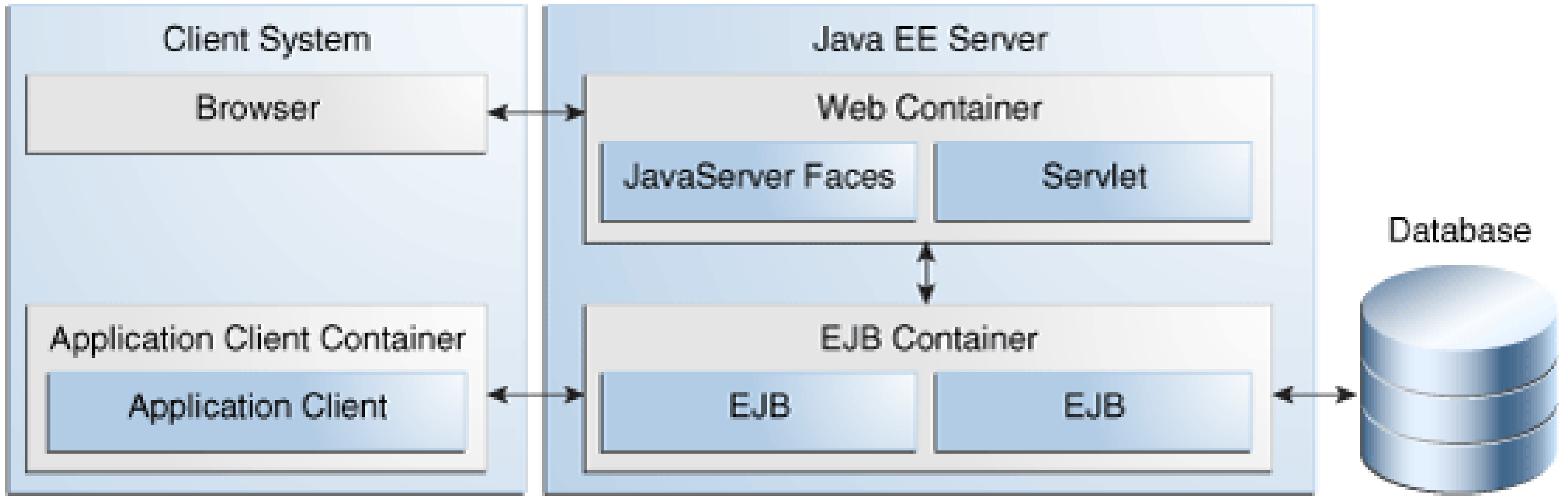
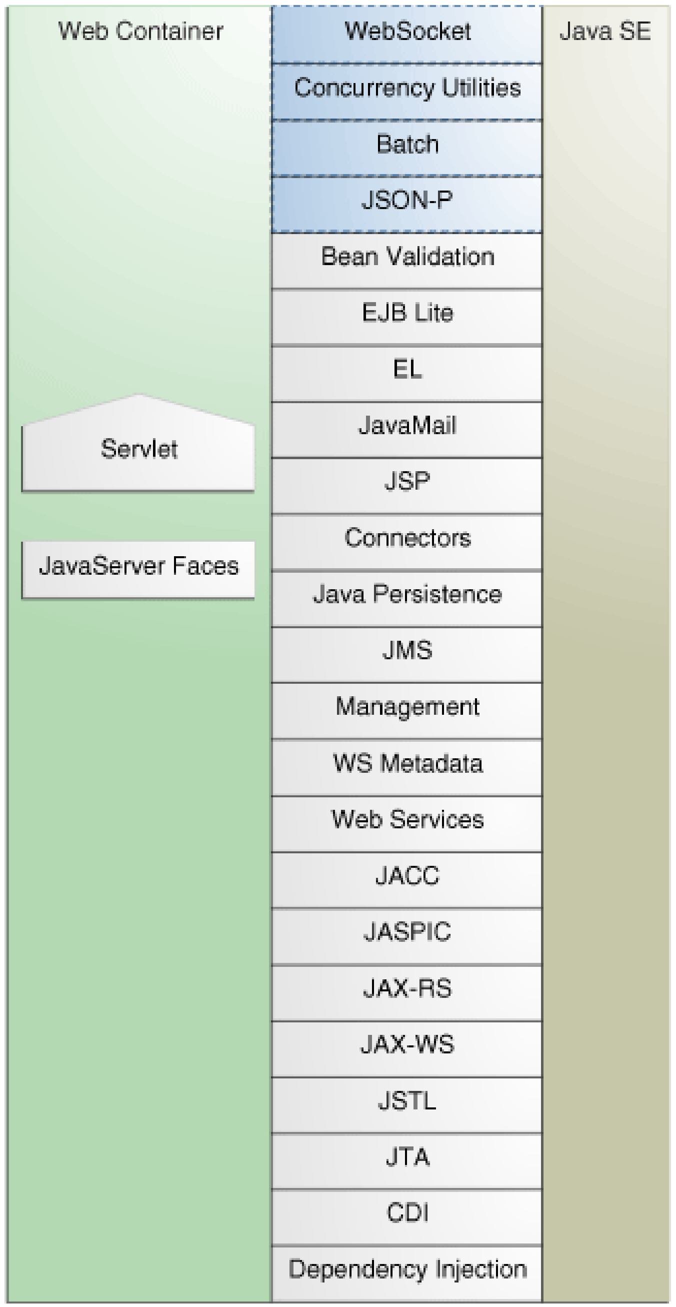
Java EE

- Introduction To Web Technologies
  - **Java EE Platform**
  - **Application Servers & Web Servers**

Java EE Platform

- The Java EE platform is built on top of the Java SE platform.
- The Java EE platform provides an API and runtime environment for developing and running large-scale, multi-tiered, scalable, reliable, and secure network applications.
- A shorthand name for such applications is "enterprise applications," so called because these applications are designed to solve the problems encountered by large enterprises.





Multi-tiered Applications

- Typically, multi-tiered applications have a client tier, a middle tier, and a data tier (often called the enterprise information systems tier).
- The client tier consists of a client program that makes requests to the middle tier.
- Clients can be a web browser, a standalone application, or other servers, and they run on a different machine from the Java EE server.
- The middle tier's business functions handle client requests and process application data, storing it in a permanent data store in the data tier.
- The following Java EE Technologies are used in the business tier in Java EE applications:
  - Enterprise JavaBeans (enterprise bean) components
  - JAX-RS RESTful web services
  - JAX-WS web service endpoints
  - Java Persistence API entities

Multi-tiered Applications

- The enterprise information_systems (EIS) tier also called as data tier consists of database servers, enterprise resource planning systems, and other legacy data sources, like mainframes. These resources typically are located on a separate machine than the Java EE server, and are accessed by components on the business tier.

- The following Java EE technologies are used to access the EIS tier in Java EE applications:
  - The Java Database Connectivity API (JDBC)
  - The Java Persistence API
  - The Java EE Connector Architecture
  - The Java Transaction API (JTA)

- *Java EE application development concentrates on the middle tier* to make enterprise application management easier, more robust, and more secure.

The Web Tier

The web tier consists of components that handle the interaction between clients and the business tier.

Its primary tasks are the following:

- Dynamically generate content in various formats for the client.
- Collect input from users of the client interface and return appropriate results from the components in the business tier.
- Control the flow of screens or pages on the client.
- Maintain the state of data for a user's session.
- Perform some basic logic and hold some data temporarily in JavaBeans components.

Java EE Technologies Used in the Web Tier

Technology	Purpose
Servlets	Java programming language classes that dynamically process requests and construct responses, usually for HTML pages
Java Server Faces (JSF) Technology	A user-interface component framework for web applications that allows you to include UI components (such as fields and buttons) on a page, convert and validate UI component data, save UI component data to server-side data stores, and maintain component state.
JSF Facelets Technology	Facelets applications are a type of Java Server Faces applications that use XHTML pages rather than JSP pages.
Expression Language (EL)	A set of standard tags used in JSP and Facelets pages to refer to Java EE components.
Java Server Pages (JSP)	Text-based documents that are compiled into servlets and define how dynamic content can be added to static pages, such as HTML pages.
JavaServer Pages Standard Tag Library (JSTL)	A tag library that encapsulates core functionality common to JSP pages
Java Beans Components	Objects that act as temporary data stores for the pages of an application

Java Application Server Vs Web Server

Application servers serve application data to clients, much like Web servers serve web pages to web browsers.

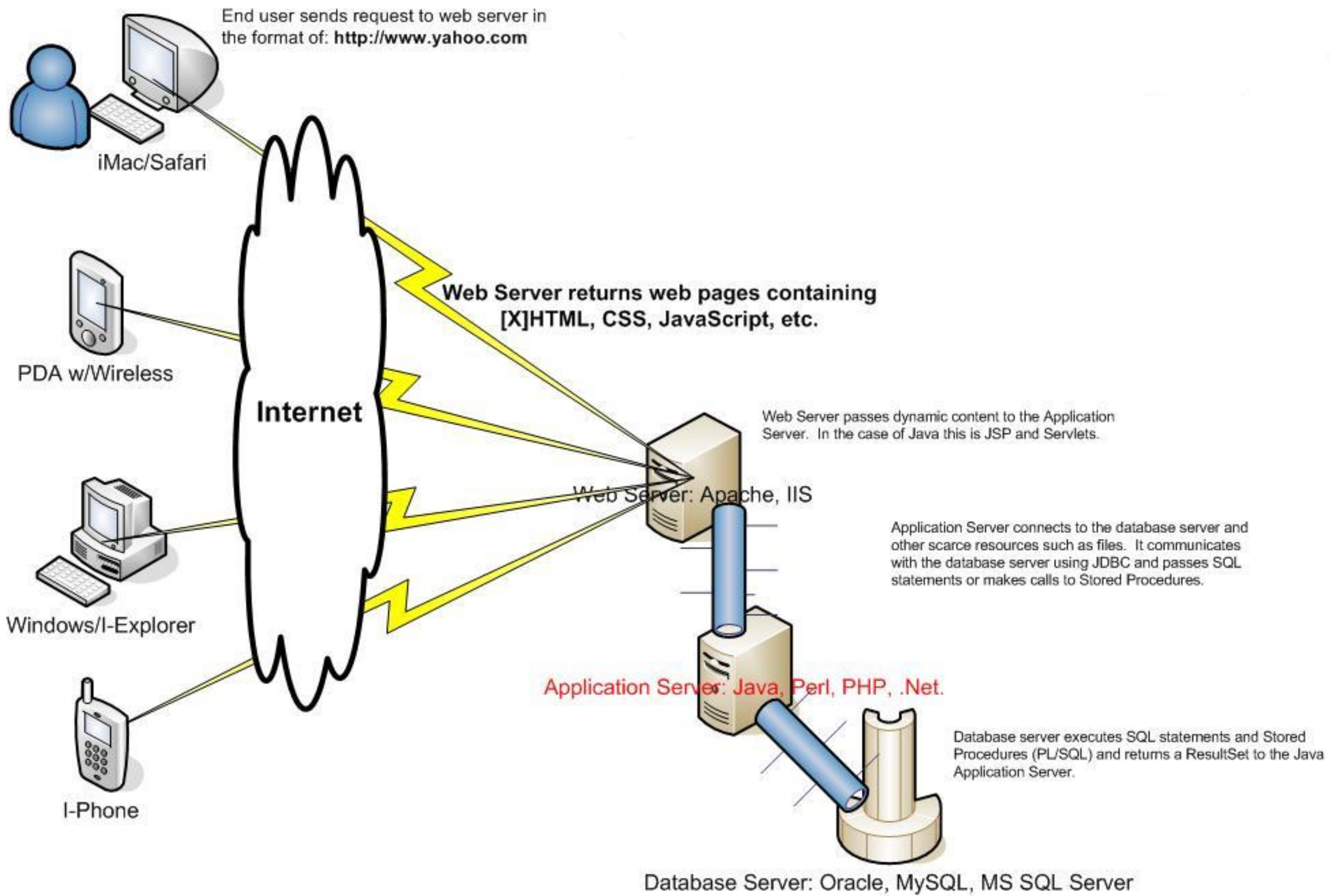
An application server exposes **business logic** to client applications through various protocols, possibly including HTTP, which means web server is part of Application Server.

Also called as ***app server*** provides applications with services such as *security, data services, transaction support, load balancing, and management of large distributed systems.*

Application Servers typically contain web container, EJB container, Java Messaging Services(JMS), Web Services etc.

Some Java Application Servers leave off many Java EE features like EJB and JMS including Tomcat from Apache, and Jetty from Eclipse Foundation. *These are called as web containers or web servers*. Their focus is more on Java Servlets and Java Server Pages.

Java Application Server


End user sends request to web server in the format of: **http://www.yahoo.com**
iMac/Safari
PDA w/Wireless
Windows/I-Explorer
I-Phone
Internet
Web Server returns web pages containing [X]HTML, CSS, JavaScript, etc.
Web Server passes dynamic content to the Application Server. In the case of Java this is JSP and Servlets.
Web Server: Apache, IIS
Application Server connects to the database server and other scarce resources such as files. It communicates with the database server using JDBC and passes SQL statements or makes calls to Stored Procedures.
Application Server: Java, Perl, PHP, .Net.
Database server executes SQL statements and Stored Procedures (PL/SQL) and returns a ResultSet to the Java Application Server.
Database Server: Oracle, MySQL, MS SQL Server

Java Application Servers

Commercial, non-open source App Servers:

- BEA Weblogic (taken over by Oracle Corp.)
- IBM Websphere
- Microsoft ‘s Windows Server 2012
- Oracle's Oracle Application Server 4.0,

Non-Commercial, open source App/Web Servers:

- JBoss AS from JBoss(division of Red Hat)
- TomEE from Apache
- GlassFish from Oracle

Open Source Web Containers

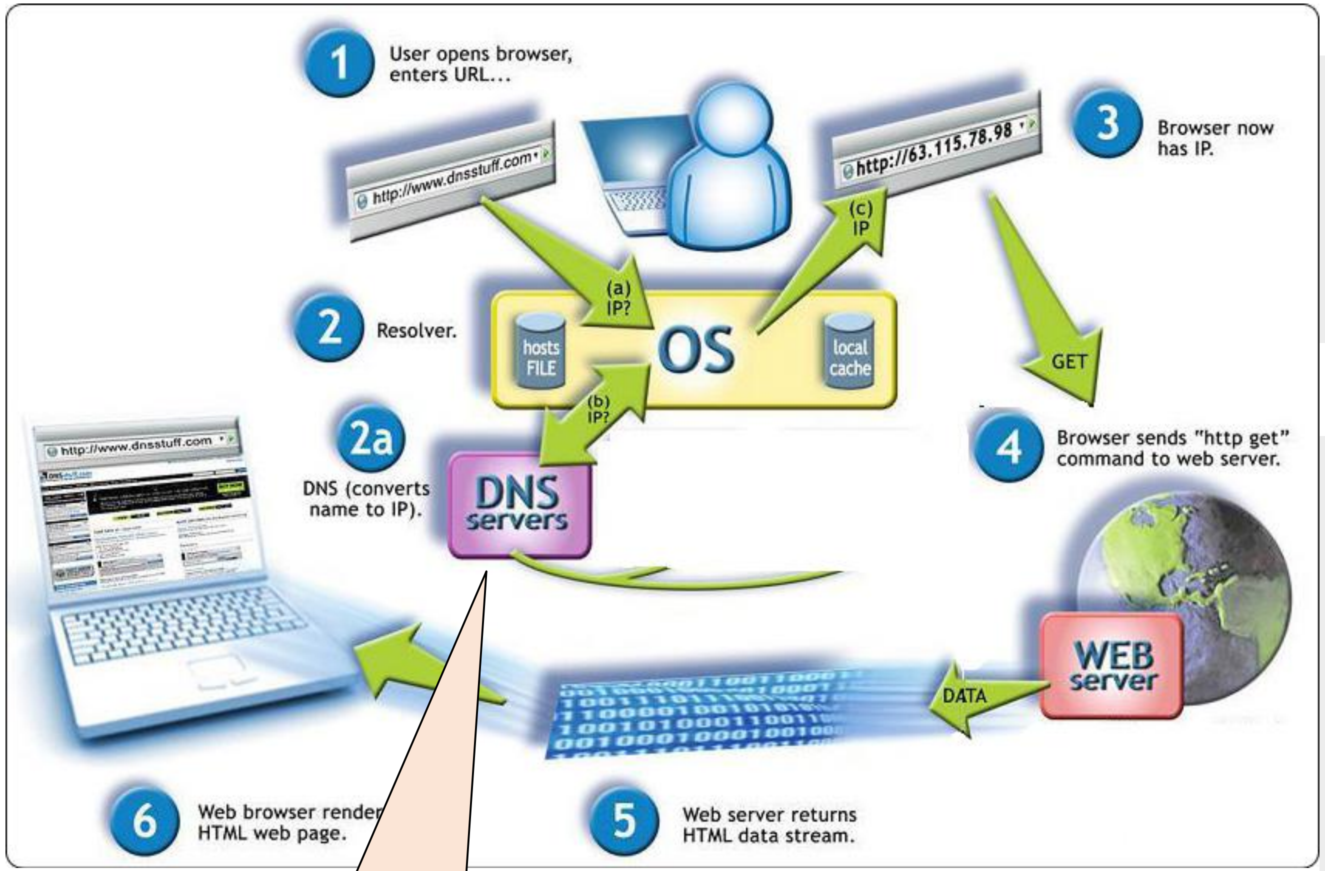
- **Apache Tomcat**
- **Jetty**

Note: **Microsoft Application Server :** Microsoft positions their middle-tier applications and services infrastructure in the Windows Server operating system and the .NET Framework technologies in the role of an application server

How Internet Works

- Internet uses client/server technology for its working.
- The World Wide Web (WWW) uses the browser as the client software and Web Server as the server software.
- The user types the required URL in the browser. The IP address of the server is found from the URL.
- The Web Server will be listening in that Server at Port No. 80
- The browser connects to Port N0. 80 of the specified Server.
- *Upon receiving the request, 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.

How Internet Works


1 User opens browser, enters URL...
http://www.dnsstuff.com
(a) IP?
2 Resolver.
hosts FILE
OS
local cache
(b) IP?
2a DNS (converts name to IP).
DNS servers
(c) IP
http://63.115.78.98
3 Browser now has IP.
GET
4 Browser sends "http get" command to web server.
WEB server
DATA
5 Web server returns HTML data stream.
6 Web browser renders HTML web page.
http://www.dnsstuff.com

Domain Name System (DNS)

A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web.

URL has the following syntax:

protocol*://*hostname*:*port*/*path-and-file-name

DNS Servers convert the domain name to its IP Address.

For ex. If you type the URL: http://www.google.com, the DNS Server will convert to IP address:

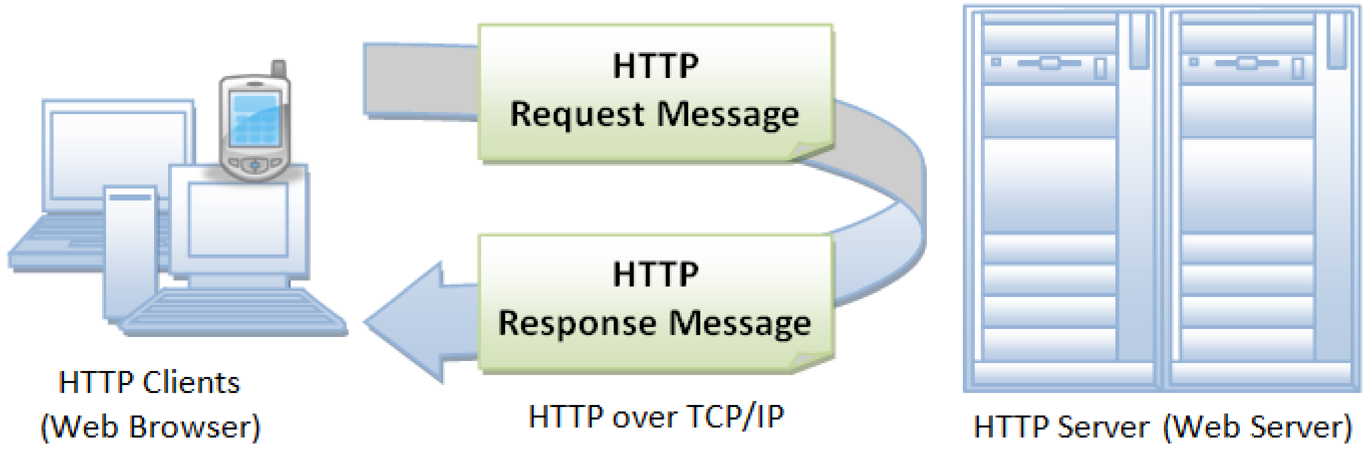
http://74.125.224.72/

ip address format:

xxx.xxx.xxx.xxx where xxx will be between 0 to 255

HTTP (Hyper Text Transfer Protocol)

The client and server interact with each other by *exchanging messages* using a protocol called HTTP (**H**yper **T**ext **T**ransfer **P**rotocol).



•HTTP is an *asymmetric request-response protocol .*

•The connection is asymmetric - the client and server use different message vocabularies; the client issuing *commands* and the server sending *replies*. So, the client *pulls* information from the server, instead of server *pushing* information to the client.

•The client initiates the TCP connection. The server listens on a TCP port number(8080), and clients connect to the server.

Java EE

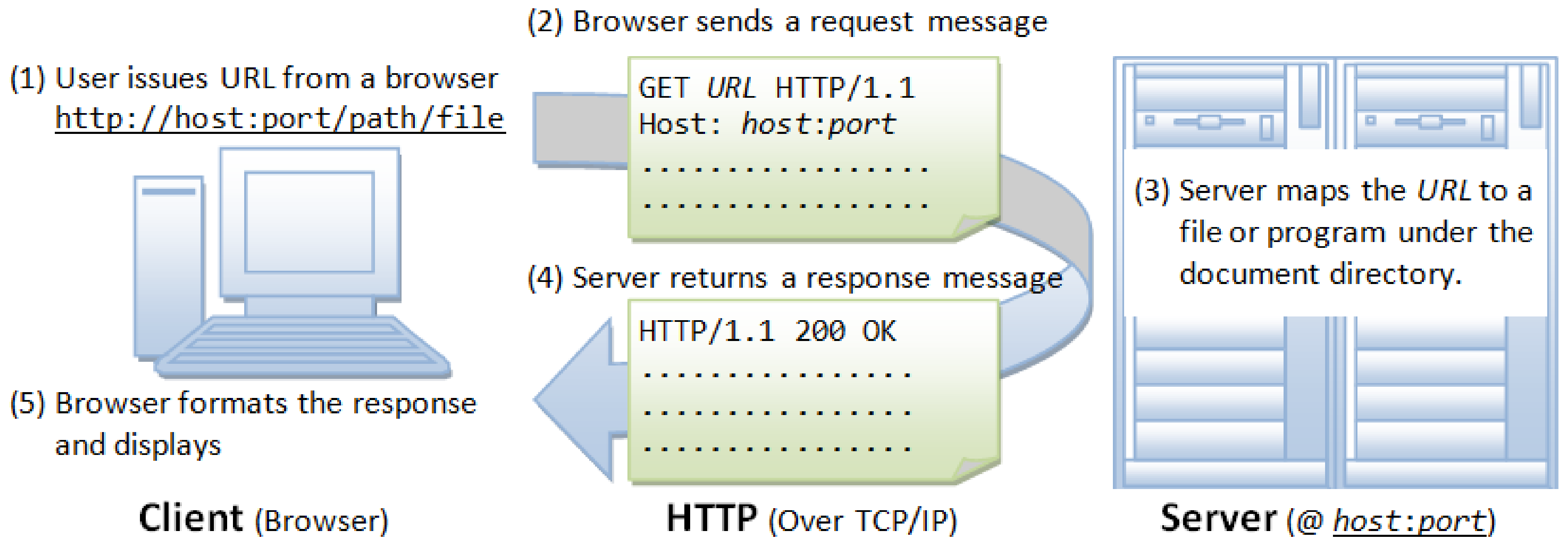
HTTP Request Header

HTTP Response Header

HTTP Request and HTTP Response

The user types URL on the browser. But according to HTTP protocol, the request should contain some more information in a special format.

Since the browser follows the HTTP protocol, it will automatically create the *HTTP Request header* in the specific format using the URL typed by the user and adding the extra information.



(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)

HTTP Request Header & HTTP Response Header

- The HTTP Request header is the information client’s browser sends to a Web server. It contains details of what the browser wants and will accept back from the server.
- The request header also contains the type, version and capabilities of the browser that is making the request so that server returns compatible data.
- Upon receipt of the request header, the server will return an HTTP response header to the client that is attached to the file(s) being sent.
- The response header contains the date, size and type of file that the server is sending back to the client and also data about the server itself.

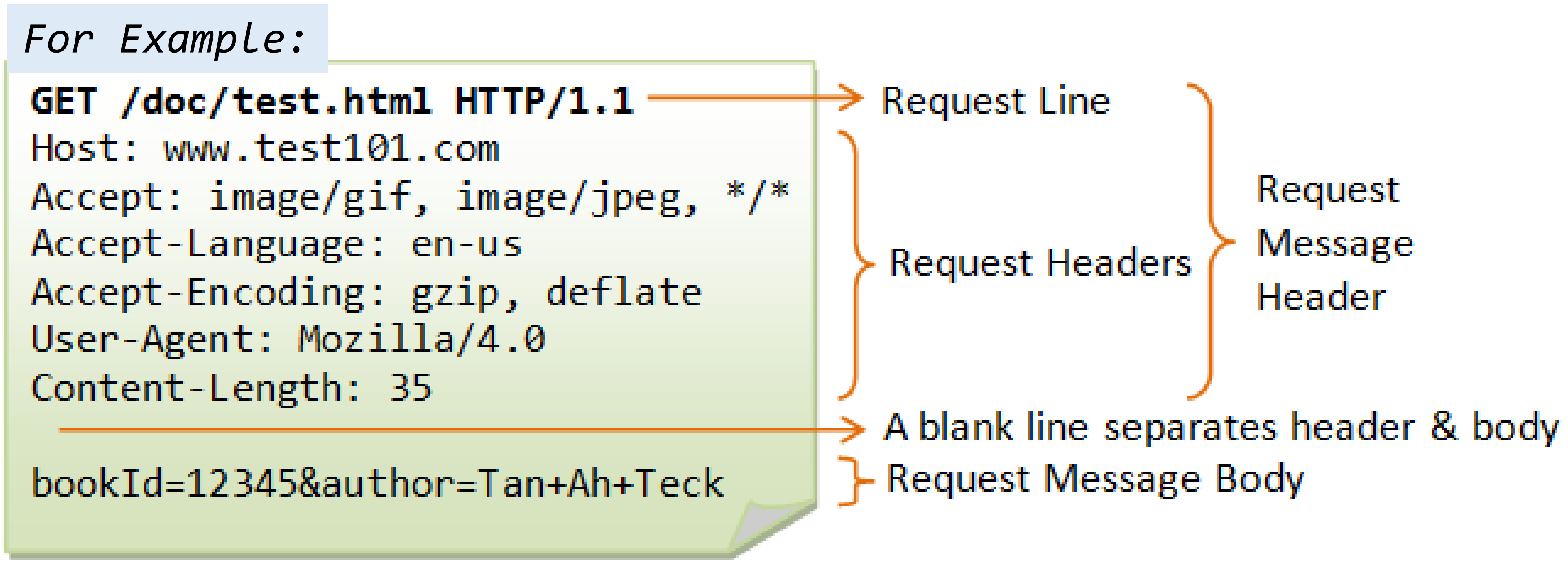
HTTP Request Header

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, ...`



For Example:

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

HTTP Response Header

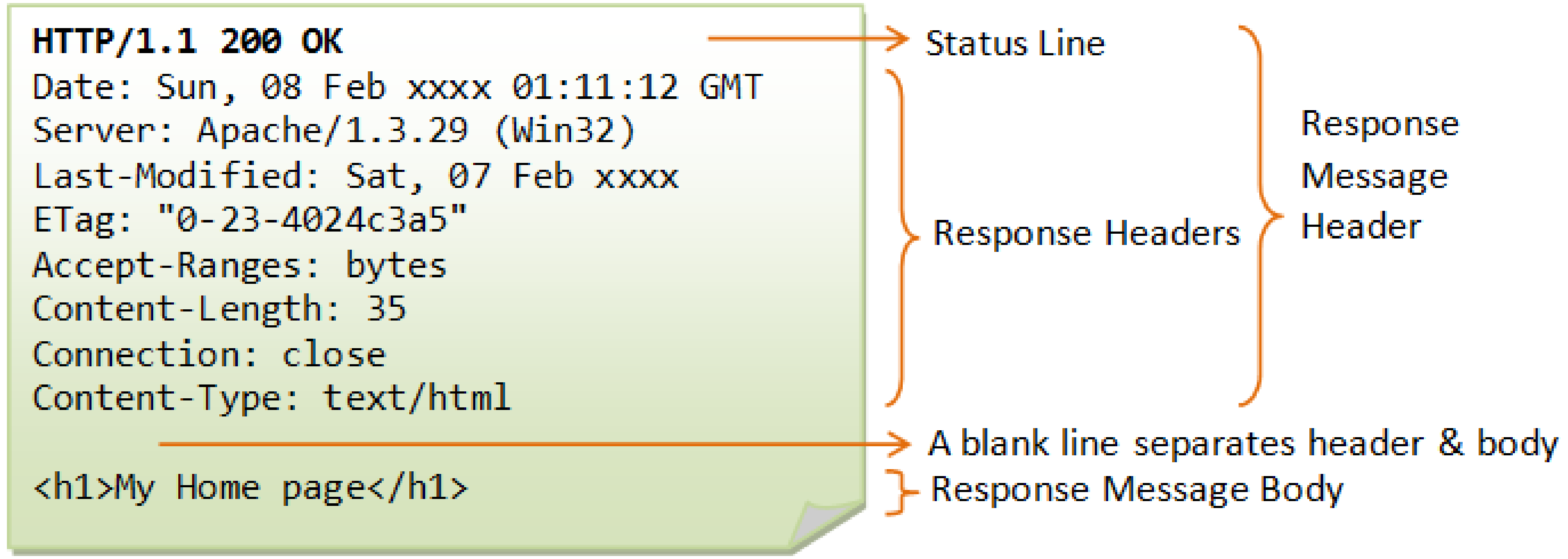
Response Headers

The response headers are in the form of name:value pairs.

Multiple values, separated by commas, can be specified.

`response-header-name: response-header-value1, response-header-value2, ...`

`For Example:`



```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

HTTP Request Methods

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 Request Method

"GET" Request Method

GET is the most common HTTP request method. A client can use the GET request method to request (or "get") for a piece of resource from an HTTP server.

A GET request message takes the following syntax:

```
GET request-URI HTTP-version (optional request headers)
(blank line)
(optional request body)
```

•The keyword GET is case sensitive and must be in uppercase.
•*request-URI*: specifies the path of resource requested, which must begin from the root "/" of the document base directory.
•*HTTP-version*: Either HTTP/1.0 or HTTP/1.1. This client *negotiates* the protocol to be used for the current session. For example, the client may request to use HTTP/1.1. If the server does not support HTTP/1.1, it may inform the client in the response to use HTTP/1.0.
•The client uses the optional request headers (such as Accept, Accept-Language, and etc) to *negotiate* with the server and ask the server to deliver the preferred contents (e.g., in the language that the client preferred).
•GET request message has an optional request body which contains the query string (*will be explained later*).

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:

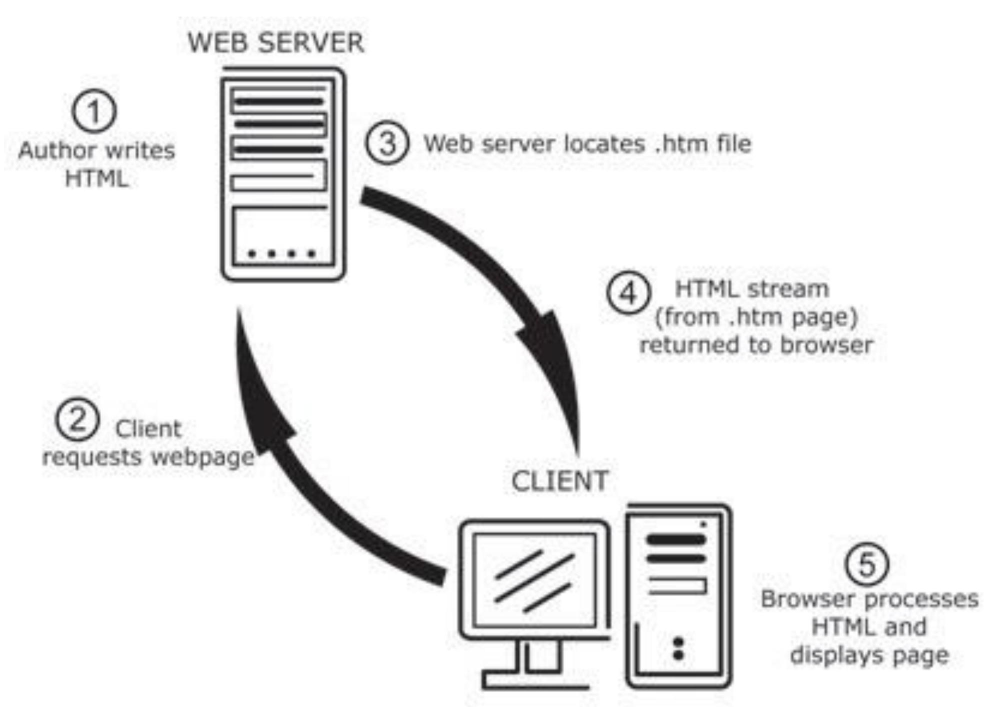
- 1xx (Informational): Request received, server is continuing the process.
- 2xx (Success): The request was successfully received, understood, accepted and serviced.
- 3xx (Redirection): Further action must be taken in order to complete the request.
- 4xx (Client Error): The request contains bad syntax or cannot be understood.
- 5xx (Server Error): The server failed to fulfil an apparently valid request.

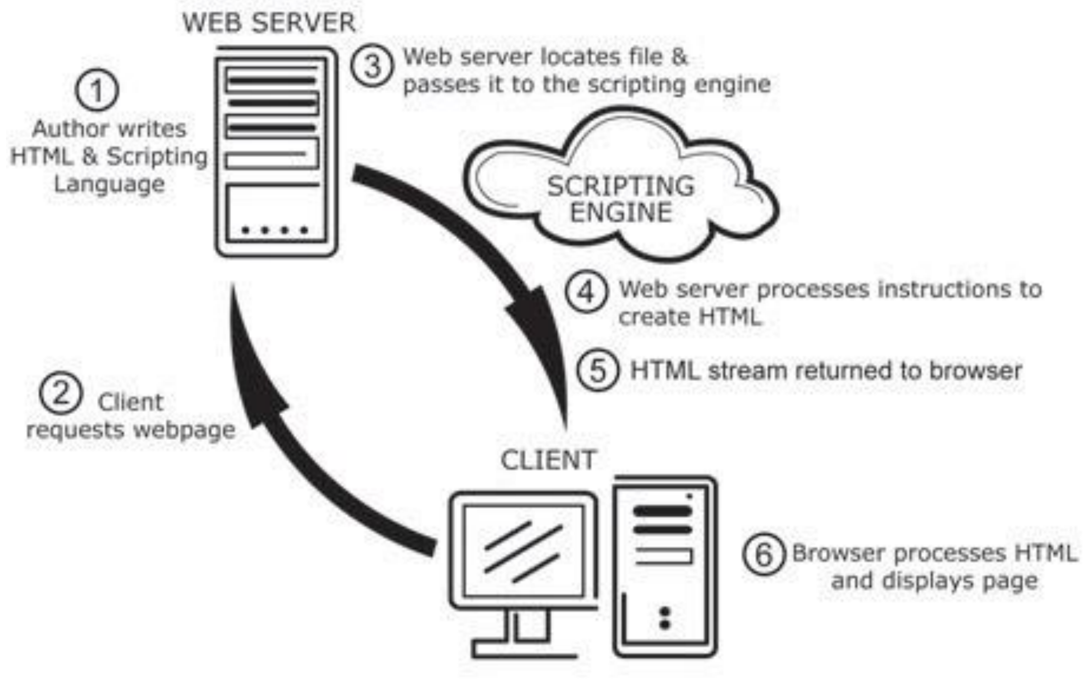
Java EE

- Static Vs Dynamic Web Pages
- Servlet Container
- Creating web application with Eclipse IDE

Static and Dynamic Web Pages

- From the web, we get static pages as well as dynamic pages
- Static page is a page that does not change with user and/or time whereas dynamic pages can change with user and/or time.
- For a static page, all we require at the server side is the Web Server and the HTML files.
- Dynamic sites, on the other hand, construct HTML pages on fly as they are requested, using server side scripting languages such as Servlets, JSP, PHP , ASP etc.
- Depending on the information site visitor requests for, the content for web pages is dynamically generated and if required, retrieving data from the database.


WEB SERVER
1 Author writes HTML
3 Web server locates .htm file
4 HTML stream (from .htm page) returned to browser
2 Client requests webpage
CLIENT
5 Browser processes HTML and displays page


WEB SERVER
3 Web server locates file & passes it to the scripting engine
1 Author writes HTML & Scripting Language
SCRIPTING ENGINE
4 Web server processes instructions to create HTML
5 HTML stream returned to browser
2 Client requests webpage
CLIENT
6 Browser processes HTML and displays page

Server-side Web Development

Java is arguably the most powerful platform for server-side web development today.

The two widely used technologies for developing dynamic web pages are:

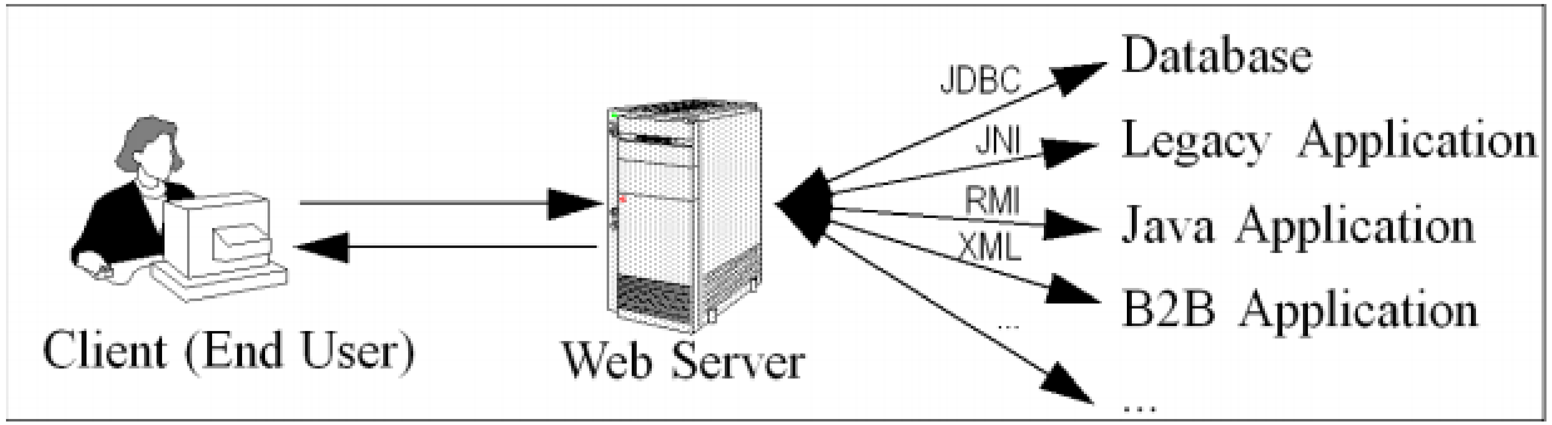
1. Servlets
2. JSP – Java Server Pages

***A* Servlet** is a specialized Java class that runs on a web server to generate dynamic web pages.

Servlets work on a request - response programming model i.e. they accept requests from the clients , generate responses dynamically and send the responses in a format such as ***html*** to the clients/browsers.

A Servlet’s Job

- Read explicit data sent by client (form data)
- Read implicit data sent by client (request headers)
- Generate the results
- Send the explicit data back to client (HTML)
- Send the implicit data to client (status codes and response headers)



Servlet Container

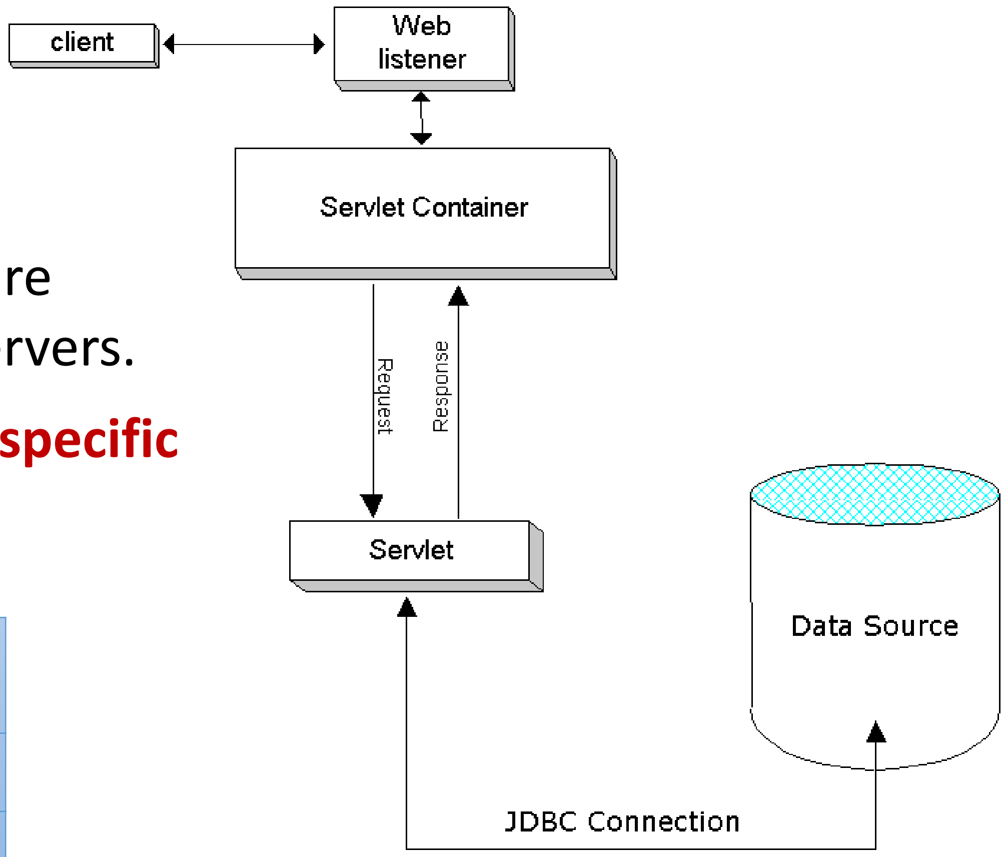
For executing the servlets, the web server requires the help of another piece of software called as servlet container also called as Web Container.

Unlike a Java client program, a servlet has no static main() method. Therefore, a servlet must execute under the control of an external container.

Apache Tomcat is a well-know web server.

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.**



Servlet/JSP Spec	Actual release revision	Minimum Java Version
3.1/2.3	**8.0.15**	**1.7**
3.0/2.2	7.0.35	1.6

Tomcat Server

- Download latest version of tomcat from the following link:
  https://tomcat.apache.org/download-80.cgi

  Note: select 32-bit or 64-bit depending on your OS

- Unzip and copy into your favourite folder, preferably *in c:\program files*

- Create the following system variables:
  - *Right-Click on My Computer->properties->Advanced System Properties->Environmental variables->System Properties*
  - *Click on new button, enter variable name and value:*

    JAVA_HOME
    C:\Program Files\Java\jdk1.7.0_79

    CATALINA_HOME
    C:\apache-tomcat-8.0.35

- Edit path variable and append the following:
  …. ;%JAVA_HOME%\bin;%CATALINA_HOME%\bin

- To verify, go to *C:\apache-tomcat-8.0.35\bin and run startup.bat file*

Creating Web Application

1. **Create Dynamic Web Project**
   File-> New-> Project->Web-> Dynamic Web Project.
   Note: Eclipse remembers the recent types, so once you do this once, you can just do File->New-> Dynamic Web Project.

2. Accept Java EE perspective

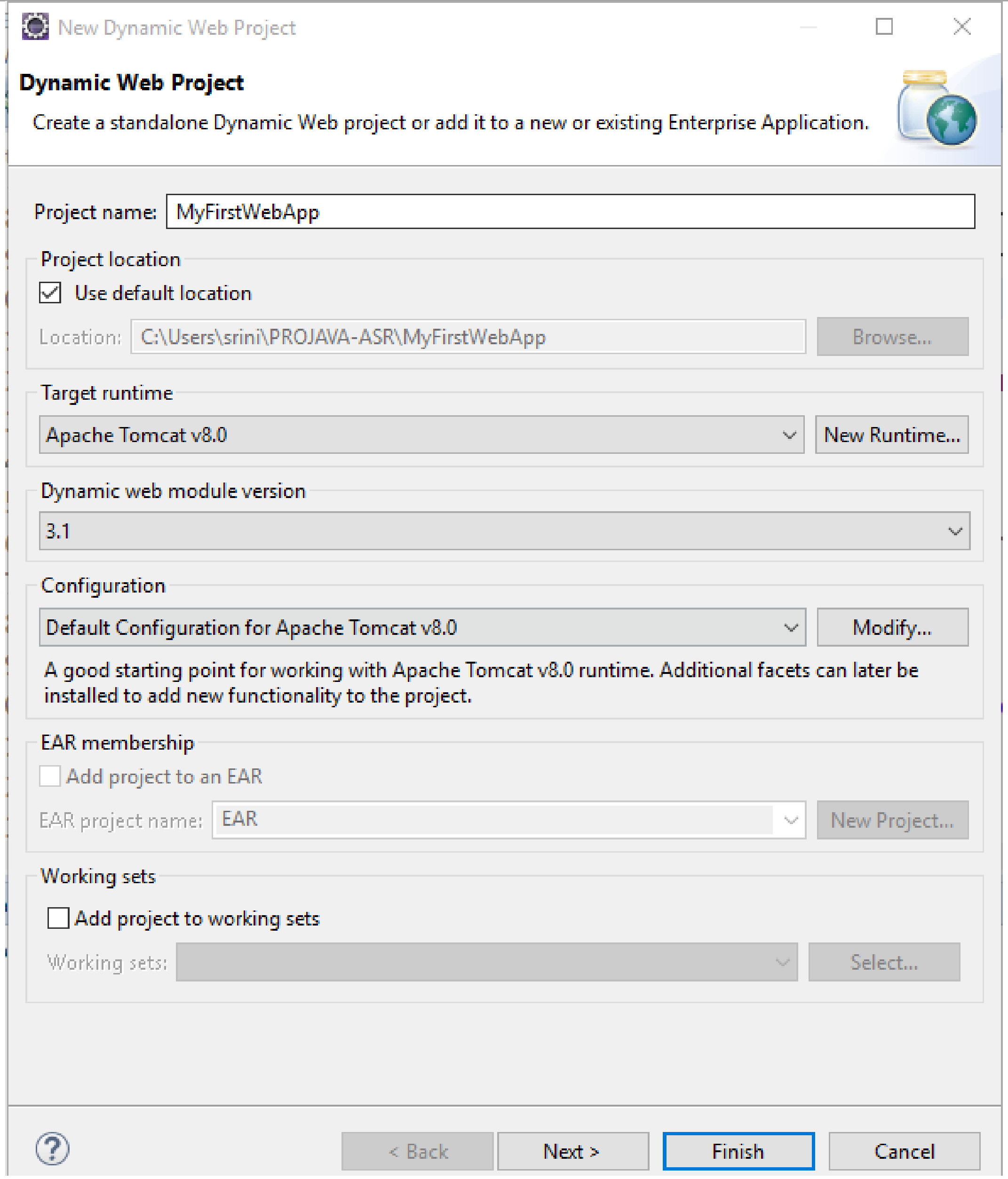
3. For "Target Runtime", choose "Apache Tomcat v8.0“

4. Enter project name (e.g., “MyFirstWebApp").

5. Click on **Nex**t button. *No changes to be made.*

5. Click on **Next** button. You will see the following window:

6. By default, Tomcat Server 8.x uses annotations while creation of *web.xml* is optional.
7. If you want web.xml to be created, click on the check box*(Presently, we require web.xml file so click on the checkbox)* and click on **Finish** button.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: MyFirstWebApp

Project location

[✓] Use default location

Location: C:\Users\srini\PROJAVA-ASR\MyFirstWebApp

Target runtime: Apache Tomcat v8.0

Dynamic web module version: 3.1

Configuration: Default Configuration for Apache Tomcat v8.0

A good starting point for working with Apache Tomcat v8.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership

[ ] Add project to an EAR

EAR project name: EAR

Working sets

[ ] Add project to working sets

Working sets:

Creating Web Application

8. Open web.xml file, you will see few welcome files:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID" version="3.1">
  <display-name>MyFirstWebApp</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
    <welcome-file>index.htm</welcome-file>
    <welcome-file>index.jsp</welcome-file>
    <welcome-file>default.html</welcome-file>
    <welcome-file>default.htm</welcome-file>
    <welcome-file>default.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

The deployment descriptor file, web.xml is an XML file whose root element is <web-app> resides in WEB-INF/ directory.

When the web server receives a request for the application, it uses the deployment descriptor to map the URL of the request to the code (servlet) that is supposed to handle the request.

9. Create ***index.html*** file under ***WebContent*** folder as shown in the next slide.

Creating Web Application

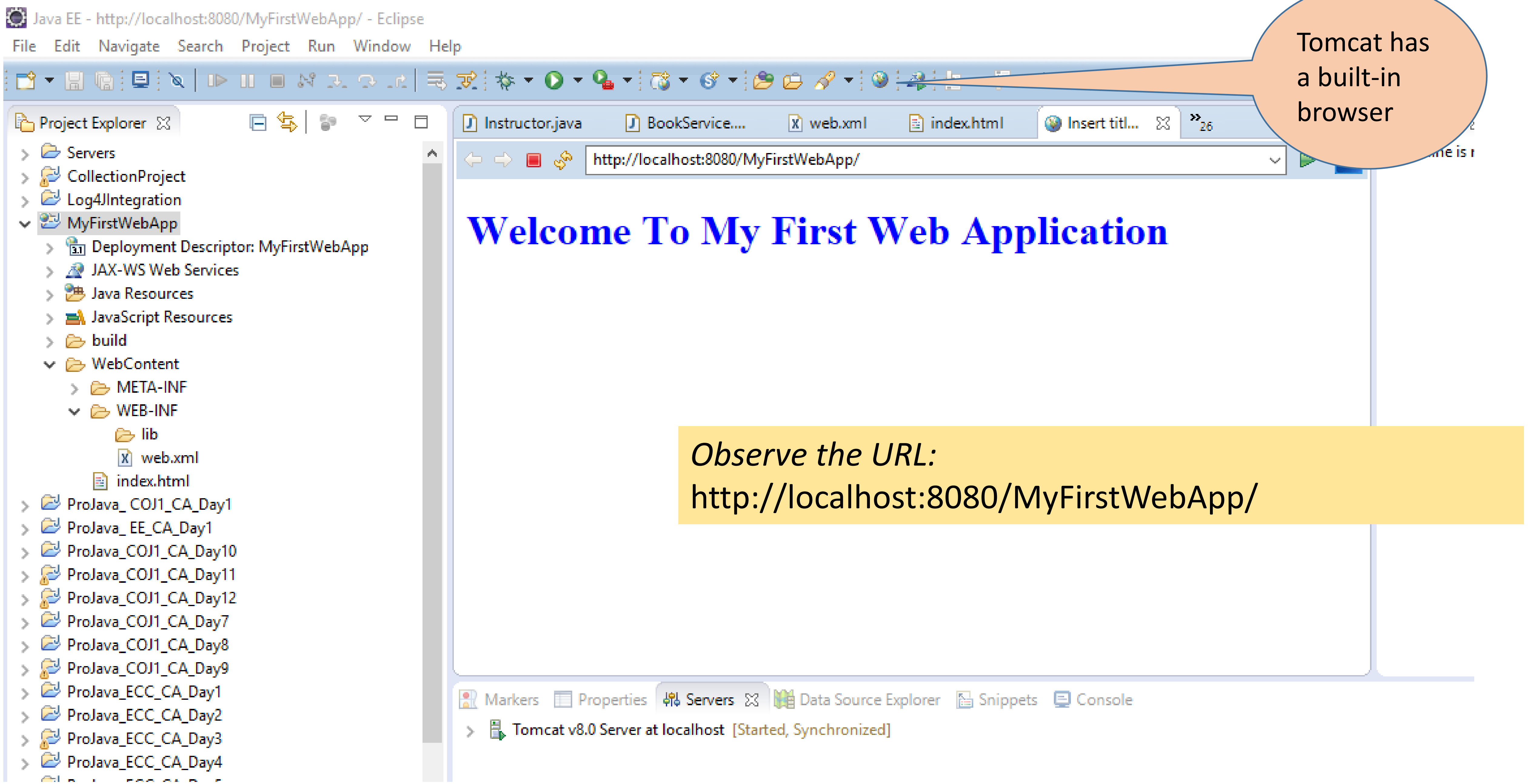
Right-click on **WebContent** folder, click on New->HTML file.
Enter filename as **index.html** and click on **Finish** button.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<h1><font color="blue">Welcome To My First Web Application</font></h1>
</body>
</html>
```

http://localhost:8080/MyFirstWebApp/

10. Right-click on the project, which is also called ***context root,*** *click on Run As -> Run on server.*
    Select Tomcat v8.0 Server at localhost as your web server, click on Finish button.

Creating Web Application



Welcome To My First Web Application

Tomcat v8.0 Server at localhost [Started, Synchronized]

Tomcat has a built-in browser

Observe the URL:
http://localhost:8080/MyFirstWebApp/

Thank You!