JAVA SERVLETS

NAMIT KHANDUJA
ASSTT.PROF.
DEPT. OF CS&E,FET,GKV.
namitkhanduja@gkv.ac.in

BCE-C602 (ENTERPRISE COMPUTING WITH JAVA)

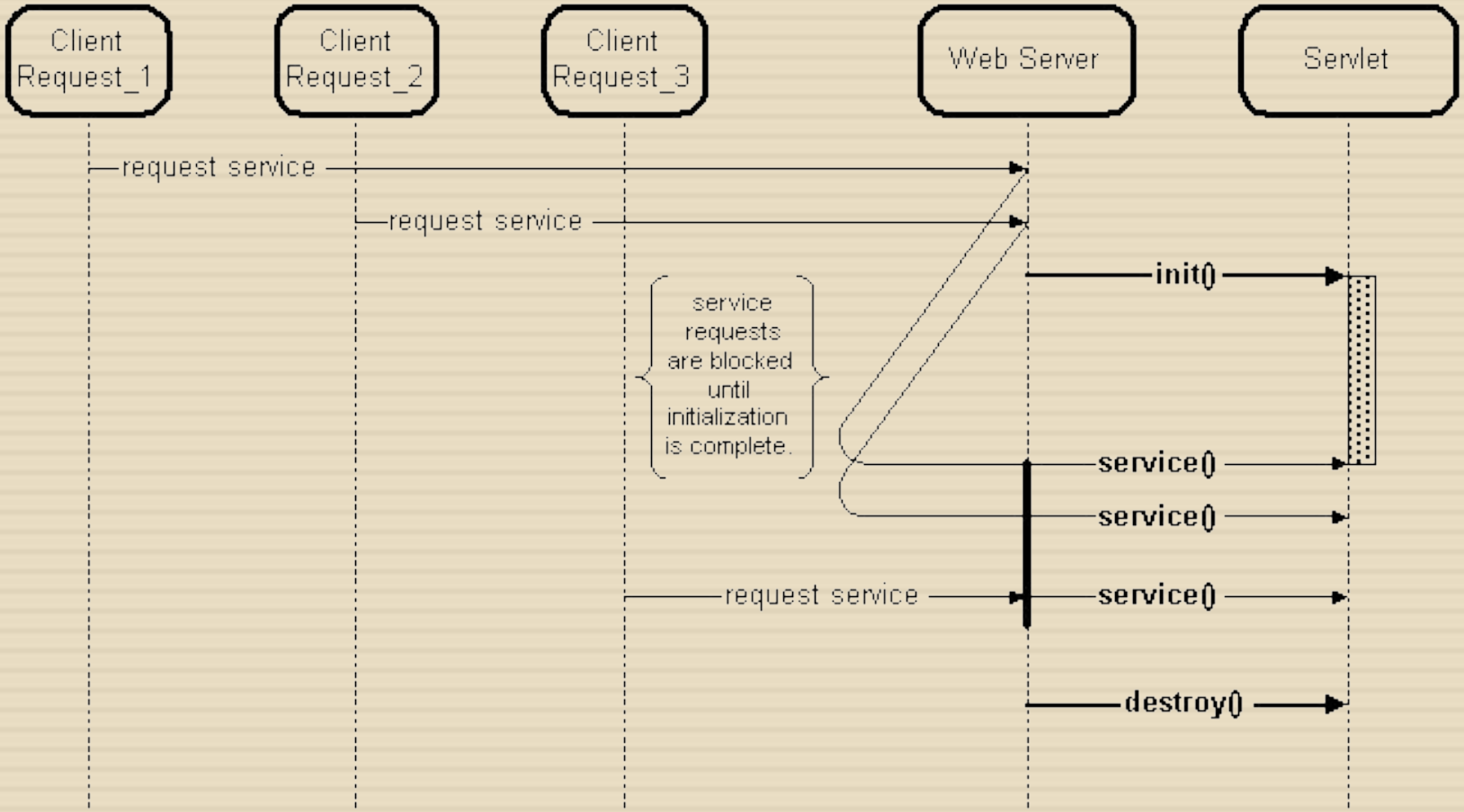
Outline

- Introduction
- Lifecycle
- Servlet api
- Steps to create servlet
- Hello World! – Example
- …
- …
- …

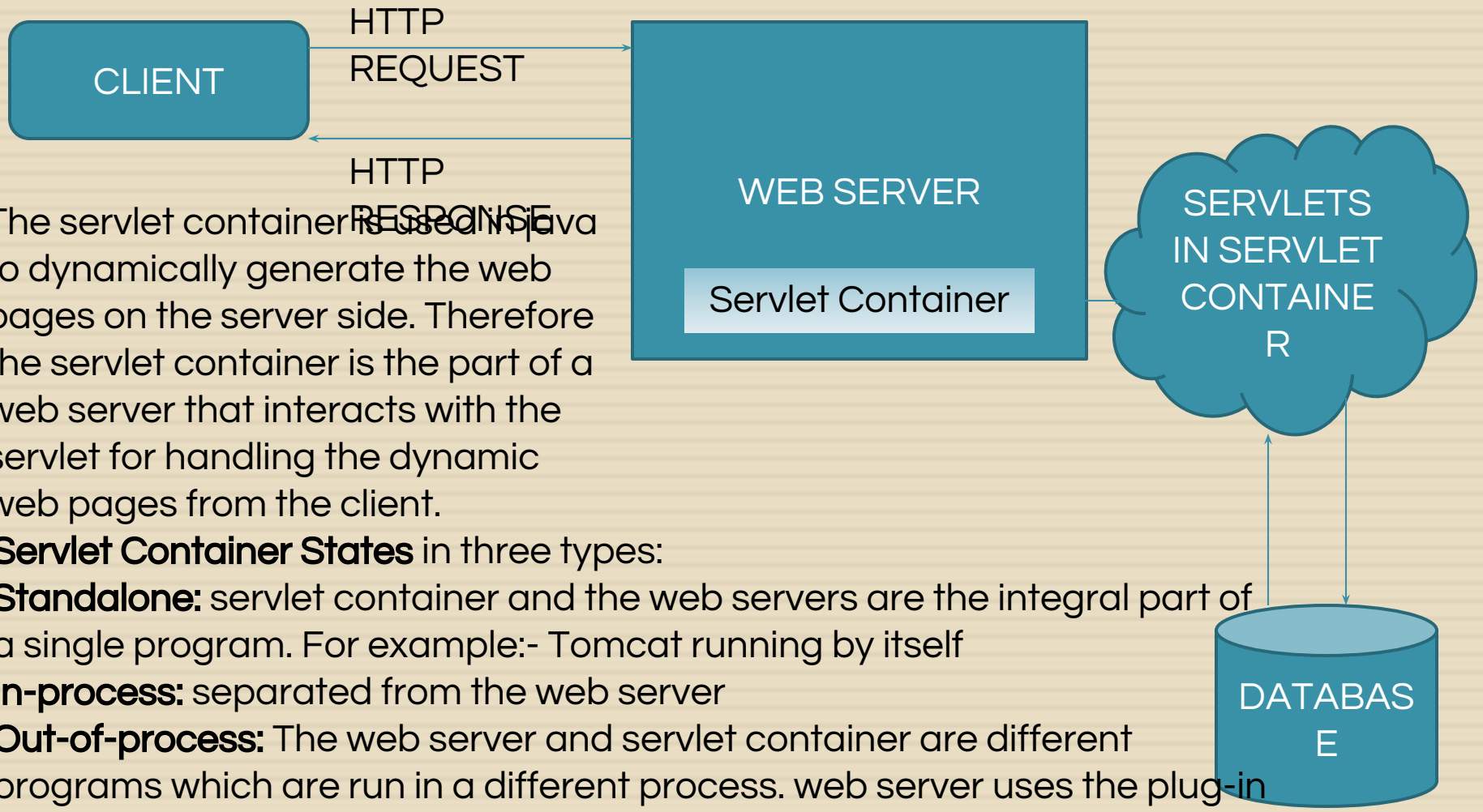
Introduction

- A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed by means of a request-response programming model.
- Servlet is a web component that is deployed on the server to create dynamic web page.
- Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes.
- servlets typically run on the HTTP protocol. HTTP is an *asymmetrical request-response protocol*.

Servlet Life Cycle



Macroscopic View of Life Cycle



The servlet container is used in Java to dynamically generate the web pages on the server side. Therefore the servlet container is the part of a web server that interacts with the servlet for handling the dynamic web pages from the client.

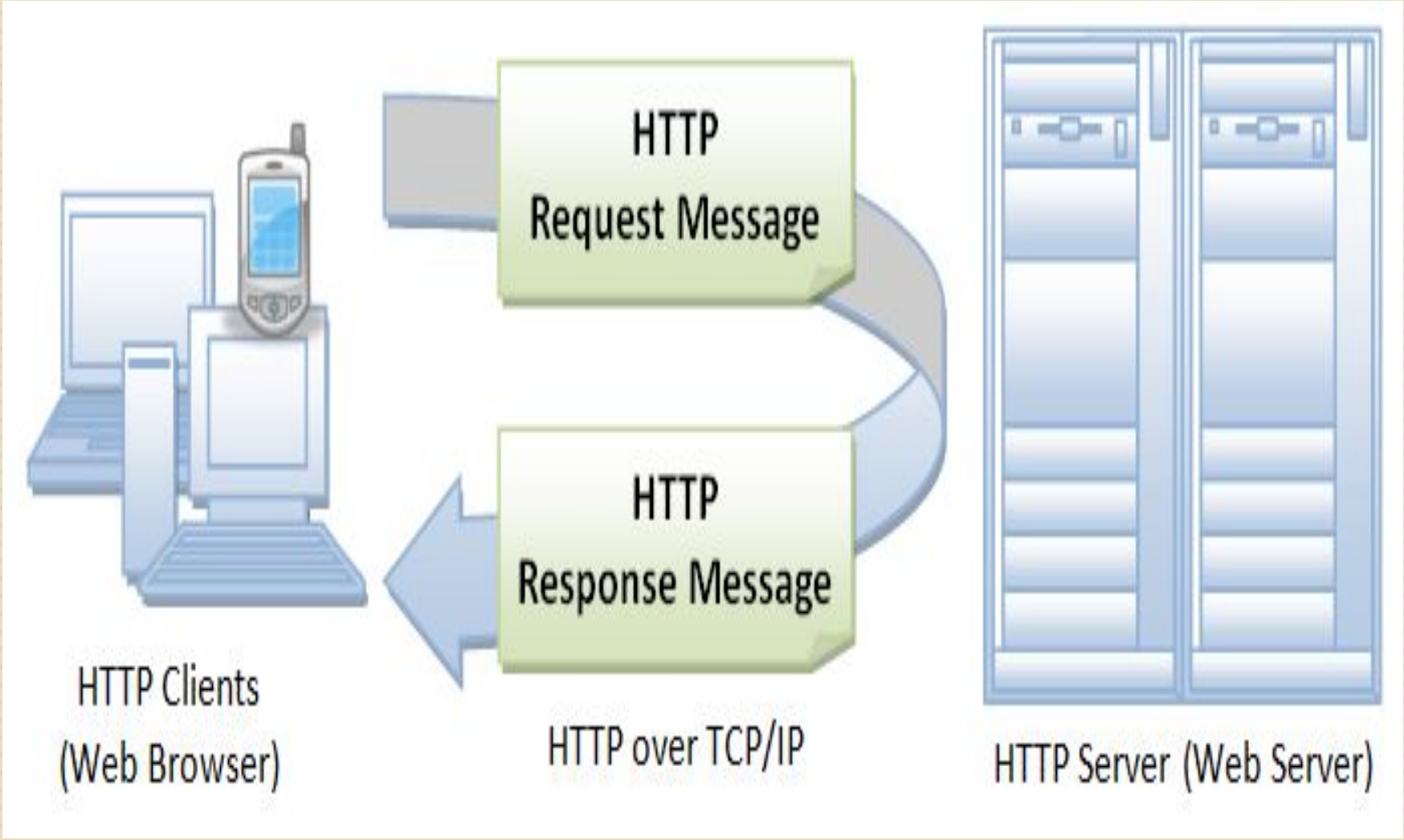
Servlet Container States in three types:

Standalone: servlet container and the web servers are the integral part of a single program. For example:- Tomcat running by itself

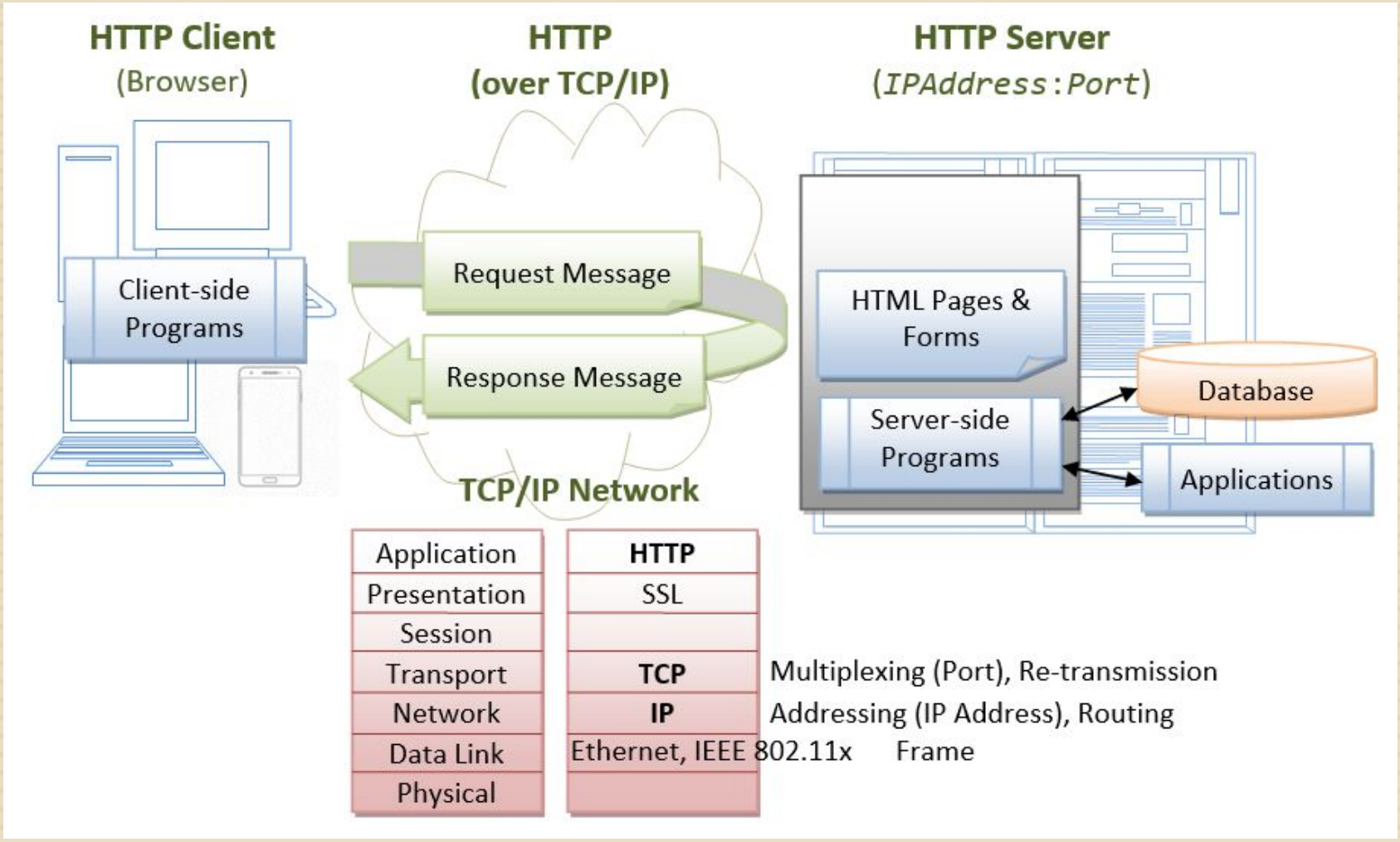
In-process: separated from the web server

Out-of-process: The web server and servlet container are different programs which are run in a different process. web server uses the plug-in provided by the servlet container.

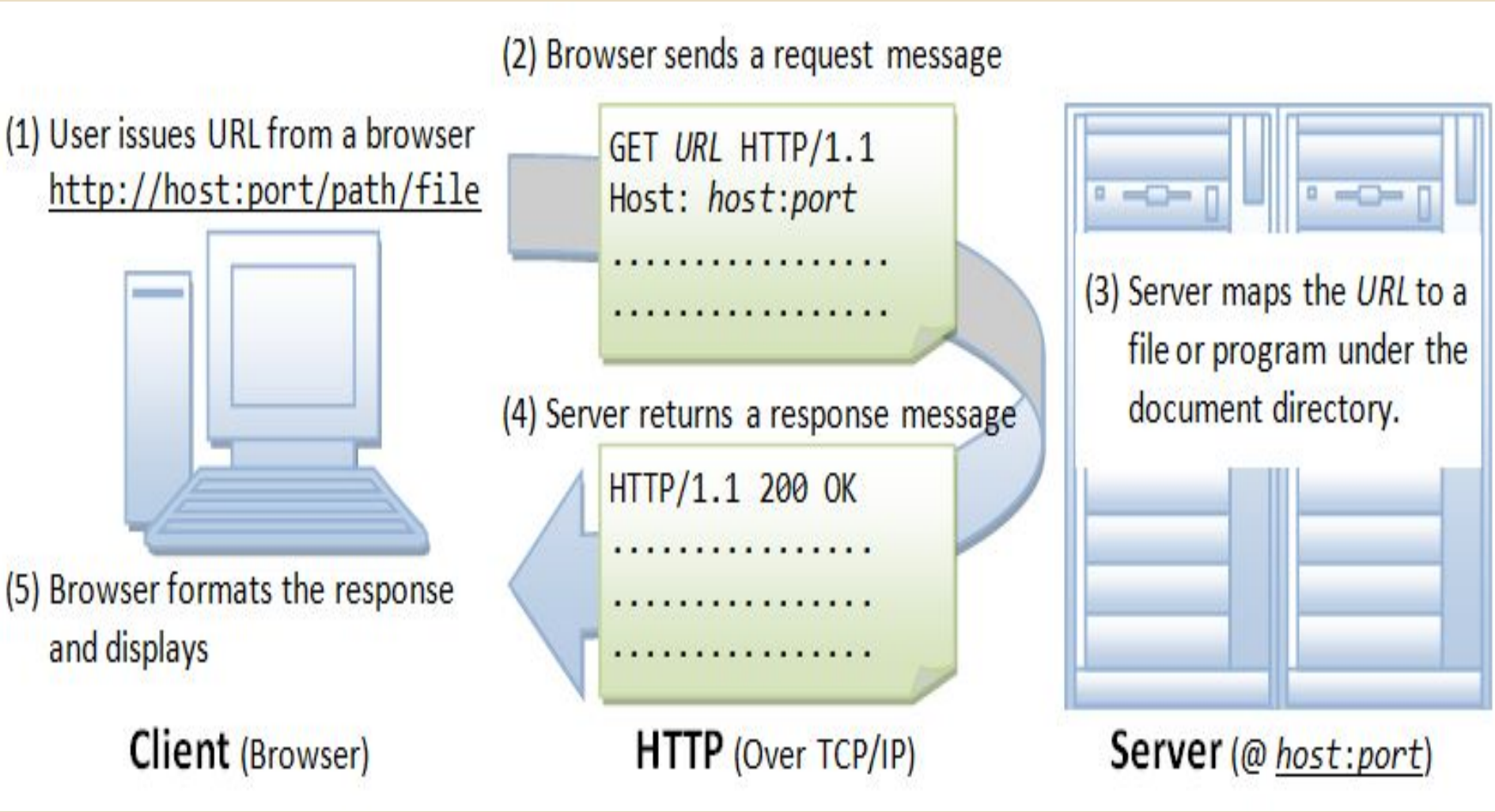
Http Request – Response Model



HTTP-Client Server Architecture



HTTP-Steps


(2) Browser sends a request message
(1) User issues URL from a browser
http://host:port/path/file
GET URL HTTP/1.1
Host: host:port
.....
.....
(3) Server maps the URL to a file or program under the document directory.
(4) Server returns a response message
HTTP/1.1 200 OK
.....
.....
.....
(5) Browser formats the response and displays
Client (Browser)
HTTP (Over TCP/IP)
Server (@ host:port)

Dissecting the HTTP steps

- Uniform Resource Locator (URL)
- HTTP Protocol

Uniform Resource Locator (URL)

- *protocol*://*hostname*:*port*/*path-and-file-name*There are 4 parts in a URL:
- *Protocol*: The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
- *Hostname*: The DNS domain name (e.g., www.nowhere123.com) or IP address (e.g., 192.128.1.2) of the server.
- *Port*: The TCP port number that the server is listening for incoming requests from the clients.
- *Path-and-file-name*: The name and location of the requested resource, under the server document base directory.
- For example, in the URL http://www.nowhere123.com/docs/index.html, the communication protocol is HTTP; the hostname is www.nowhere123.com. The port number was not specified in the URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is "/docs/index.html".

HTTP Protocol

- EG:- URL http://www.nowhere123.com/doc/index.html into the following request message:
- **GET /docs/index.html HTTP/1.1**
- **Host: www.nowhere123.com**
- **Accept: image/gif, image/jpeg, */***
- **Accept-Language: en-us**
- **Accept-Encoding: gzip, deflate**
- **User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)**
- **(blank line)**
- When this request message reaches the server, the server can take either one of these actions:
  - The server interprets the request received, maps the request into a *file* under the server's document directory, and returns the file requested to the client.
  - The server interprets the request received, maps the request into a *program* kept in the server, executes the program, and returns the output of the program to the client.
  - The request cannot be satisfied, the server returns an error message.

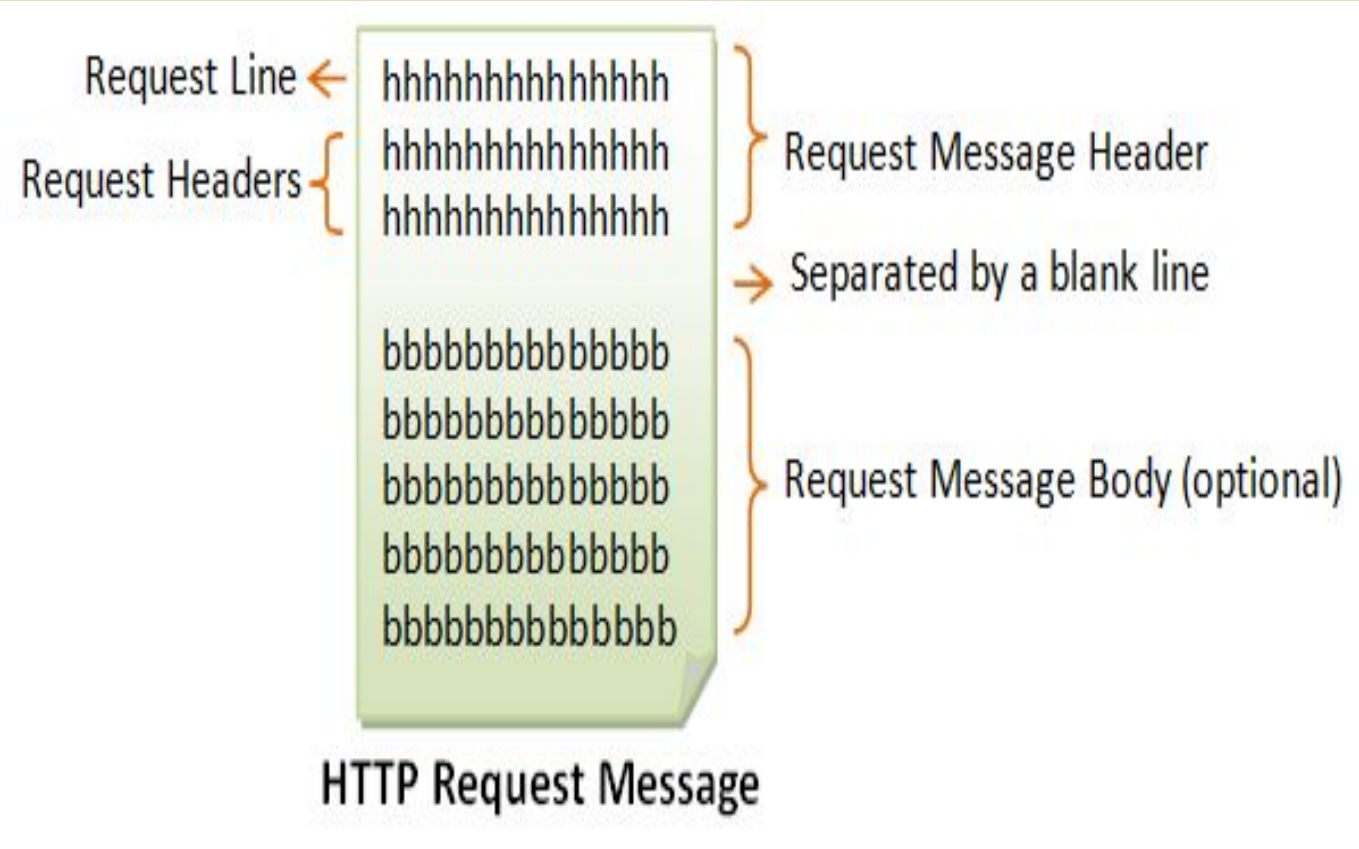
Contd.

- An example of the HTTP response message is as shown:
- **HTTP/1.1 200 OK**
- **Date: Sun, 18 Oct 2009 08:56:53 GMT**
- **Server: Apache/2.2.14 (Win32)**
- **Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT**
- **ETag: "10000000565a5-2c-3e94b66c2e680"**
- **Accept-Ranges: bytes**
- **Content-Length: 44**
- **Connection: close**
- **Content-Type: text/html**
- **X-Pad: avoid browser bug**
- **<html><body><h1>It works!</h1></body></html>**
- The browser receives the response message, interprets the message and displays the contents of the message on the browser's window according to the media type of the response (as in the Content-Type response header). Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf".

Http Request Response Message

- **HTTP Request and Response Messages**
- HTTP client and server communicate by sending text messages. The client sends a *request message* to the server. The server, in turn, returns a *response message*.

http request message format



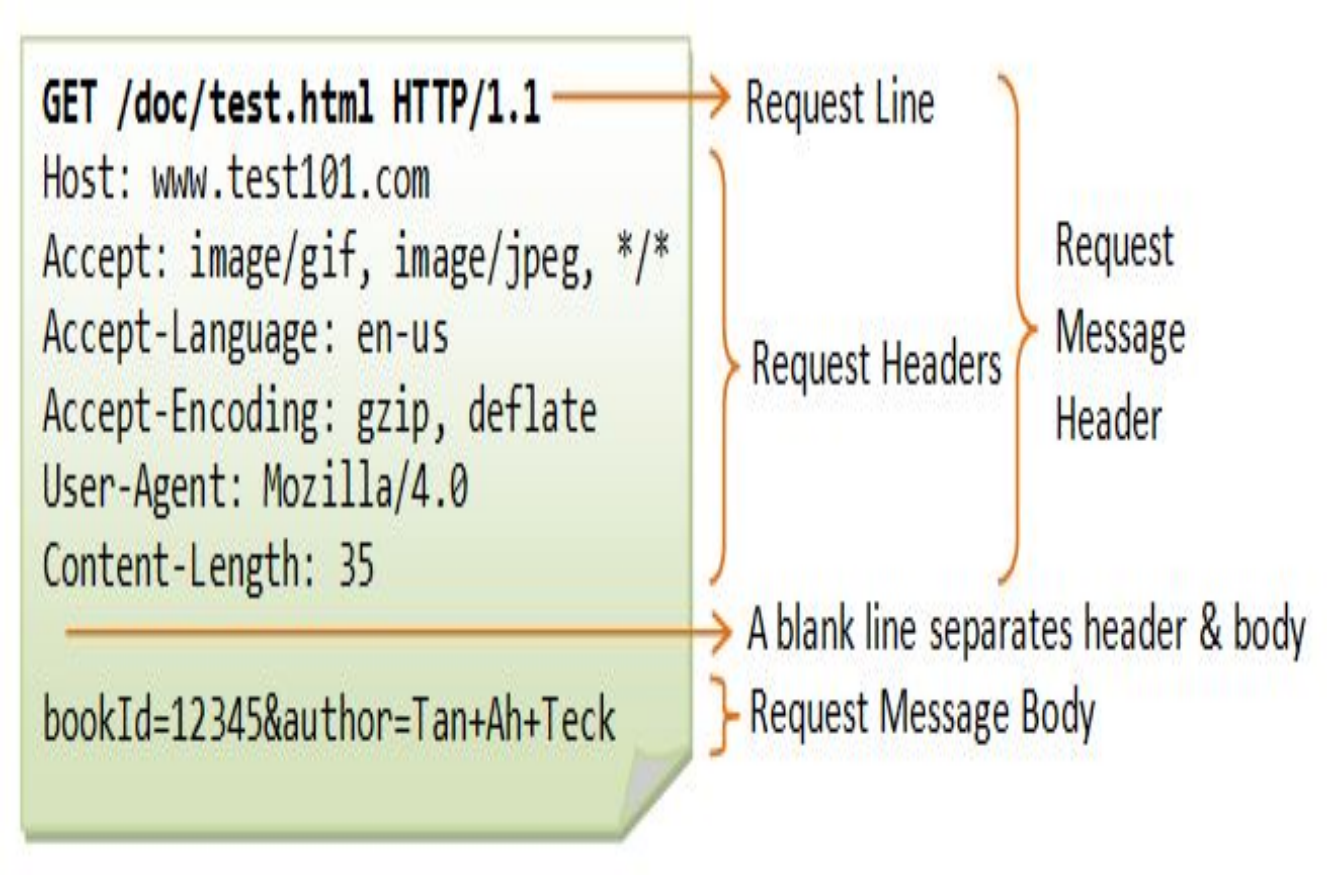
Request Line

- The first line of the header is called the *request line*, followed by optional *request headers*.
- The request line has the following syntax:
- ***request-method-name request-URI HTTP-version***
  - *request-method-name*: HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
  - *request-URI*: specifies the resource requested.
  - *HTTP-version*: Two versions are currently in use: HTTP/1.0 and HTTP/1.1.
- Examples of request line are:
  - GET /test.html HTTP/1.1
  - HEAD /query.html HTTP/1.0
  - POST /index.html HTTP/1.1

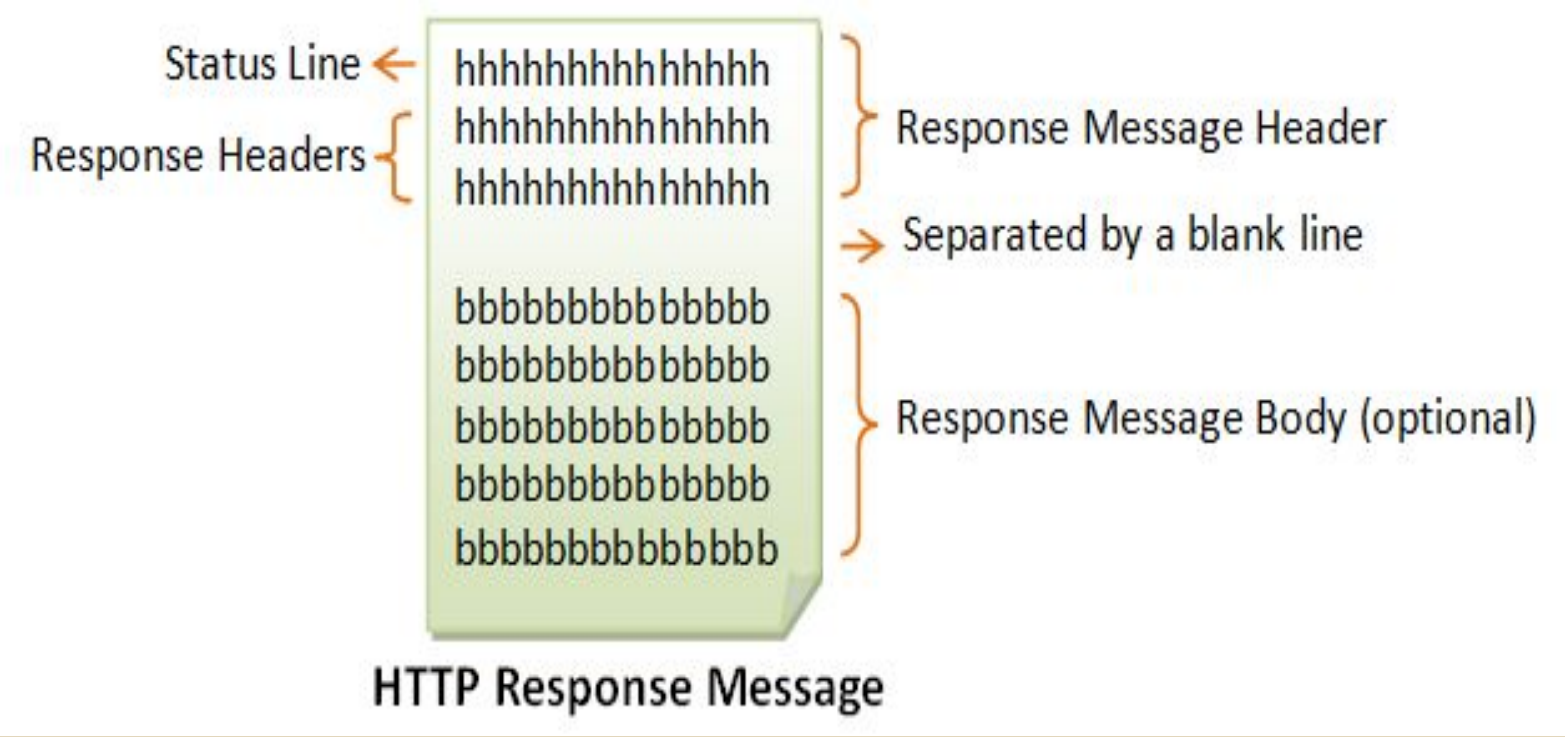
Request Headers

- The request headers are in the form of name:value pairs. Multiple values, separated by commas, can be specified.
  - *request-header-name*: *request-header-value1*, *request-header-value2*, ...
- Examples of request headers are:
  - Host: www.xyz.com
  - Connection: Keep-Alive
  - Accept: image/gif, image/jpeg, */*
  - Accept-Language: us-en, fr, cn

Request Message Example



HTTP Response Message



Status Line

- The first line is called the *status line*, followed by optional response header(s).
- The status line has the following syntax:

  HTTP-version status-code reason-phrase

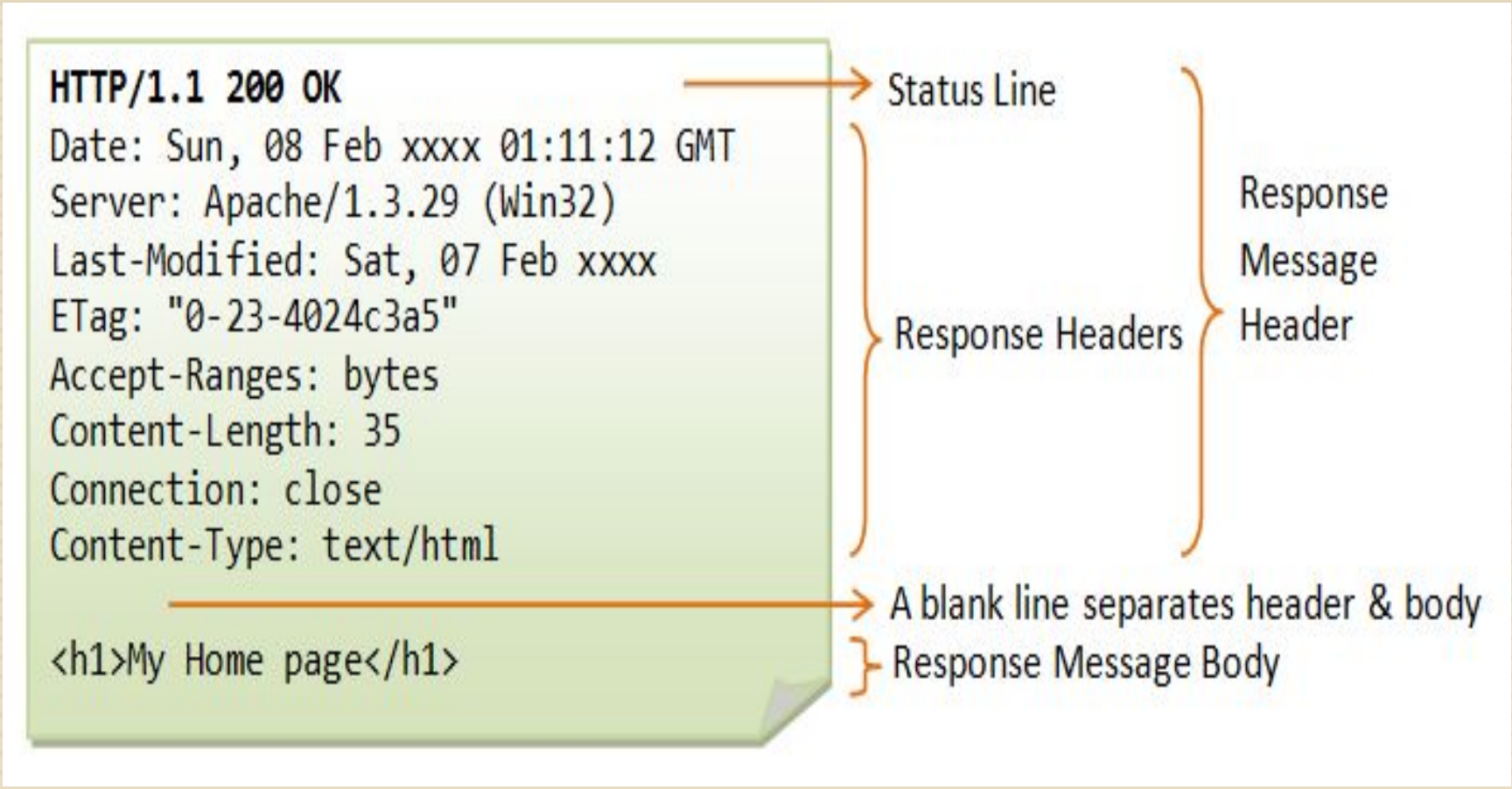
  - *HTTP-version*: The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
  - *status-code*: a 3-digit number generated by the server to reflect the outcome of the request.
  - *reason-phrase*: gives a short explanation to the status code.
- **Common status code and reason phrase** are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".
- Examples of status line are:
  - HTTP/1.1 200 OK
  - HTTP/1.0 404 Not Found
  - HTTP/1.1 403 Forbidden

Response Headers

- The response headers are in the form name:value pairs:
- *response-header-name*: *response-header-value1*, *response-header-value2*, ...

- Examples of response headers are:
  - Content-Type: text/html
  - Content-Length: 35
  - Connection: Keep-Alive
  - Keep-Alive: timeout=15, max=100
- The response message body contains the resource data requested.

Response Message Example



HTTP Request Methods

HTTP protocol defines a set of request methods. A client can use one of these request methods to send a request message to an HTTP server. The methods are:

- GET: A client can use the GET request to get a web resource from the server.
- HEAD: A client can use the HEAD request to get the header that a GET request would have obtained. Since the header contains the last-modified date of the data, this can be used to check against the local cache copy.
- POST: Used to post data up to the web server.
- PUT: Ask the server to store the data.
- DELETE: Ask the server to delete the data.
- TRACE: Ask the server to return a diagnostic trace of the actions it takes.
- OPTIONS: Ask the server to return the list of request methods it supports.
- CONNECT: Used to tell a proxy to make a connection to another host and simply reply the content, without attempting to parse or cache it. This is often used to make SSL connection through the proxy.
- Other extension methods.

Get vs. Post

Two common methods for the request-response between a server and client are:

- **GET**- It requests the data from a specified resource
- **POST**- It submits the processed data to a specified resource

GET	POST
1) In case of Get request, only **limited amount of data** can be sent because data is sent in header.	In case of post request, **large amount of data** can be sent because data is sent in body.
2) Get request is **not secured** because data is exposed in URL bar.	Post request is **secured** because data is not exposed in URL bar.
3) Get request **can be bookmarked.**	Post request **cannot be bookmarked.**
4) Get request is **idempotent** . It means second request will be ignored until response of first request is delivered	Post request is **non-idempotent.**
5) Get request is **more efficient** and used more than Post.	Post request is **less efficient** and used less than get.

Response Status Code

- The first line of the response message (i.e., the status line) contains the response status code, which is generated by the server to indicate the outcome of the request.
- The status code is a 3-digit number:
  - 1xx (Informational): Request received, server is continuing the process.
  - 2xx (Success): The request was successfully received, understood, accepted and serviced.
  - 3xx (Redirection): Further action must be taken in order to complete the request.
  - 4xx (Client Error): The request contains bad syntax or cannot be understood.
  - 5xx (Server Error): The server failed to fulfill an apparently valid request.

commonly encountered status codes

- 100 Continue: The server received the request and in the process of giving the response.
- 200 OK: The request is fulfilled.
- 301 Move Permanently: The resource requested for has been permanently moved to a new location. The URL of the new location is given in the response header called Location. The client should issue a new request to the new location. Application should update all references to this new location.
- 302 Found & Redirect (or Move Temporarily): Same as 301, but the new location is temporarily in nature. The client should issue a new request, but applications need not update the references.
- 304 Not Modified: In response to the If-Modified-Since conditional GET request, the server notifies that the resource requested has not been modified.
- 400 Bad Request: Server could not interpret or understand the request, probably syntax error in the request message.
- 401 Authentication Required: The requested resource is protected, and require client's credential (username/password). The client should re-submit the request with his credential (username/password).
- 403 Forbidden: Server refuses to supply the resource, regardless of identity of client.
- 404 Not Found: The requested resource cannot be found in the server.
- 405 Method Not Allowed: The request method used, e.g., POST, PUT, DELETE, is a valid method. However, the server does not allow that method for the resource requested.
- 408 Request Timeout:
- 414 Request URI too Large:
- 500 Internal Server Error: Server is confused, often caused by an error in the server-side program responding to the request.
- 501 Method Not Implemented: The request method used is invalid (could be caused by a typing error, e.g., "GET" misspell as "Get").
- 502 Bad Gateway: Proxy or Gateway indicates that it receives a bad response from the upstream server.
- 503 Service Unavailable: Server cannot response due to overloading or maintenance. The client can try again later.
- 504 Gateway Timeout: Proxy or Gateway indicates that it receives a timeout from an upstream server.

Servlet Api

- The javax.servlet and javax.servlet.http packag es provide interfaces and classes for writing servlets. All servlets must implement the **Servlet interface**, which defines lifecycle methods.
- When implementing a generic service, extend the GenericServlet class.
- For Http protocol extend the HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

Servlet Interface

- **Servlet interface** provides common behaviour 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.

Servlet interface methods

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

GenericServlet Class

- **GenericServlet** class implements **Servlet**, **ServletConfig** and **Serializable** interfaces.
- GenericServlet class can handle any type of request so it is protocol-independent.
- You may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

HttpServlet class

- The HttpServlet class extends the GenericServlet class and implements Serializable interface.
- It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

- **public void service(ServletRequest req,ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
- **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
- **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
- **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
- **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
- **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
- **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
- **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
- **protected void doDelete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
- **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Install Apache Tomcat

- I shall assume that you have created a directory called "c:\myWebProject" (for Windows)
- **STEP 1: Download and Install Tomcat**
- Goto http://tomcat.apache.org ⇒ Under "Tomcat 9.0.{xx} Released" (where {*xx*} is the latest upgrade number) ⇒ Click "Download" ⇒ Under "9.0.{xx}" ⇒ Binary Distributions ⇒ Core ⇒ "**ZIP**" package (e.g., "apache-tomcat-9.0.{*xx*}.**zip**", about 9.8 MB).
- UNZIP the downloaded file into your project directory "c:\myWebProject". Tomcat will be unzipped into directory "c:\myWebProject\apache-tomcat-9.0.{*xx*}".
- For **EASE OF USE**, we shall shorten and rename this directory to "c:\myWebProject\tomcat".
- **Take note of Your Tomcat Installed Directory**. Hereafter, I shall refer to the Tomcat installed directory as <TOMCAT_HOME>.

Contd.

- For academic learning, I recommend "zip" (or "tar.gz") version, as you could simply delete the entire directory when Tomcat is no longer needed (without running any un-installer). You are free to move or rename the Tomcat's installed directory. You can install (unzip) multiple copies of Tomcat in the same machine.

Set JAVA_HOME

- You need to create an *environment variable* (system variable available to all applications) called "JAVA_HOME", and set it to your JDK installed directory.
- First, find your JDK installed directory. The default is "c:\Program Files\Java\jdk-9.0.{xx}", where {xx} is the upgrade number. Take note of your JDK installed directory.
- To set the environment variable JAVA_HOME in Windows 10/8/7: Launch "Control Panel" ⇒ System and Security (Optional) ⇒ System ⇒ Advanced system settings ⇒ Switch to "Advanced" tab ⇒ Environment Variables ⇒ System Variables (the bottom panel) ⇒ "New" ⇒ In "Variable Name", enter "JAVA_HOME" ⇒ In "Variable Value", enter your JDK installed directory you noted in Step 1 (for Windows 10: use "Browse Directory" and select the JDK installed directory).
- To verify, **RE-START** a CMD (restart needed to refresh the environment) and issue:**SET JAVA_HOME**
  JAVA_HOME=c:\Program Files\Java\jdk-9.0.{xx} <== Verify that this is YOUR JDK installed directory

Configure Tomcat

- The Tomcat configuration files are located in the "conf" sub-directory of your Tomcat installed directory, e.g. "c:\myWebProject\tomcat\conf" (for Windows)
  - server.xml
  - web.xml
  - context.xml
- **Make a BACKUP of the configuration files before you proceed!!!**

Server.xml

- **"conf\server.xml" - Set the TCP Port Number**
- Use a programming text editor (e.g., NotePad++, TextPad, Sublime, Atom for Windows) to open the configuration file "server.xml", under the "conf" sub-directory of Tomcat installed directory.
- The default TCP port number configured in Tomcat is 8080, you may choose any number between 1024 and 65535, which is not used by an existing application. We shall choose 9999 in this article. (For production server, you should use port 80, which is pre-assigned to HTTP server as the default port number.)
- Locate the following lines (around Line 69) that define the HTTP connector, and change port="8080" to port="9999".

Web.xml

- **"conf\web.xml" - Enabling Directory Listing**
- Again, use a programming text editor to open the configuration file "web.xml", under the "conf" sub-directory of Tomcat installed directory.
- We shall enable directory listing by changing "listings" from "false" to "true" for the "default" servlet. This is handy for test system, but not for production system for security reasons.
- Locate the following lines (around Line 108) that define the "default" servlet; and change the "listings" from "false" to "true".

Context.xml

- **"conf\context.xml" - Enabling Automatic Reload**
- We shall add the attribute reloadable="true" to the <Context> element to enable automatic reload after code changes. Again, this is handy for test system but not for production, due to the overhead of detecting changes.
- Locate the <Context> start element (around Line 19), and change it to <Context reloadable="true">.

Starting Tomcat

- The Tomcat's executable programs and scripts are kept in the "bin" sub-directory of the Tomcat installed directory.
- **Start Server (For Windows)**
- I shall assume that Tomcat is installed in "c:\myWebProject\tomcat". Launch a CMD shell and issue:
- **c:** // Change drive **cd \myWebProject\tomcat\bin** // Change directory to your Tomcat's binary directory **startup** // Start tomcat server

Start a Client to Access the Server

- Start a browser (Firefox, Chrome) as an HTTP client. Issue URL "http://localhost:9999" to access the Tomcat server's welcome page. The hostname "localhost" (with IP address of 127.0.0.1) is meant for local loop-back testing inside the same machine. For users on the other machines over the net, they have to use the server's IP address or DNS domain name or hostname in the format of "http://*serverHostnameOrIPAddress*:9999".
- Try issuing URL http://localhost:9999/examples to view the servlet and JSP examples. Try running some of the servlet examples.

Shutdown Server(For Windows)

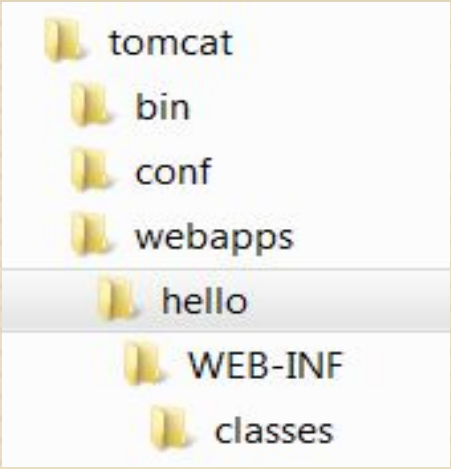
- You can shutdown the tomcat server by either:
- Press Ctrl-C on the Tomcat console; OR
- Run "<TOMCAT_HOME>\bin**shutdown.bat**" script. Open a new "cmd" and issue: **c:** // Change the current drive **cd \myWebProject\tomcat\bin** // Change directory to your Tomcat's binary directory **shutdown** // Shutdown the server

Steps to create servlet

- we are going to use **apache tomcat server** in this example. The steps are as follows:
  - Create a directory structure
  - Create a Servlet
  - Compile the Servlet
  - Create a deployment descriptor
  - Start the server and deploy the project
  - Access the servlet

Directory Structure for WebApp

- Let's call our first webapp "hello". Goto Tomcat's "webapps" sub-directory and create the following directory structure for your webapp "hello" (as illustrated):
  - Under Tomcat's "webapps", create your webapp's *root* directory "hello" (i.e., "<TOMCAT_HOME>\webapps\hello").
  - Under "hello", create a sub-directory "WEB-INF" (case sensitive, a "dash" not an underscore) (i.e., "<TOMCAT_HOME>\webapps\hello\WEB-INF").
  - Under "WEB-INF", create a sub-sub-directory "classes" (case sensitive, plural) (i.e., "<TOMCAT_HOME>\webapps\hello\WEB-INF\classes").



Contd.

- You need to keep your web resources (e.g., HTMLs, CSSs, images, scripts, servlets, JSPs) in the proper directories:
  - "hello": The is called the *context root* (or *document base directory*) of your webapp. You should keep all your HTML files and resources visible to the web users (e.g., HTMLs, CSSs, images, scripts, JSPs) under this *context root*.
  - "hello/WEB-INF": This directory, although under the context root, is *not visible* to the web users. This is where you keep your application's web descriptor file "web.xml".
  - "hello/WEB-INF/classes": This is where you keep all the Java classes such as servlet class-files.
- You should RE-START your Tomcat server to pick up the hello webapp. Check the Tomcat's console to confirm that "hello" application has been properly deployed
- You can issue the following URL to access the web application "hello":
- http://localhost:9999/**hello**
- You should see the directory listing of the directory "<TOMCAT_HOME>\webapps\hello", which shall be empty at this point of time. (Take note that we have earlier enabled directory listing in "web.xml". Otherwise, you will get an error "404 Not Found").

Compile the servlet

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in **WEB-INF/classes** directory.

Create the servlet-HelloServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class HelloServlet 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! Hello World!!!");
pw.println("</body></html>");

pw.close();//closing the stream
}}
```

Create the deployment descriptor (web.xml file)

- It is an xml file, from which Web Container gets the information about the servlet to be invoked.
- The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.
- There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

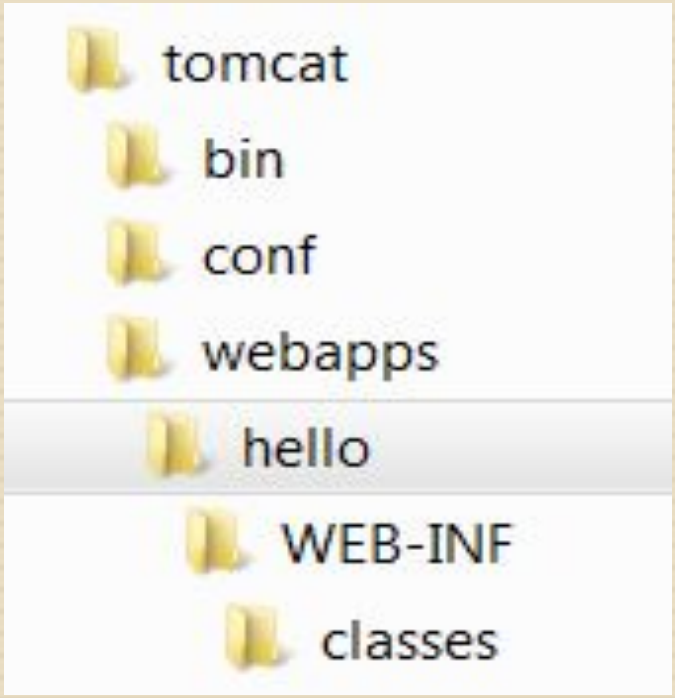
- <web-app>
-
- <servlet>
- <servlet-name>welcome</servlet-name>
- <servlet-class>HelloServlet</servlet-class>
- </servlet>
-
- <servlet-mapping>
- <servlet-name>welcome</servlet-name>
- <url-pattern>/hello</url-pattern>
- </servlet-mapping>
-
- </web-app>

Deploy and run

- Copy the directory structure under the webapps folder of tomcat
- Start the server
- Open browser and write http://hostname:portno/contextroot/urlpattern ofservlet.
- For example:
- http://localhost:9999/hello/hello

Helloworld example Revisted

- Directory structure



Helloworld example

- Step 1 --Create a HTML file (Next Slide)
  - Put this file under hello directory
  - Check the result of
    - http://localhost:9999/hello/**HelloHome.html**
    - http://localhost:9999/hello

    Modify the name to index.html

    and use http://localhost:9999/hello

Write a Welcome Page

HelloHome.html

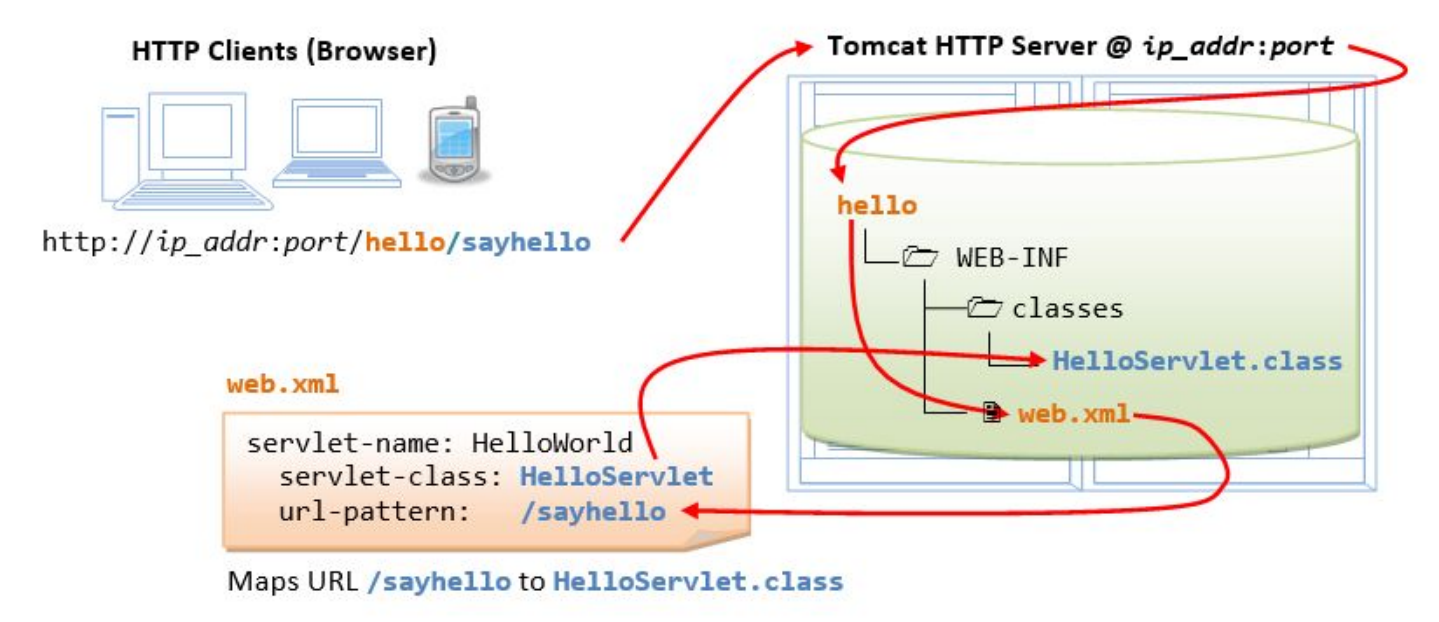
```
<html>
    <head>
        <title>My Home Page</title>
    </head>
    <body>
        <form method="get" action="http://localhost:9999/hello/sayhello">
    <b>Enter your Name:</b> <input type="text" name="author" value="Name">
    <input type="submit" value="Submit"> </form>
    </body>
</html>
```

Step2 **Write a "Hello-world" Java Servlet**

```
import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;
public class HelloServlet extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
 try {
        out.println("<html>");
        out.println("<head><title>Hello, World</title></head>");
        out.println("<body>");
        out.println("<h1>Hello, world!</h1>");
        out.println("</body></html>");
} finally { out.close(); // Always close the output writer }
 }//end doGet()
 }//End HelloServlet
```

Compile the HelloServlet

- Put servlet-api.jar in your class path or
- **javac -cp .;c:\myWebProject\tomcat\lib\servlet-api.jar HelloServlet.java**



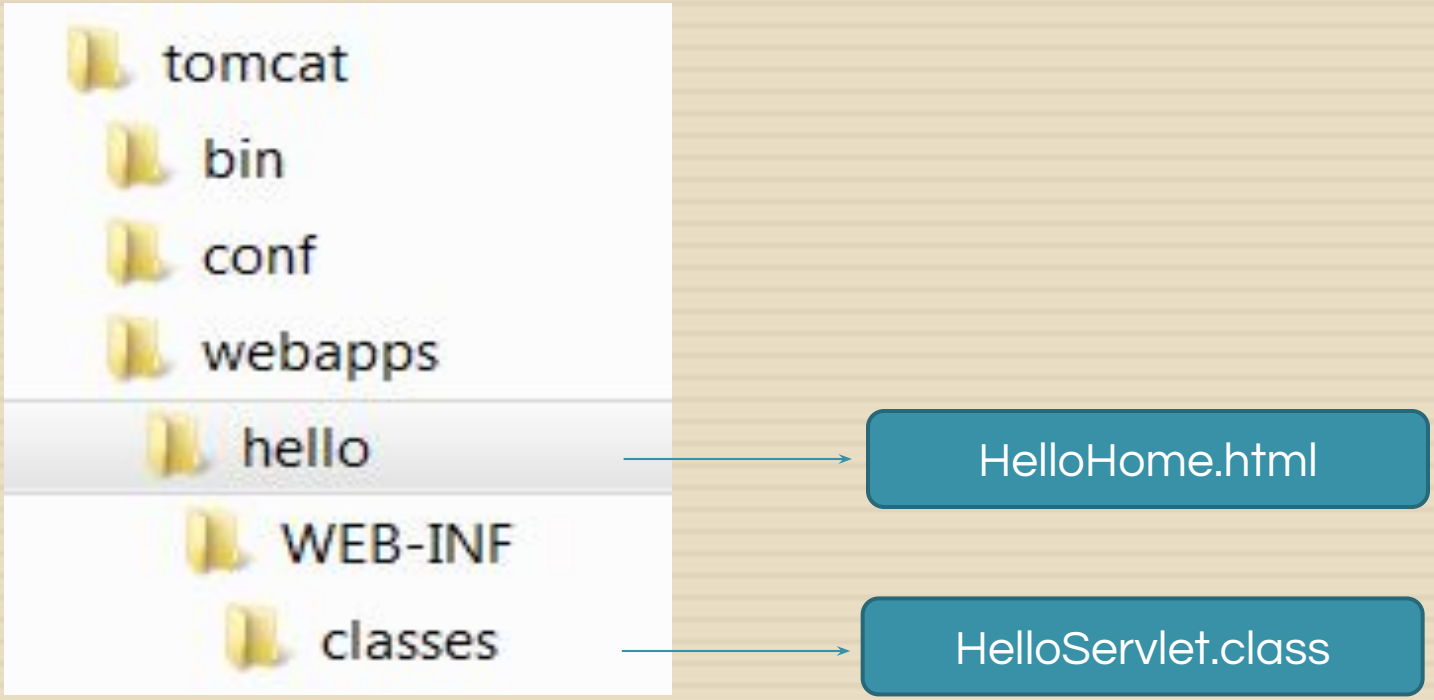
Deployment Descripter

```
1   <?xml version="1.0" encoding="ISO-8859-1"?>
2   <web-app version="3.0“
3        xmlns="http://java.sun.com/xml/ns/javaee"
4        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
6
7     <!-- To save as "hello\WEB-INF\web.xml" -->
8
9      <servlet>
10       <servlet-name>HelloWorld</servlet-name>
11       <servlet-class>HelloServlet</servlet-class>
12     </servlet>
13
14    <!-- Note: All <servlet> elements MUST be grouped together and
15     placed IN FRONT of the <servlet-mapping> elements -->
16
17     <servlet-mapping>
18       <servlet-name>HelloWorld</servlet-name>
19       <url-pattern>/sayhello</url-pattern>
20     </servlet-mapping>
21  </web-app>
```

Create a web.cml file. Wrongly created web.xml file will cause errors during loading servlet

Deploy your Hello World Project

- The directory structure now in tomcat folder is



Run and Execute

- Start Tomcat server
  - Open Browser and issue this url:
    - http://localhost:9999/hello/**HelloHome.html**
    - **Fill the entry and click submit.**
    - **It calls the Helloservlet which dynamically creates the page and sends you the response.**

    ……DO MORE EXCERCISES….