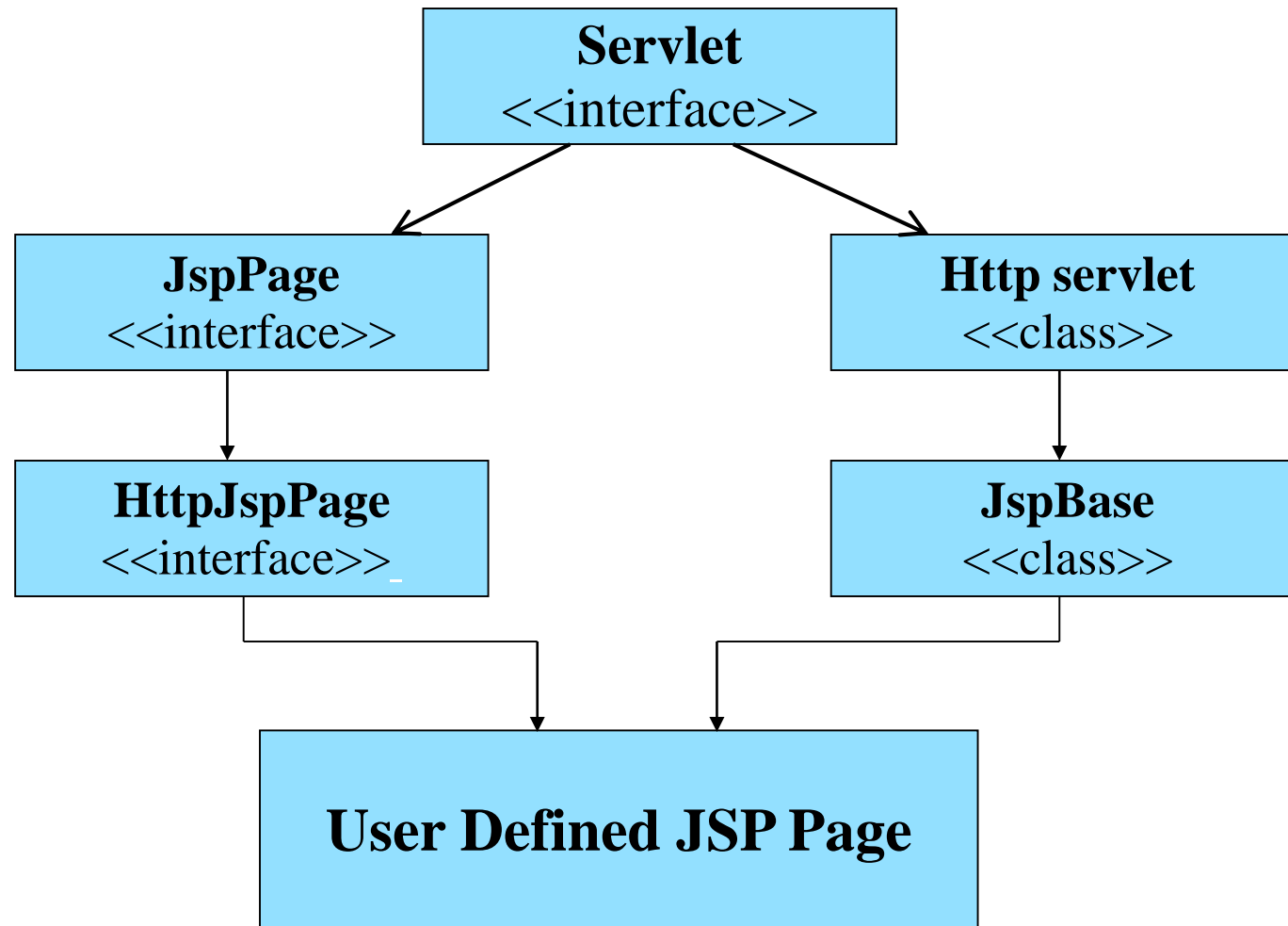
```
<body>
```

```
    WELCOME!!!!!!
```

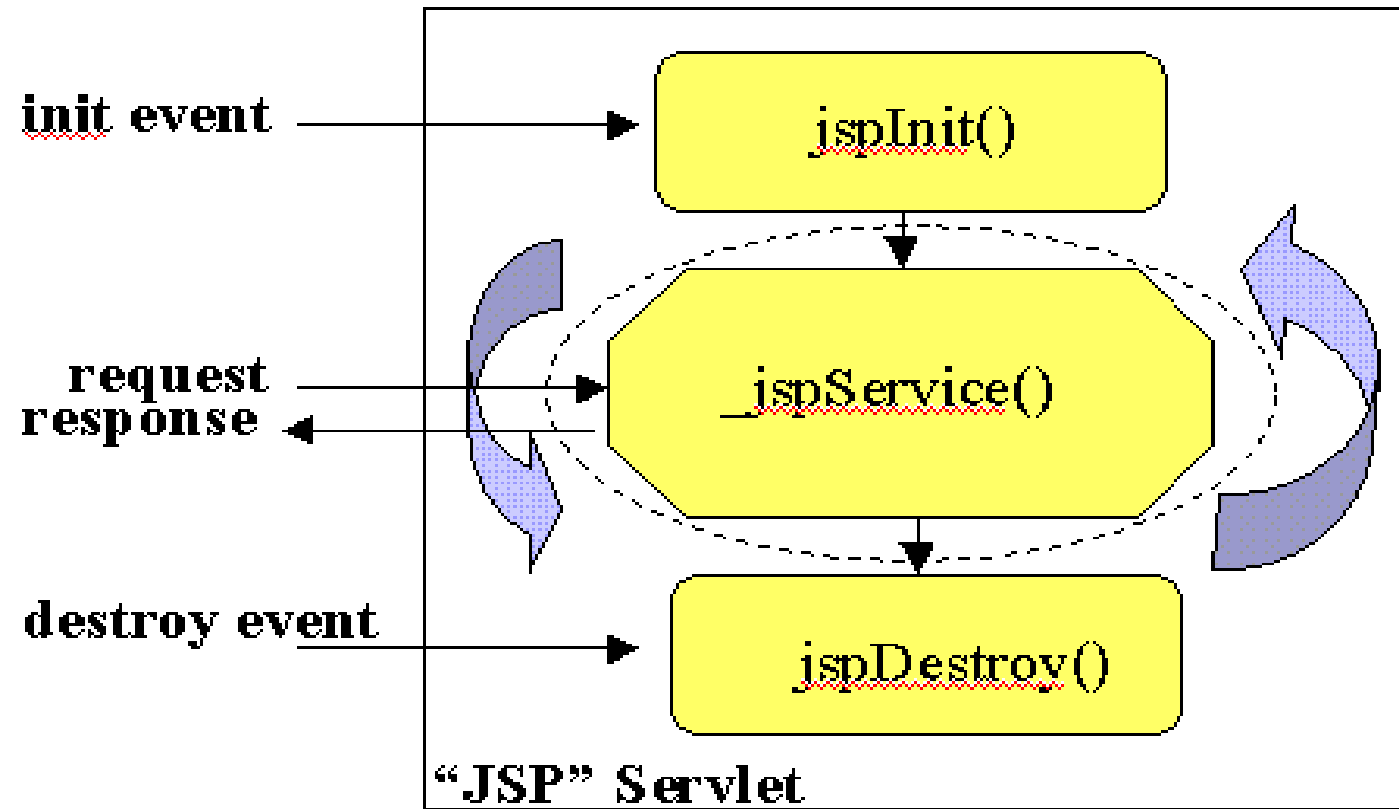
```
</body>
```

```
</html>
```

The JSP API



The JSP LifeCycle



The Elements of JSP

```
<%@ page import = "packjava.Counter" %>
```

Directives

```
<html>
```

```
<body>
```

```
<%
```

Scriptlets

The page count is

```
out.println(Counter.getCount());
```

```
%>
```

```
</body>
```

```
</html>
```

```
package packjava;
    public class Counter{
        private static int count;
        public static int getCount(){
            count++;
            return count;
        }
    }
```

The Elements of JSP (Contd...)

```
<%@ page import ="packjava.* "%>
```

```
<html>
```

```
<body>
```

```
<%
```

The page count is

```
<%= Counter.getCount() %>
```

```
%>
```

```
</body>
```

```
</html>
```



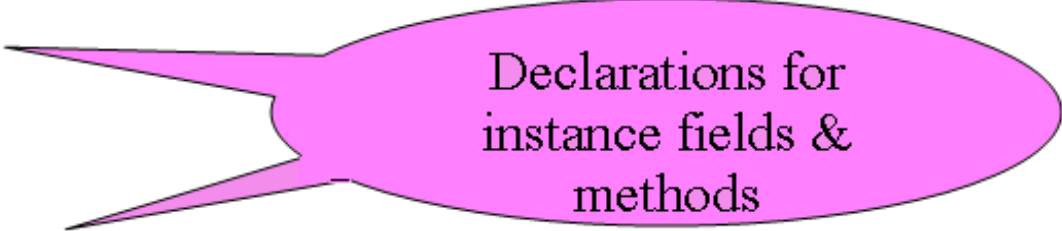
Expressions

```
package packjava;  
    public class Counter{  
        private static int count;  
        public static int getCount(){  
            count++;  
            return count;  
        }  
    }
```

The Elements of JSP (Contd...)

```
<%! static int count = 0 %>
```

```
<%! public void sayHello(){  
    out.println("Hello you are the " +  
    count + "visitor to the site");  
    count++;  
}  
%>
```



Declarations for
instance fields &
methods

```
public class Counter extends HttpServlet {  
    static int count =0;  
    public void sayHello(){  
        ---- ;  
    }  
    public void _jspService(HttpServletRequest req,HttpServletResponse res)  
        throwsServletException{  
        ----- ;  
    }  
}
```

Various Elements in a JSP page

- **HTML Comment** : `<!-- -->`
- **JSP Comment** : `<%-- --%>`
- **JSP Scriptlets** : `<% %>`
- **JSP Directives** : `<% @ %>`
- **JSP Expressions** : `<% = %>`
- **JSP Declaration** : `<%! %>`

Configuring init parameters for JSP

```
<web-app>
```

```
....
```

```
<servlet>
```

```
  <servlet-name>MyJsp</servlet-name>
```

```
  <jsp-file>/MyJsp.jsp</jsp-file>
```

```
  <init-param>
```

```
    <param-name>email</param-name>
```

```
    <param-value>admin@yahoo.com</param-value>
```

```
  </init-param>
```

```
</servlet>
```

```
.....
```

```
</web-app>
```

Module 11: Predefined Variables and Page Directives

Overview :

- Using predefined variables
- Handling response using response variable
- Using Page directives

JSP predefined variables

Implicit Objects :

- | | | |
|------------------------|--------|-------------|
| 1. JspWriter | —————→ | out |
| 2. HttpServletRequest | —————→ | request |
| 3. HttpServletResponse | —————→ | response |
| 4. HttpSession | —————→ | session |
| 5. ServletContext | —————→ | application |
| 6. ServletConfig | —————→ | config |
| 7. PageContext | —————→ | pageContext |
| 8. Object | —————→ | page |

The page Directives

- Attributes to the page directive
 - import
 - errorPage
 - isErrorPage
 - contentType
 - extends

Module 12 : Including Files

Overview :

- Compile time and Runtime inclusion
- The 'include' action and 'include' directive
- Forwarding request/including response
- Pros and cons of Compile and Runtime inclusions

Compile Time File Inclusion

Include Directive :

Includes the specified page during translation phase.

➤ **Syntax :**

```
<%@include file="/JSPPageToBeIncluded.jsp" %>
```

Including pages at runtime

- **The jsp:include Action :**

- `<jsp:include page="URL" flush="true">`

- **The page Attribute :**

- It specifies the relative URL of the page whose output should be included.

- **The flush Attribute :**

- This attribute is optional.

- Specifies whether the output stream of the main page should be flushed before the inclusion of page.

The <jsp:forward> Tag

- **Propagates the request object from one JSP to another JSP or servlet.**
- **Syntax :-**
 - `<jsp:forward page = “ URL ” >`
- **Example :-**
 - `<jsp:forward page="/servlet/login" />`
 - `<jsp:forward page="/servlet/login">
 <jsp:param name="username" value="jsmith" />
</jsp:forward>`

The “param” element

- **Syntax :**

- `<jsp:param name= “paramName” value=“paramValue”>`

- **Used with include and forward actions**

- `<jsp:include page=“URL” flush=“true”>`

- `<jsp:param name=“paramname” value=“paramvalue”>`

- `</jsp:include>`

Differences in compile time and runtime file inclusion

	jsp : include Action	Include Directive
Syntax	<code><jsp:include page=“...”/></code>	<code><%@ include file=“..”%></code>
Time of inclusion	Request Time	Page Translation time
What is included	Output of included page	Actual code of the file
Number of Servlets generated	2 (main page and included page each become separate servlet)	1 (included file is inserted into main page, that page is translated into a servlet)

The Pro's and Cons

- Maintenance
- Speed

Module 13: The 'useBean' Tag in JSP

Overview :

- Building and accessing Beans
- Collecting Form Data using Beans
- Sharing Beans

Collecting Form Parameters using JSP

```
<HTML>
<BODY>
  <TITLE>JSP Collect Parameters</TITLE>
  <%= request.getParameter("flights")%>
  <%
    String val1 = request.getParameter("emailAddress");
    String val2 = request.getParameter("origin");
    String val3 = request.getParameter("desti");
    String val4 = request.getParameter("totCost");
    String val5 = request.getParameter("advance");
  %>
  Information Details : <BR>
  Email Address   :   <%=val1%>
  Origin          :   <%=val2%>
  Destination     :   <%=val3%>
  Total Cost      :   <%=val4%>
  Advance Paid    :   <%=val5%>
</BODY>
</HTML>
```

The “useBean” Tag

- The “useBean” Tag :
 - **<jsp : useBean id : “idName” class = “beanClass” type = “refType” scope = “variableScope” />**
- The setProperty :
 - **<jsp:setProperty name= “idName” property= “propertyName” value = “value” >**
- The getProperty :
 - **<jsp:getProperty name = “ idName” property = “propertyName” >**

Collecting Form Data using “useBean” tag

```
<jsp:useBean id = “ entry ” class = “ pack020param.TravAgentBean ” />
```

Information Details :

Email Address	:	<jsp : getProperty name = “ entry ” property= “ emailAddress ” />
Origin	:	<jsp : getProperty name = “ entry ” property= “ origin ” />
Destination	:	<jsp : getProperty name = “ entry ” property= “ destination ” />
Total Cost	:	<jsp : getProperty name = “ entry ” property= “ totalCost ” />
Advance Paid	:	<jsp : getProperty name = “ entry ” property= “ advance ” />

Setting values using “useBean” Tag

```
<jsp:useBean id = "entry" class = "pack020param.TravAgentBean" />
```

```
<% -- 1 --- %>
```

```
<jsp:setProperty name = "entry" property = "emailAddress"  
    value = '<%=request.getParameter("emailAddress") %>' />
```

```
<%-- 2 --%>
```

```
<% float tcost = 1.0f; tcost = Float.parseFloat(request.getParameter("totalCost")); %>
```

```
<jsp:setProperty name = "entry" property = "totalCost" value = "<%= tcost %>" />
```

```
<%-- 3 --%>
```

```
<jsp:setProperty name = "entry" property = "destination" param = "desti" />
```

```
<jsp:setProperty name = "entry" property = "totalCost" param = "totCost" />
```

```
<%-- 4 --%>
```

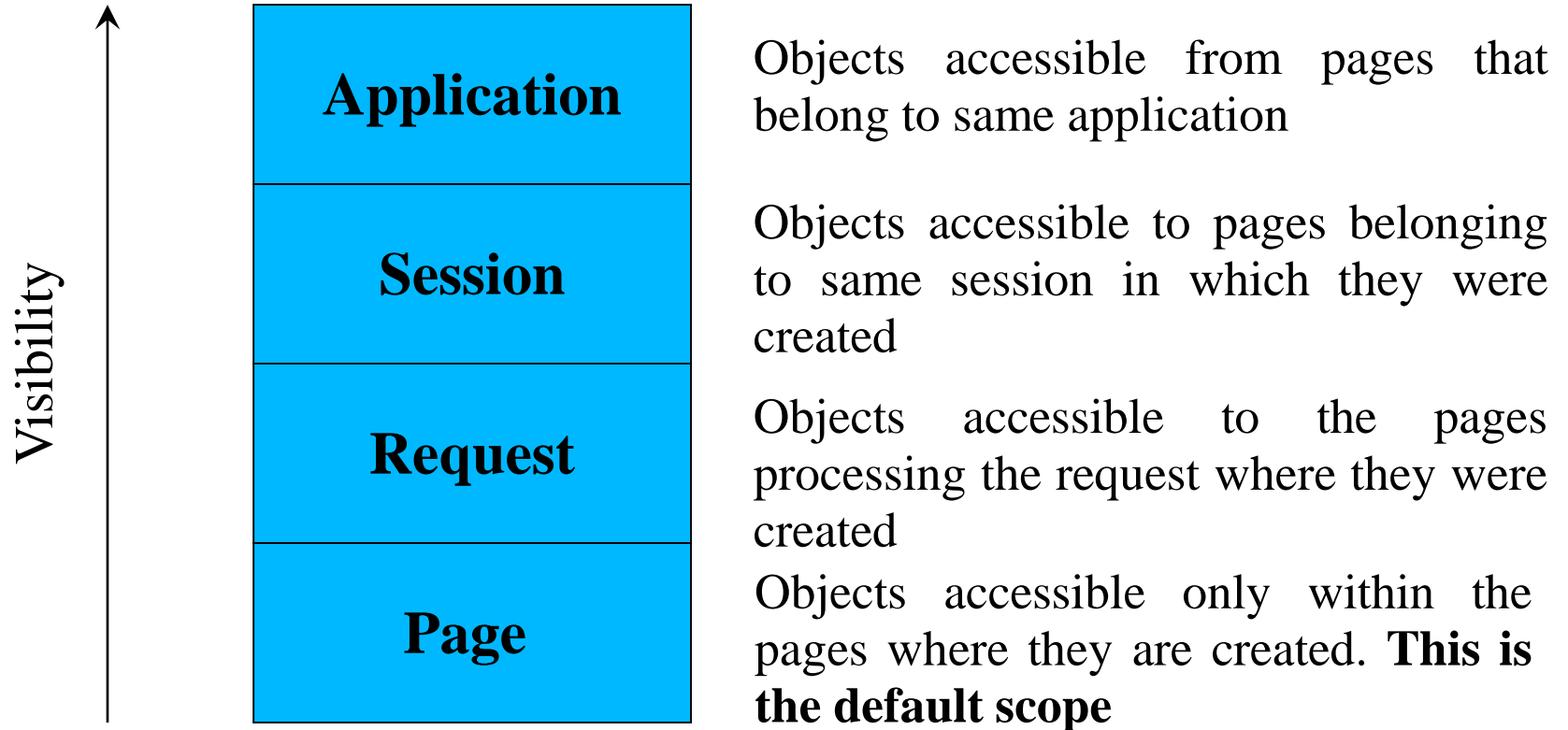
```
<jsp:setProperty name = "entry" property = "advance" />
```

```
<jsp:setProperty name = "entry" property = "origin" />
```

```
<%--5--%>
```

```
<jsp:setPropert name = "entry" property = " * " />
```

Sharing Beans



```
<jsp:setProperty name = "entry" property = "advance" scope = "request"/>
```

Module 14: The Expression Language (EL)

Overview :

- Accessing properties, arrays and collections
- The EL Implicit objects
- The EL Scopes
- Few other EL Operators

Importance of EL

- It simplifies the presentation layer by replacing hard-to-maintain Java scripting elements with short and readable entries.
- **Syntax :**
\$ { expression }

Why EL?

- Concise access to stored objects.
- Shorthand notation for bean properties.
- Simple access to collection elements.
- Access to request parameters, cookies and other request data.
- Small but useful set of simple operators.
- Conditional output.
- Automatic type conversion.
- Empty values instead of error messages.

The Implicit Object references in EL

- **Map of Scope Attributes :**
 - pageScope
 - requestScope
 - sessionScope
 - applicationScope
- **Maps of request parameter :**
 - param
 - paramValues
- **Maps of RequestHeader :**
 - header
 - headerValues
- **Map of context init parameter :**
 - initParam
- **PageContext**
 - Actual reference to pageContext object

Accessing Implicit Objects

Accessing Context Information

`${pageContext.servletContext.serverInfo }`

`${pageContext.servletContext.majorVersion }`

`${pageContext.servletContext.minorVersion }`

`${pageContext.servletContext.servletContextName}`

Accessing session information

`${pageContext.session.id }`

`${pageContext.session.lastAccessedTime }`

`${pageContext.session.creationTime }`

Accessing config information

`${pageContext.servletConfig.servletName }`

Accessing request information

`${ pageContext.request.method }`

`${ pageContext.request.contextPath }`

Accessing Bean Properties

`<%@ page isELIgnored='false' %>`

Accessing Bean properties using the “Bean” object with session scope

Email Address : \${ Bean.emailAddress }
Destination : \${ Bean.destination }
Total Cost : \${ Bean.totalCost }
Advance Paid : \${ Bean.advance }

Accessing an ArrayList using the “WeakNames” reference with session scope

\${ WeekNames[0] } \${ WeekNames[1] }
\${ WeekNames[2] } \${ WeekNames[3] }
\${ WeekNames[4] } \${ WeekNames[5] }
\${ WeekNames[6] } \${ WeekNames[7] }

Accessing an Array using the “MonthNames” reference with session scope

\${ MonthNames[0] } \${ MonthNames[1] }
\${ MonthNames[2] } \${ MonthNames[3] }
\${ Friends["Deshpande"] } \${ Friends["Mishra"] }
\${ Friends["Dawane"] } \${ Friends["Govindraj"] }

The EL Operators

- **Arithmetic Operators**

- +
- −
- *
- / or div
- % or mod

- **Relational Operators**

- == or eq
- != or ne
- < or lt
- < or gt
- <= or le
- >= or ge

- **Logical Operators**

- &&
- and
- ||
- or
- !
- not

- **Empty Operator**

- empty

Using EL Operators

```
<html>
<body>
${num>3}<br><br>
${integer le 12 }<br><br>
${list[0] || list["1"] and true}<br><br>
${num > integer }<br><br>
${num == integer-1 }<br><br>
${42 div 0}
${list[0]} ${list["1"]}
${requestScope[integer] ne 4 and 6 le
    num || false }<br><br>
</body>
</html>
```

```
protected void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    String num="2";
    request.setAttribute("num",num);
    Integer i = new Integer("3");
    request.setAttribute("integer",i);

    ArrayList list = new ArrayList();

    list.add("true"); list.add("false");
    list.add("2"); list.add("10");

    request.setAttribute("list",list);

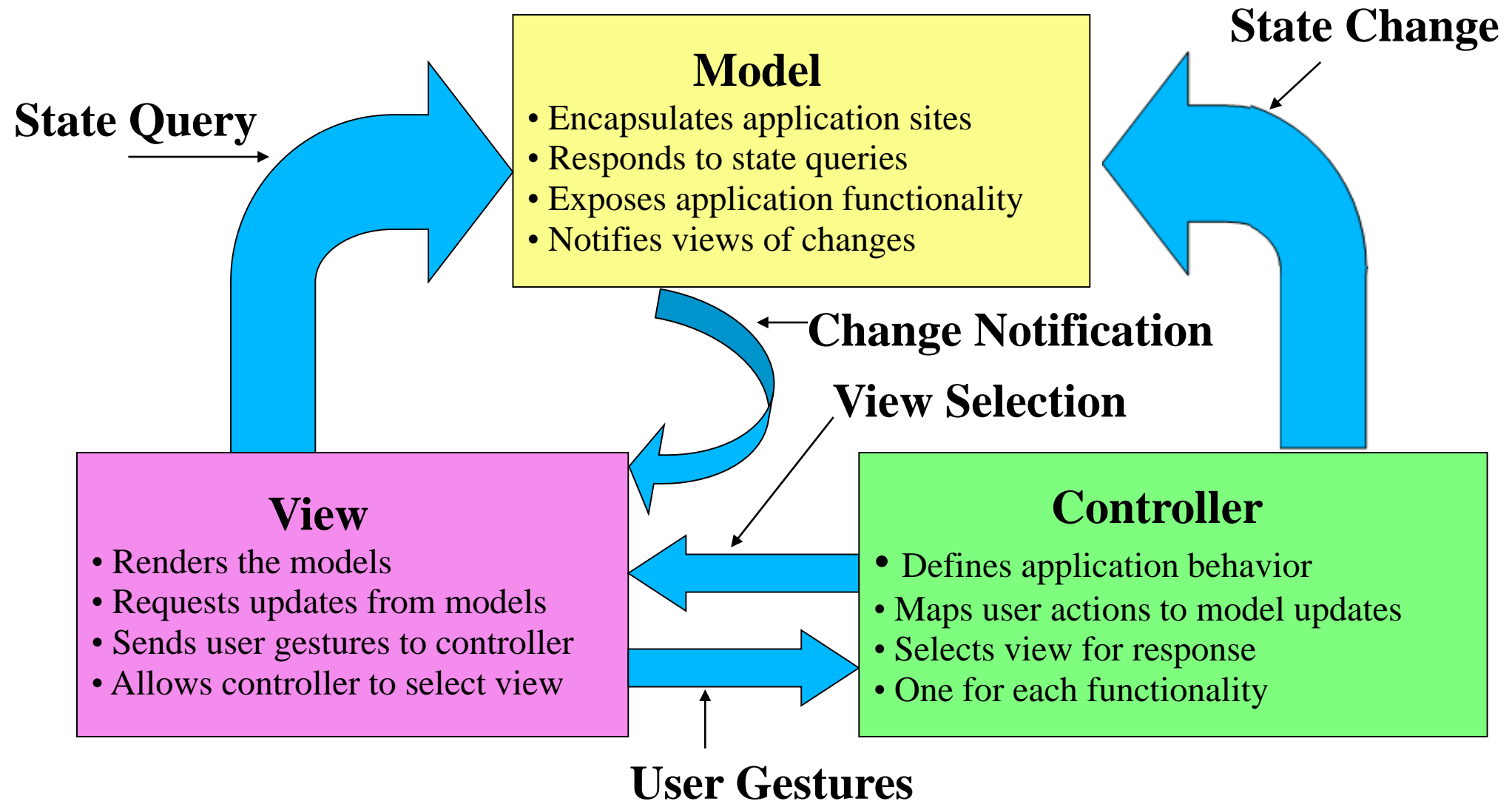
    RequestDispatcher rd =
        request.getRequestDispatcher("/eloperator");
    rd.forward(request,response);
}
```

Module 15: The MVC Architecture

Overview :

- The MVC Architecture
- Dispatching requests between Servlet and JSP
- Packaging elements of Application

Model View Controller (MVC)



Model View Controller (MVC) contd...

Model :

- Encapsulates the application logic.
- Could be more than one for an application.
- Can be a POJO or EJB or DAO or combination of these.
- Is the only part of the application which interacts with the DB.

View :

- Forms the presentation layer of your application.
- Could be GUI or JSP or HTML or any other presentation type.

Controller :

- Processes and responds to events, typically user actions, and may invoke changes on the model.

Module 16: The Custom Tags

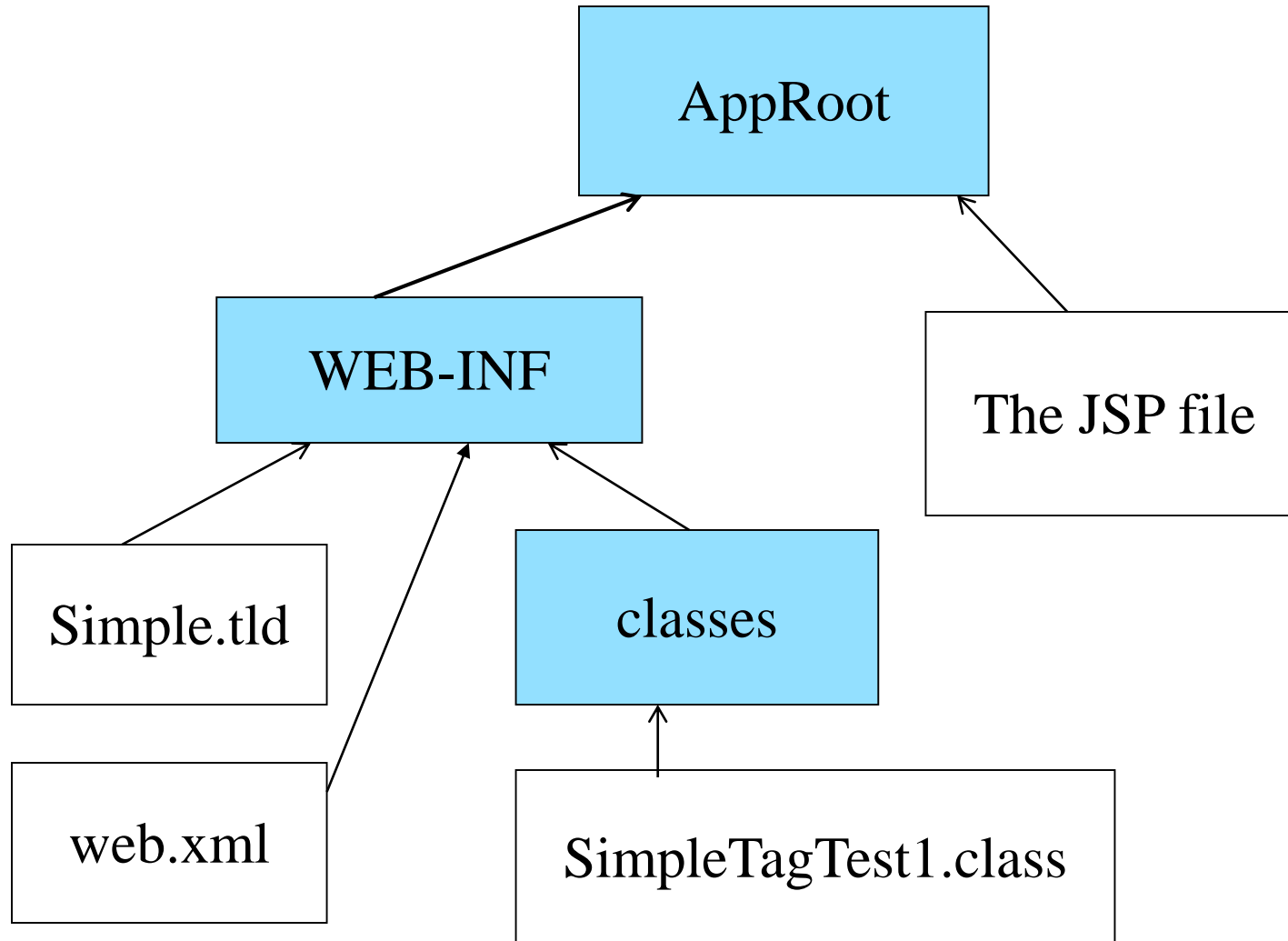
Overview :

- Simplifying JSP using Custom Tags.
- The TagSupport and BodyTagsupport classes.
- Designing Custom Tags and using them

Simplifying JSP using Custom Tags.

- These are easy to use elements for JSP, but represents complex server side behavior.
- You can define your own tag, group them into collection called tag library that can be used in any number of JSP files.
- Steps necessary to build your Custom tag :
 - Create a “ TAG HANDLER CLASS ”.
 - Add a tag description into “TAG LIBRARY DESCRIPTOR FILE ” (.tld file).
 - Add <taglib> tag in web.xml
 - Use the tag in the JSP File

Making a Simple Tag Handler



The Tag Handler class

```
import javax.servlet.jsp.tagext.TagSupport;

public class TagHandlerClassTest1 extends TagSupport {
    public int doStartTag() throws JspException {
        try {
            JspWriter out = pageContext.getOut();
            out.println("Pragati Software Pvt. Ltd.");
        }
        catch (IOException ioe) {
            throw new JspException("Exception in CompanyName10", ioe);
        }
        return (SKIP_BODY);
    }
}
```

The Tag Library Descriptor File

```
<taglib>
  <tlib-version>1.2</tlib-version>
  <uri>simpleTags</uri>
  <tag>
    <description>Demo of Custom Tags</description>
    <name>TagName</name>
    <tag-class>Mapping with .class</tag-class>
    <body-content>empty</body-content>
  </tag>
  <tag>
    ....
  </tag>
</taglib>
```

The JSP File using Tag

```
<%@ taglib uri="/tags" prefix="tagjsp" %>
<HTML>
<HEAD>
<TITLE><tagjsp:COMPANYNAME/></TITLE>
</HEAD>
<BODY>
<CENTER>
<H1><tagjsp:COMPANYNAME/></H1>
<H3><tagjsp:COMPANYADDRESS/></H3>
</CENTER>
</BODY>
</HTML>
```

Custom Tag with Body

```
public class Tag30DateFormat extends TagSupport {
    int format = 1;
    public int doStartTag() throws JspException{
        JspWriter out = pageContext.getOut();
        Calendar cal = Calendar.getInstance();
        cal.setTime(new Date());
        String [] monNm = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep",
            "Oct", "Nov", "Dec"};
        String [] weekNm = {"Sunday", "Monday", "Tuesday", "Wednesday",
            "Thursday", "Friday", "Saturday"};
        String formatString="";
        switch(format) {
            case 2 :    formatString = cal.get(Calendar.DAY_OF_MONTH)+"th of
            "+monNm[cal.get(Calendar.MONTH)]+ " " +
            cal.get(Calendar.YEAR) ;
            break;
            case 3 :    formatString = cal.get(Calendar.DAY_OF_MONTH)+" th " +
            weekNm[cal.get(Calendar.DAY_OF_WEEK)]+" "+
            (cal.get(Calendar.MONTH)+1) + " "+cal.get(Calendar.YEAR);
            break;
            default: formatString = cal.get(Calendar.DAY_OF_MONTH) + "/" +
            (cal.get(Calendar.MONTH)+1)+"/"+cal.get(Calendar.YEAR);
        }
    }
}
```

Custom Tag with Body

```
try {
    out.println(formatString);
}
catch(IOException ie){
    throw new JspException("Exception in todayformat30", ie);
}
return (SKIP_BODY);
}

public void setFormat(Integer i){
    format = i.intValue();
    System.out.println("In setFormat:"+format);
}
}
```


Custom Tag with Body

```
<%@ taglib uri="/tags" prefix="tagjsp" %>
<HTML>
  <HEAD>
    <TITLE><tagjsp:COMPANYNAME/></TITLE>
  </HEAD>
  <BODY>
    <H1><tagjsp:COMPANYNAME/></H1>
    Today(D) is:<tagjsp:DATEFORMAT/><BR>
    Today(1) is:<tagjsp:DATEFORMAT format='1'/><BR>
    Today(2) is:<tagjsp:DATEFORMAT format='2'/><BR>
    Today(3) is:<tagjsp:DATEFORMAT format='3'/><BR>
  </BODY>
</HTML>
```

Custom Tag with Body

```
<tag>  
  <name>DATEFORMAT</name>  
  <tag-class>packtags.Tag30DateFormat</tag-class>  
  <attribute>  
    <name>format</name>  
    <required>false</required>  
    <rtextprvalue>true</rtextprvalue>  
    <type>java.lang.Integer</type>  
  </attribute>  
</tag>
```