Introduction to Web Engineering SENG2050/6050

Lecture 2 – Servlets

Announcement

- Tutorial links for CSE students
- SENG1050 Labs for CSE students

Server-side Web Technologies

- Serving **static** Web pages is simple using a Web Server
  1. Server receives request
  2. Maps URI to a local document
  3. Transmits local document
- The document is usually the same
  - Only changes when manually updated on the server
  - It can be “cached” by intermediate devices

Server-side Web Technologies

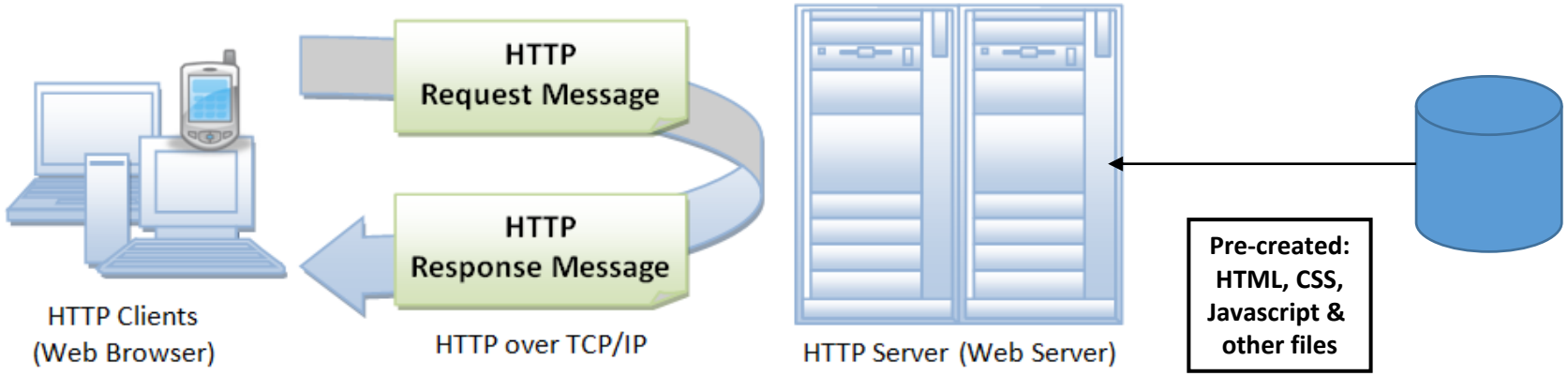


Image Source: https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html

Server-side Web Technologies

- It’s difficult to get dynamic behavior using static content
  - Can embed scripting languages in the document – JavaScript, VBScript, etc
  - Can embed Java Applets in the document
- Disadvantage
  - Browser is required to do the processing – what about low-power devices like mobile phones?
  - Browser must handle all the information – even if not needed or sensitive!

Server-side Web Technologies

- Serving **dynamic** pages is more involved
- Might include
  - Parsing the document to identify “special instructions” for the server
  - Running a local executable to generate the document in real-time
  - Contacting other servers to provide information used to generate the document
- The document can “change” each time
  - Intermediate “caches” must be careful

Server-side Web Technologies

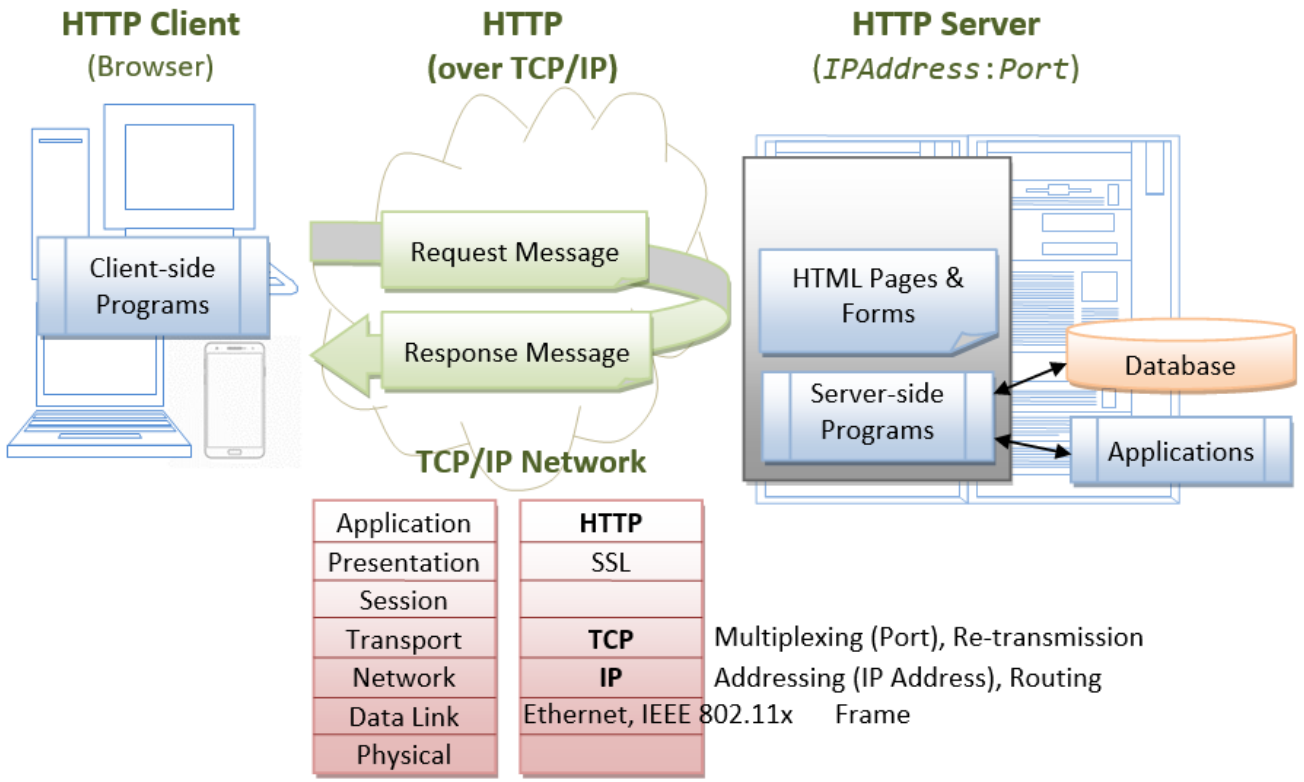


Image Source: https://www.ntu.edu.sg/home/ehchua/programming/java/JavaServlets.html

Server-side Web Technologies

Dynamic Web pages are:

- Based on data sent by the client
- Derived from data that changes frequently
- Generated with information from database and other sources
- Additional complexities: managing multiple clients and their sessions

Server Side Includes

- They provide a handler which will process files before they are sent to the client.
- They allow conditional text, the inclusion of other files or programs, and the setting and printing of environment variables.
- Points for
  - Relatively simple to use
  - Only moderate performance impact
  - Run in a standard web server
- Points against
  - Severely restricted capabilities, or
  - Suffer from major security issues

Example:

1. <!--#echo var="DATE_LOCAL"-->
2. <!--#include virtual="myfile.txt" -->
3. <!--#exec cgi="/cgi-bin/clicktrade.cgi" -->

Server Side Includes

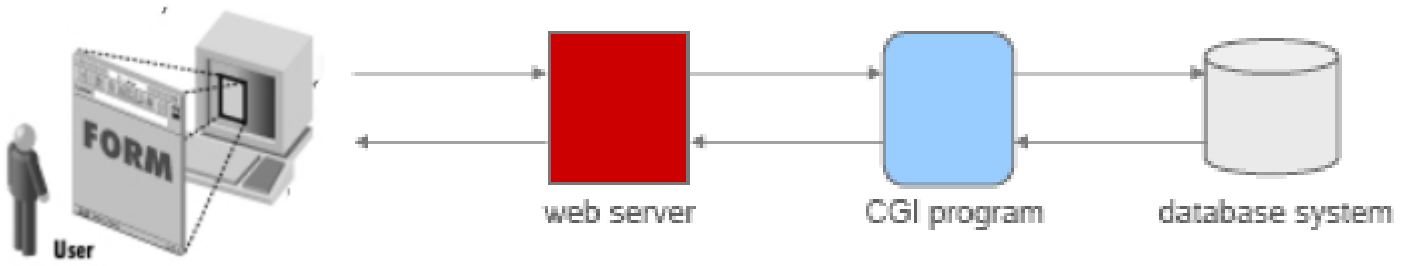
“Special tags” embedded in a HTML document to

- Conditionally use fragments of the document
  - If agent = = Firefox then … else …
- Include an external HTML fragment in the document
  - Use a common header and footer across a entire Web site
- Insert the value of certain system variables
  - Insert the current date and time into a page
- Run a local executable and paste its output into the document

- http://httpd.apache.org/docs/2.2/howto/ssi.html

Common Gateway Interface – CGI

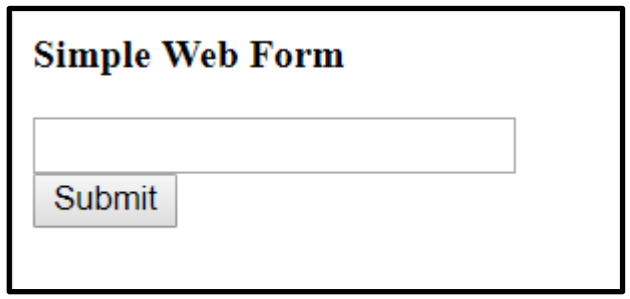
- A standard for passing information between a Web server and a local executable
- Often acts as gateways to applications or database systems

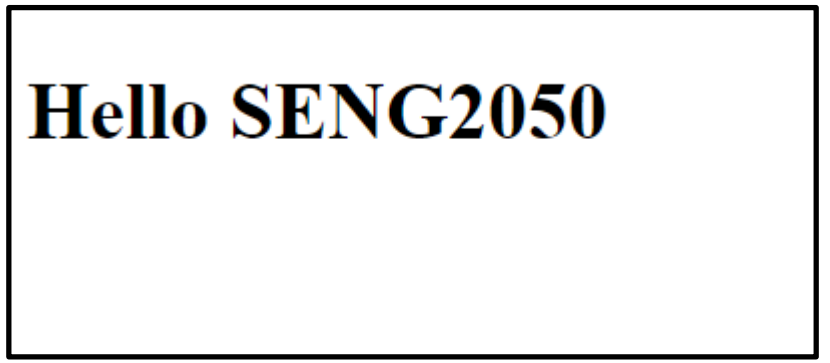


Common Gateway Interface – CGI

```html
<!DOCTYPE html>
<html>
<head> <title>Simple Web Form</title> </head>
<body>
<b>Simple Web Form</b><p>
<form action="http://nowhere.com/cgi-bin/form.pl" method="get">
<input type="test" name="field" size="25"><BR>
<input type="submit" name="Submit" value="Submit">
</form>
</body>
</html>
```

```perl
#!/usr/bin/perl -Tw

use strict;
use CGI ':standard';

my $name = param('name');

my $list;

print header,
start_html(-title=>$name),
h1("Hello $name"),
end_html;

```



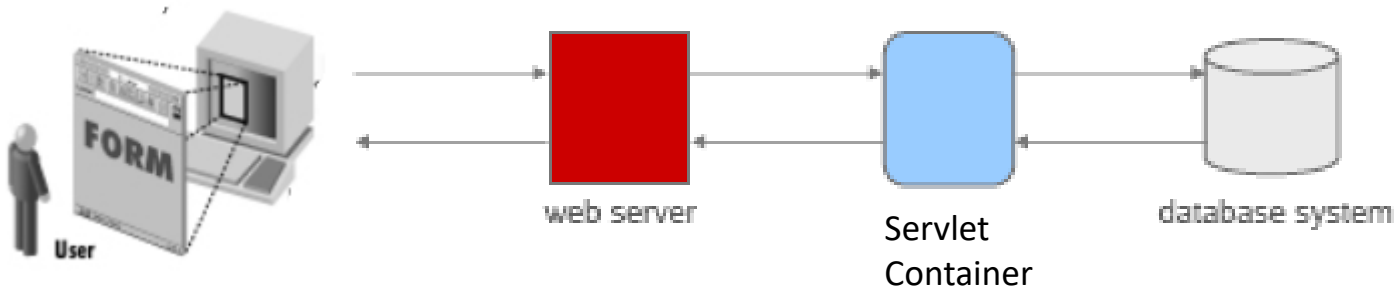


Common Gateway Interface – CGI

- Poor performance
  - process started for each CGI request
  - if CGI program acts as a gateway to a DBMS, then a connection to the DBMS has to be opened and closed for each request
  - Platform dependence
- For this reason various improvements were proposed
  - Dedicated, persistent processes for handling requests to specific applications

Servlet Containers

- Runtime environment for Java servlets
- Passes request data to a mapped servlet
- Sends response data generated from the mapped servlet
- Can also serve static web content
  - html
  - JavaScript files
  - CSS files
  - text files
  - etc.
- Apache Tomcat
- Eclipse's Jetty



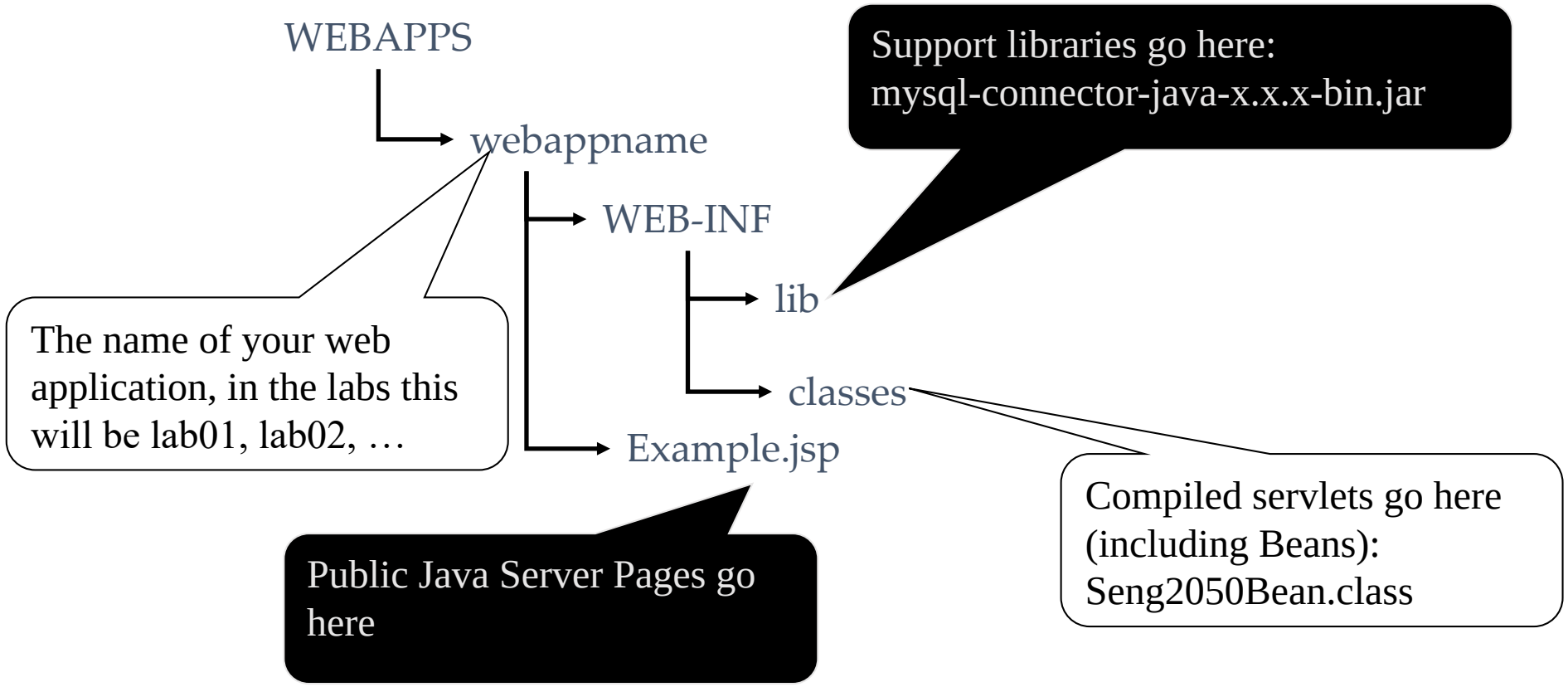
Web Application Server

- Component based product that resides in the middle-tier of a server centric architecture
- Contains a servlet container
- Adds other functionality and support
- Oracle Glassfish

Tomcat

- We will be running Java servlets under the Tomcat servlet container
- Current version **9**
- Implements version
  - 4.0 of the servlet spec
  - 2.3 of the JSP spec
- Handles HTTP requests and responses for you
  - Providing them as objects for your servlets to work with

Java Servlets and Tomcat



Java Servlet

- Java class that runs within a compatible:
  - Servlet container
  - Web application Server
- Reads the HTTP request data sent by a browser
- Generates a response
  - Generally a web page to display
  - Can be other documents (in the context of an AJAX request maybe)
  - Can forward or redirect requests
- Sends the response back over HTTP

Java Servlet

- Precompiled code “inside” the servlet container
  - Shared Library on disc, or actually in memory
  - Some servers keep common CGI scripts in memory
    - http://perl.apache.org/start/index.html
  - Java Servlets are servlets written in Java, stored in the memory of the web server
- Points for
  - Much better performance
  - Better security
- Points against
  - Harder to configure (servlet container vs. web server)

Java Servlet At Work

Java Servlet

Java Servlets provide

- Dynamic generation of documents
- Processing of other documents on the server
- Processing of other documents on other servers (through another servlet!)
- Handling insecure requests from client-side scripts
- Services independent of the Web server!

Java Servlets Tasks

- Regardless of the application, servlets usually carry out the following routine:
  1. Read any data sent by the user
     - Capture data submitted by an HTML form.
  2. Look up any HTTP information (implicit HTTP request)
     - Determine the browser version, host name of client, cookies, etc.
  3. Generate the Results
     - Connect to databases, connect to legacy applications, etc.
  4. Format the Results
     - Generate HTML on-the-fly
  5. Set the Appropriate HTTP headers (implicit HTTP response data)
     - Tell the browser the type of document being returned or set any cookies.
  6. Send the document back to the client

Basic Servlet Structure

```java
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet("/Hello")
public class Hello extends HttpServlet {

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

        // Your code to deal with a GET request

    }

    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        // Your code to deal with POST request
        doGet(request, response);
    }

}
```

GET vs POST

GET	POST
Only limited amount of data can be sent because data is sent in header.	Large amount of data can be sent because data is sent in body.
Not secured because data is exposed in URL bar.	Secured because data is not exposed in URL bar.
Can be bookmarked.	Cannot be bookmarked.
Get request is idempotent . It means second request will be ignored until response of first request is delivered	Post request is non-idempotent.
More efficient and used more than Post.	Is less efficient and used less than get.

Java Servlet

- Import the Servlet API:
  - **import javax.servlet.*;**
  - **import javax.servlet.http.*;**
- All your servlets must extend HTTPServlet.
- HTTPServlet represents the base class for creating Servlets within the Servlet API.
- The Full Servlet API is available at:
  - http://java.sun.com/products/servlet/2.3/javadoc/index.html
- Once you have extended HTTPServlet, you must override one or both:
  - doGet(): to capture HTTP Get Requests
  - doPost(): to capture HTTP Post Requests

Java Servlet

- The doGet() and doPost() methods each take two parameters:
  - HTTPServletRequest: encapsulates all information regarding the browser request.
    - Form data, client host name, HTTP request headers.
  - HTTPServletResponse: encapsulates all information regarding the servlet response.
    - HTTP Return status, outgoing cookies, HTML response.
- If you want the same servlet to handle both GET and POST, you can have doGet call doPost or vice versa.

Java Servlet

- The HTTPServletResponse object has a getWriter() method.
- This method returns a java.io.PrintWriter object for writing data out to a web page for the Web Browser.

PrintWriter out = response.getWriter();

Java Servlet

- Once you have an OutputStream object, you just call the println() method to output to the browser.
- To generate HTML, you need to add two steps:
  - Tell the browser that you are sending back HTML.
  - Modify the println() statements to return valid HTML.

Example

```
5  import javax.servlet.http.HttpServletResponse;
6  import java.io.IOException;
7  import java.io.PrintWriter;
8
9  @WebServlet("/Hello")
10 public class Hello extends HttpServlet {
11
12     @Override
13     protected void doGet(HttpServletRequest request, HttpServletResponse response)
14             throws ServletException, IOException {
15
16         response.setContentType("text/html");
17
18         PrintWriter out = response.getWriter();
19
20         try {
21             out.println("<!DOCTYPE html>");
22             out.println("<html><head>");
23             out.println("<title>Hello </title></head>");
24             out.println("<body>");
25             out.println("<h1>Hello SENG2050/6050!</h1>");  // says Hello
26             out.println("</body>");
27             out.println("</html>");
28         } finally {
29             out.close();  // Always close the output writer
30         }
31     }
32 }
33
```

Servlet’s Life Cycle

1) Servlet class is loaded and Servlet instance is created
2) **init** method is invoked
3) **service** method is invoked (iteratively)
4) **destroy** method is invoked

Java Servlet – Architecture

A Java Servlet is a class which implements the `Servlet` interface

- `init(ServletConfig config)` – called only when the servlet is first loaded
- `destroy()` – called when the servlet is unloaded
- `service(ServletRequest req, ServletResponse res)` – called every time the servlet is accessed

Java Servlet – Architecture

`ServletConfig`

- Passes configuration from the server to the servlet
- Allows the servlet to access helper methods in the server – `config.getServletContext.`…
  - `getServerInfo()` – name and version info of the servlet container
  - `log(`*`message`*`)` – write a message to the server's log file

Java Servlet – Architecture

`ServletRequest` - Represents the current request

- **`getRemoteAddr()`**, **`getRemoteHost()`** – the "client" making the request
- **`getServerName()`**, **`getServerPort()`** – the "server" processing the request
- **`getParameter(`*`name`*`)`** – the (single) value of the named parameter (e.g., a <form> input)
- **`getParameterNames()`** – list all parameters
- **`getParameterValues(`*`name`*`)`** – list all values of the named parameter (an array of **`String`**)
- **`getReader()`** – for reading uploaded files

Java Servlet – Architecture

`SerlvetResponse` - Represents the current response

- **`setContentType(`*`type`*`)`** – the MIME type of the response
- **`setContentLength(`*`length`*`)`** – the length of the response
- **`getWriter()`** – a **`PrintWriter`** for generating the response document

Java Servlet – Architecture

`HttpServletRequest`

- `getQueryString()` – query part of URI
- `getRemoteUser()` – *who* made the request
- `getCookies()` – get the cookies associated with this request
- `getSession()` – get the session associated with this request
- `getHeaderNames()`, `getHeader(`***`name`***`)` – access all HTTP headers

Java Servlet – Architecture

`HttpServletResponse`

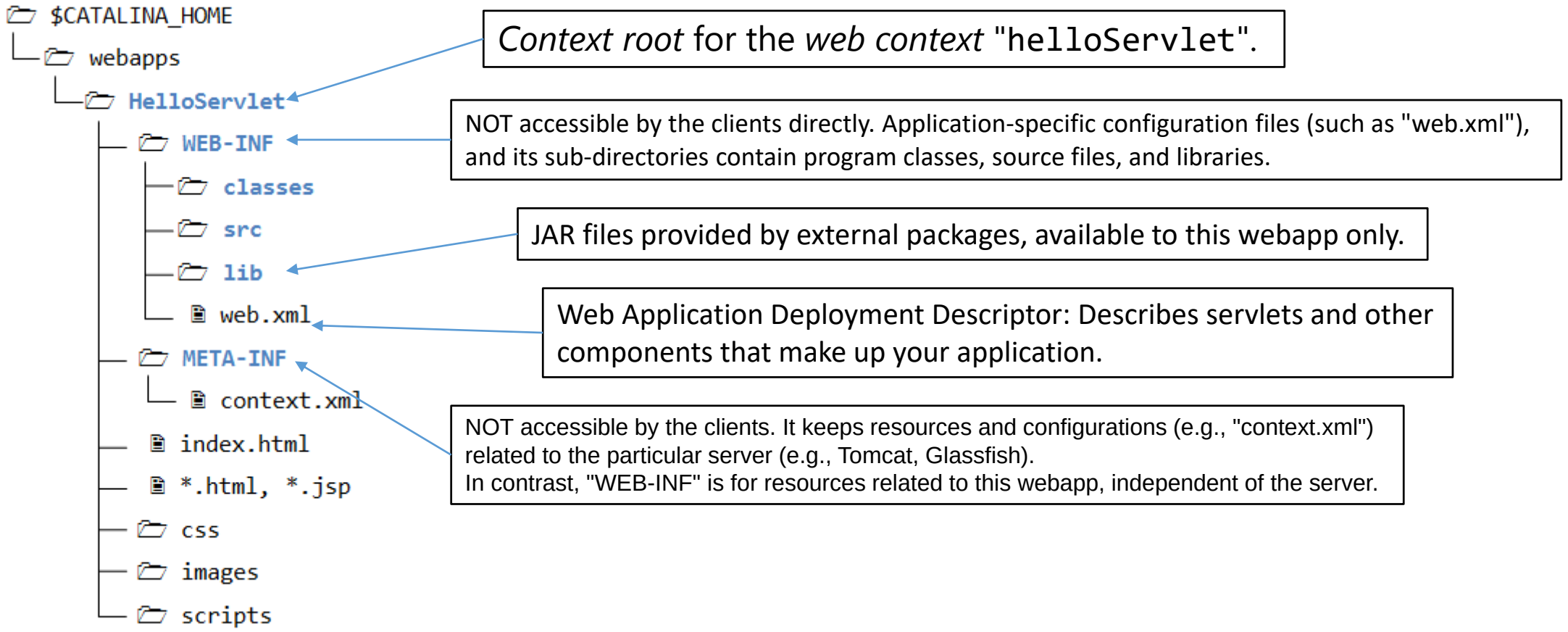
- **`setStatus()`** – the HTTP response code
- **`sendError(`*`code`*`)`** – send an error status to client using status code
- **`sendRedirect(`*`uri`*`)`** – redirect the client, sends redirect response to client
- **`addCookie(Cookie ck)`** – add a cookie associated with this response
- **`setHeader(`*`name`*`, `*`value`*`)`** – set any HTTP header with name and value

Java Packages

- Package: A group of related classes.
  - For example:
    - package coreservlets;
- In real web sites, multiple programmers may be creating multiple servlets.
- By dividing your code base into packages, it helps to modularize the code.
- A very common practice in the real world.

Java webapp directory Structure

- Standardized directory structure for storing various types of resources.



THE END

QUESTIONS??

THANKS!!