INTRODUCTION TO WEB APPLICATION ARCHITECTURE FUNDAMENTALS

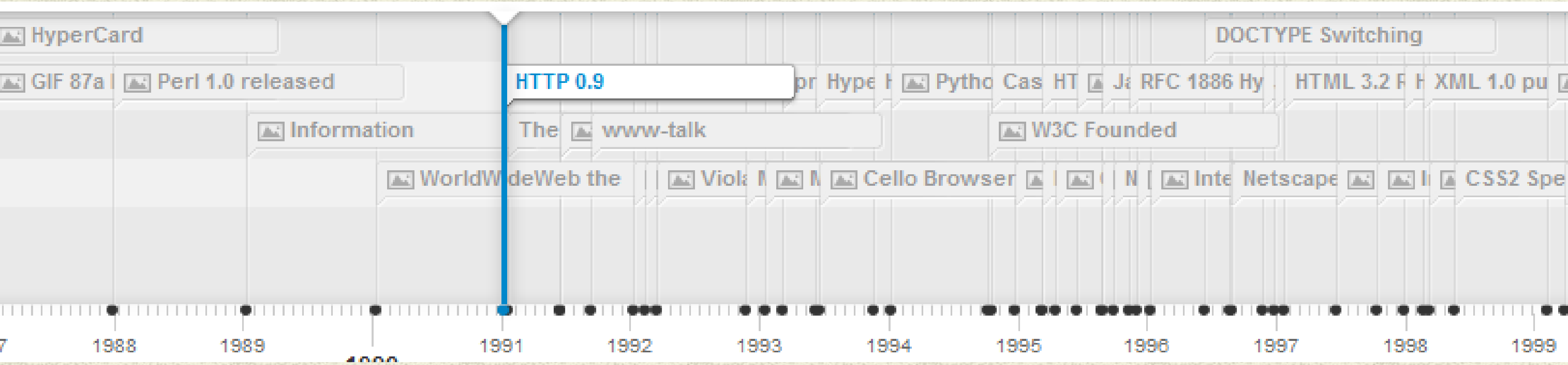
Home of All the Laws of Nature

Hypertext HTTP HTML

- Hypertext is structured text that uses logical links (hyperlinks) between nodes containing text.
  - You can go to any place on the Internet whenever you want by clicking on links — there is no set order to do things in.
- HTTP[Hyper Text Transfer Protocol] is the protocol to exchange or transfer hypertext.
- HTML[Hyper Text Markup Language]] is a *Language*, as it has code-words and syntax like any other language.
  - Markup is what **HTML tags** do to the text inside them. They mark it as a certain type of text (*italicised* text, for example).

History of the Web

1991milestones HTTP; HTML

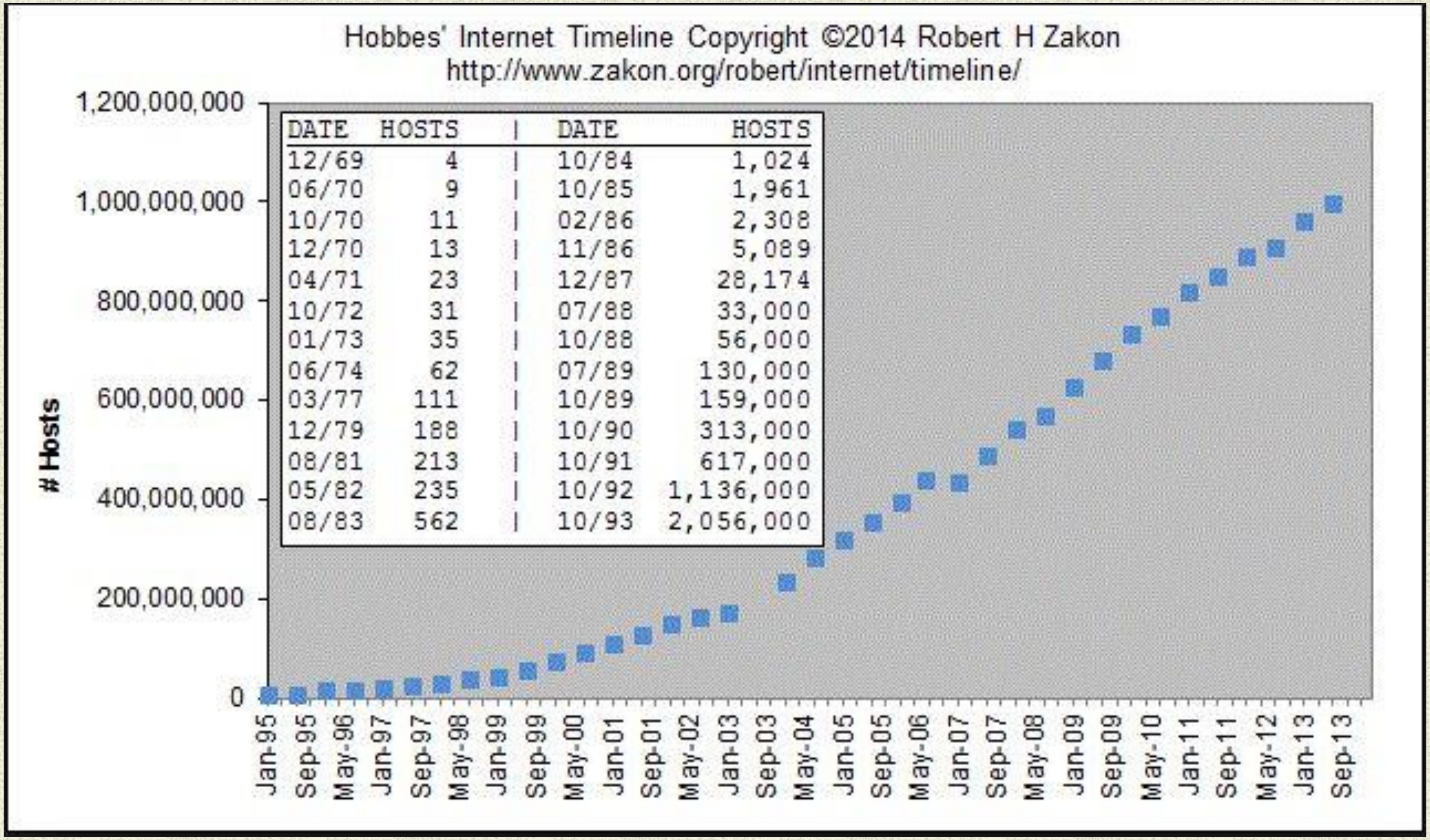


http://www.webdirections.org/history/

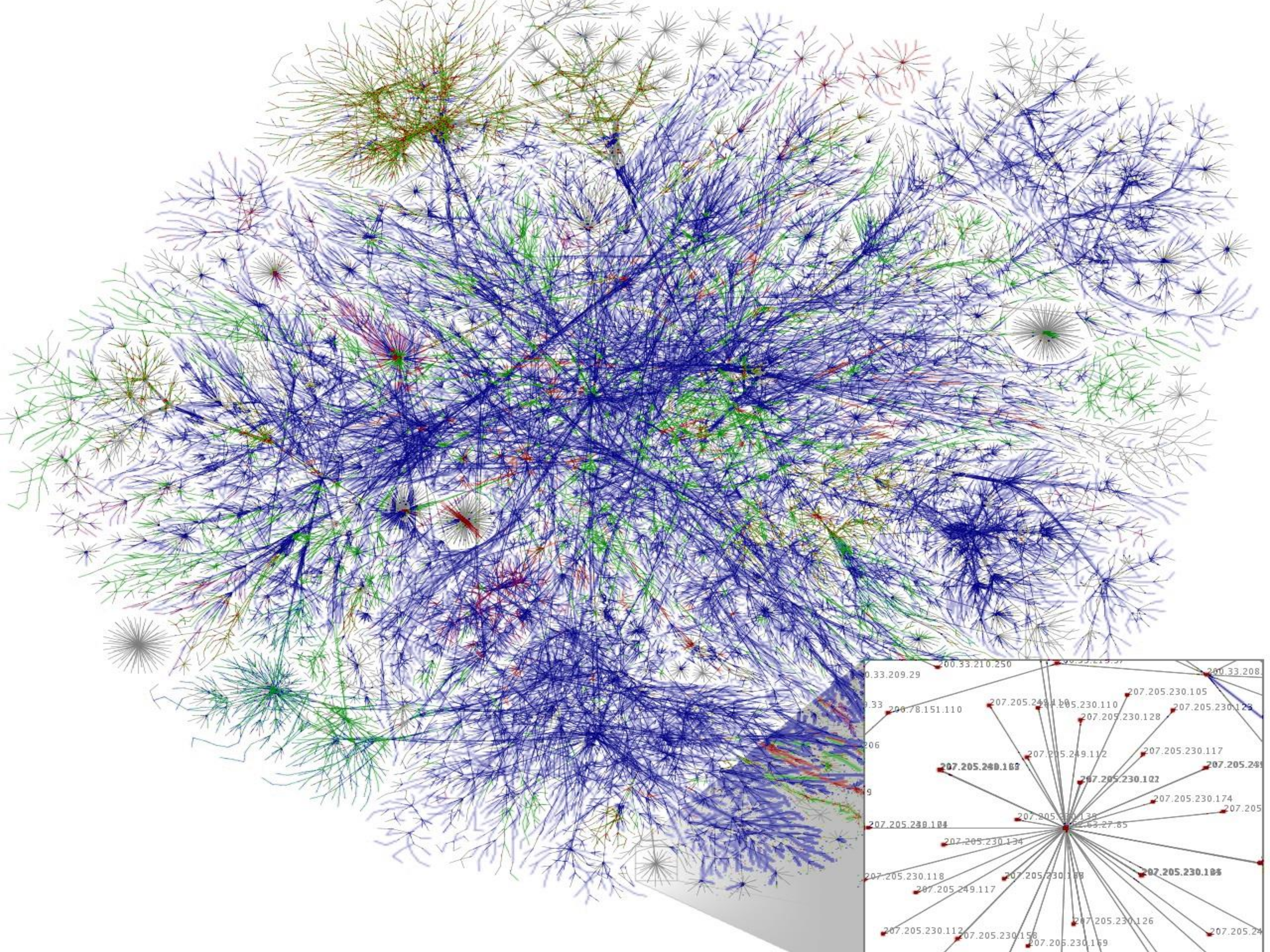
The oldest extant web page

The oldest web page continually served was last modified on Tuesday November 13 1990.

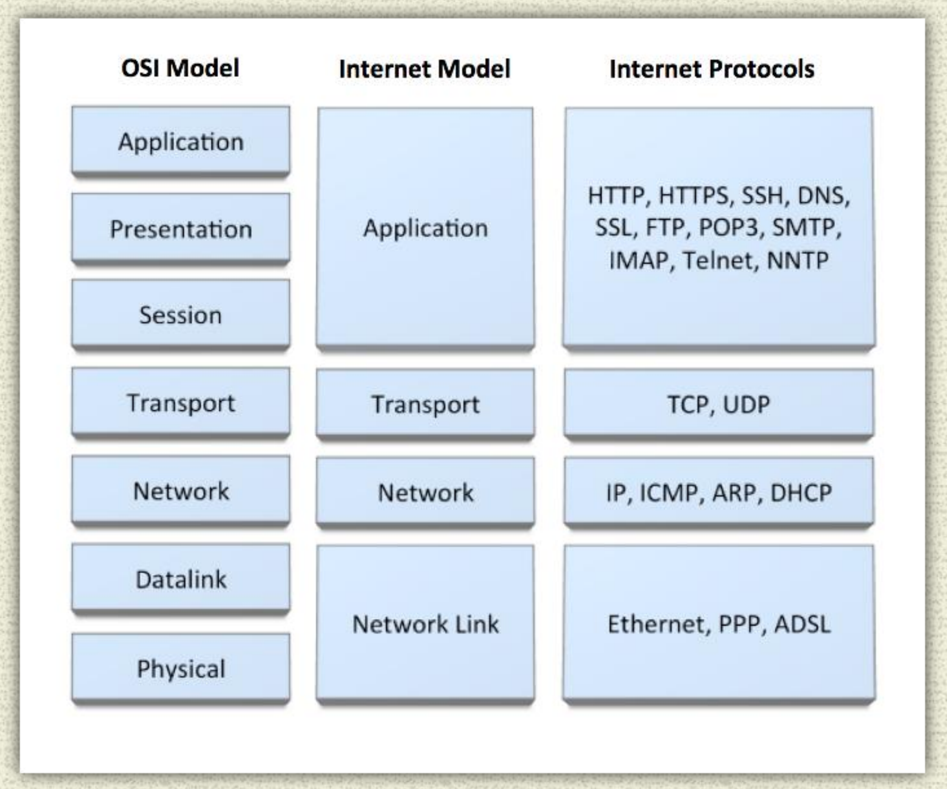
Growth of Internet Servers



DATE	HOSTS	DATE	HOSTS
12/69	4	10/84	1,024
06/70	9	10/85	1,961
10/70	11	02/86	2,308
12/70	13	11/86	5,089
04/71	23	12/87	28,174
10/72	31	07/88	33,000
01/73	35	10/88	56,000
06/74	62	07/89	130,000
03/77	111	10/89	159,000
12/79	188	10/90	313,000
08/81	213	10/91	617,000
05/82	235	10/92	1,136,000
08/83	562	10/93	2,056,000

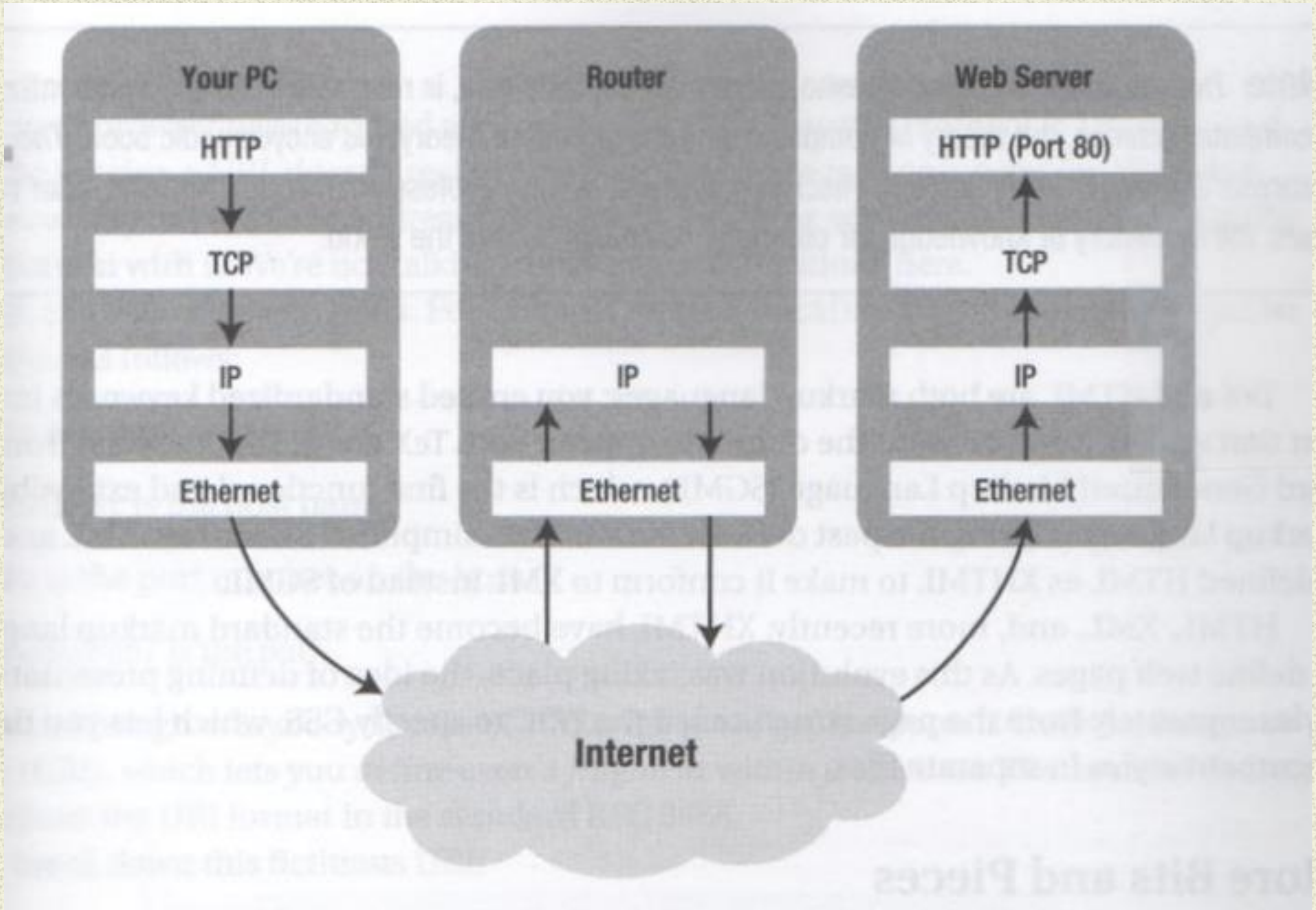

200.33.210.250
200.33.209.29
200.78.151.110
207.205.230.110
207.205.230.105
207.205.230.123
207.205.230.128
207.205.249.112
207.205.230.117
207.205.230.174
207.205.230.134
207.205.230.118
207.205.249.117
207.205.230.126
207.205.230.112
207.205.230.158
207.205.230.169

TCP/IP Suite of Protocols



OSI Model	Internet Model	Internet Protocols
Application	Application	HTTP, HTTPS, SSH, DNS, SSL, FTP, POP3, SMTP, IMAP, Telnet, NNTP
Presentation		
Session		
Transport	Transport	TCP, UDP
Network	Network	IP, ICMP, ARP, DHCP
Datalink	Network Link	Ethernet, PPP, ADSL
Physical		

HTTP Connections



1. *Following an HTTP request through the Internet protocol stacks*

HTTP: request/response protocol

A client sends request containing

- request method, URI, and protocol version
- followed by a MIME-like message containing request modifiers, client information, and possible body content

The server responds

- status line, including protocol version and a success or error code,
- followed by a MIME-like message containing server information, entity metainformation, and possible entity-body content.

- MIME (Multi-Purpose Internet Mail Extensions) allows for exchange of different kinds of data file on the Internet:

Examples

Request line (request message)

	URI		
Method	Context	File	Protocol Version
GET	/Lab01x	/Login	HTTP/1.1
POST	/Lab01x	/Login	HTTP/1.1

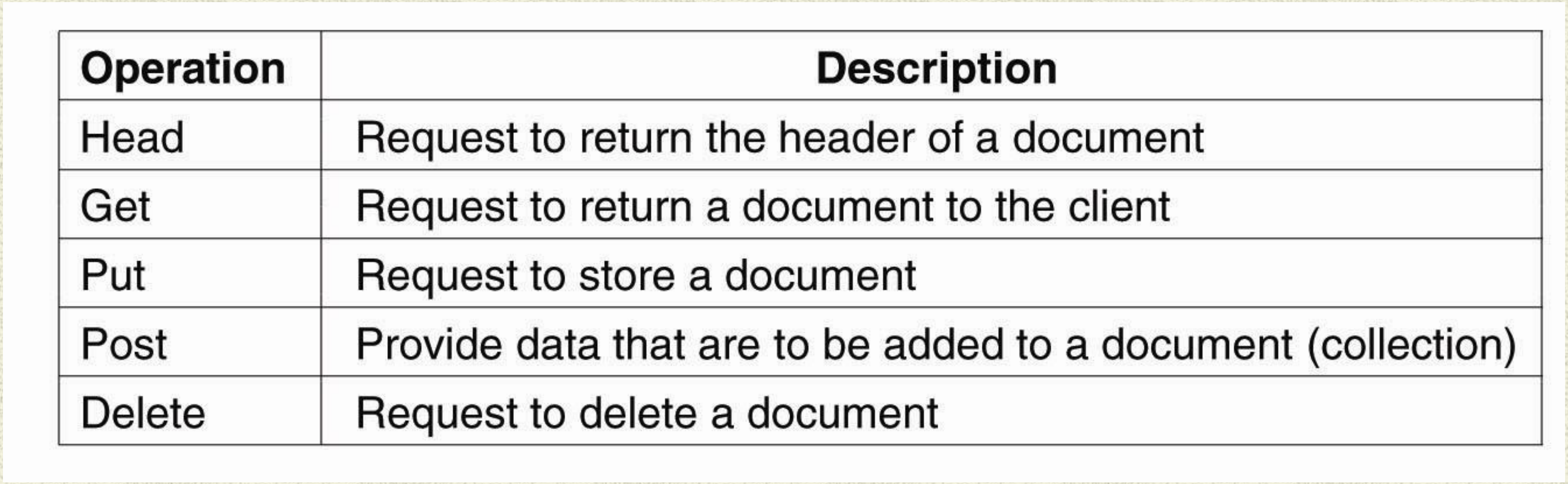
Status line (response message)

HTTP-version	status-code	reason-phrase
HTTP/1.1	200	OK
HTTP/1.1	404	/Lab01/Lab01z.html

Message headers (both)

field-name	field-value
Accept	*/*
Accept-Language	en-us
Content-Type	text/html
Content-Length	2222

HTTP Request Methods (operations)



Operation	Description
Head	Request to return the header of a document
Get	Request to return a document to the client
Put	Request to store a document
Post	Provide data that are to be added to a document (collection)
Delete	Request to delete a document

GET VS POST

The method name lets the server know what type of request it is, and how the rest of the message will be formatted.

GET

- It is simple to get something back from the server.
- Browser based limitation of ~2K characters in the GET request.
- Data sent is appended to the URL.
- By Specification- Idempotent. Multiple requests should not cause side affects or change the server state.

GET /path/file.html?firstname=Will&lastname=Pay HTTP/1.1

POST

- Powerful, request but purpose is to send (post) data to the server.
- Data sent in the message body.
- The page cannot be bookmarked.[RE: Form parameters]
- Non-idempotent.

POST /path/file.html HTTP/1.1

…

first**name=Will&lastname=Pay**

Status-Code

- 1xx: Informational - Request received, continuing process

- 2xx: Success - action successfully received, understood, and accepted

- 3xx: Redirection - Further action must be taken to complete request

- 4xx: Client Error - The request contains bad syntax or cannot be fulfilled

- 5xx: Server Error - The server failed to fulfill an apparently valid request

404 error Page Not Found

Minimal Web Server

- Define variables
  - int port =80;
- Create a ServerSocket
  - ServerSocket server = new ServerSocket(port);
- Wait for incoming requests
  - Socket socket = server.accept();
- When a request comes in:
  - Read the request from the socket.
  - Write the response to the socket.

Sample HTTP Exchange

- To retrieve the file at URL   http://www.somehost.com/path/file.html

1 Open a socket to the host **www.somehost.com**

2, Send something through the socket:

```
GET /path/file.html HTTP/1.0
From: someuser@jmarshall.com
User-Agent: HTTPTool/1.0
[blank line here]
```

3 The server responds via same socket:

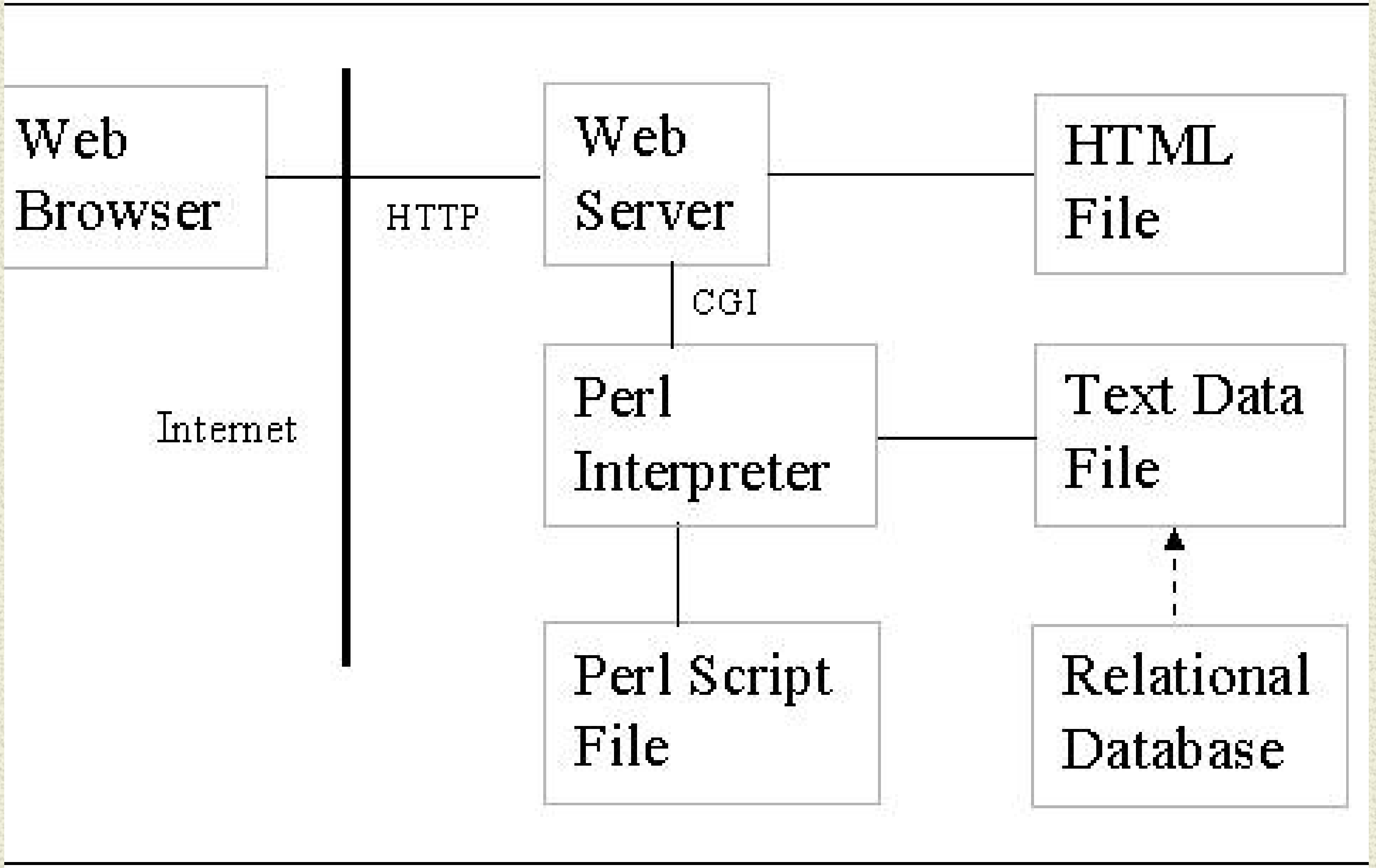
```
HTTP/1.0 200 OK
Date: Fri, 31 Dec 1999 23:59:59 GMT
Content-Type: text/html
Content-Length: 1354

<html> <body> <h1>Welcome to WAArF!</h1> blah blah blah . . . </body> </html>
```

Main point

HTTP is a simple and efficient protocol that has two types of messages, requests and responses.   Both contain informational headers and possibly body content. Requests contain a method type and resource identifier. Responses contain a success code. *There is precision and order in creation, creative intelligence is orderly and systematic.*

CGI Platform



Common Gateway Interface (CGI) is a standard method used to generate dynamic content on Web pages and Web applications.

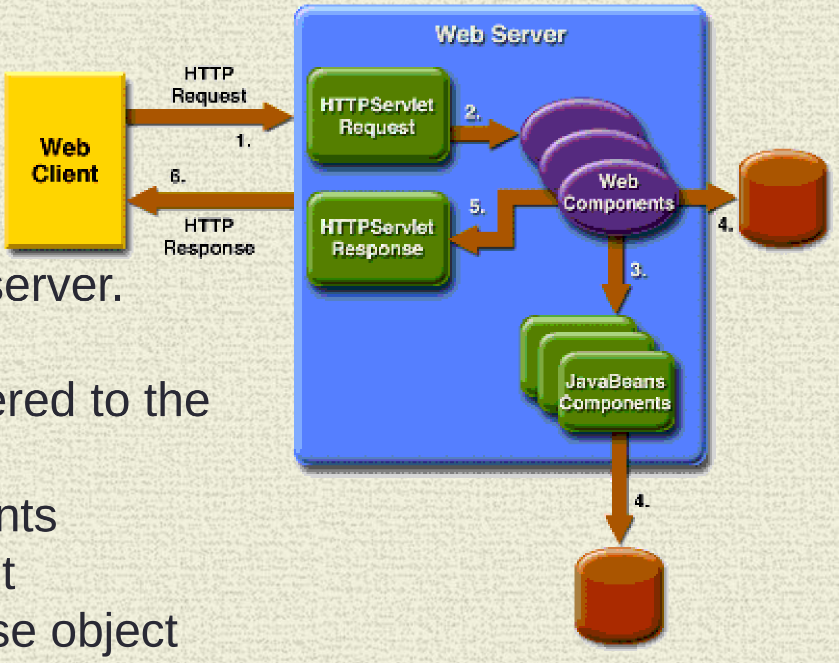
Servlets – An Efficient Technology

- A servlet is server-side java code that can handle http requests and return dynamic content

- Servlets are managed by a *servlet engine* or *container*. Each request that comes in results in the spawning of a new thread that runs a servlet (eliminating the cost of creating a new process every time).

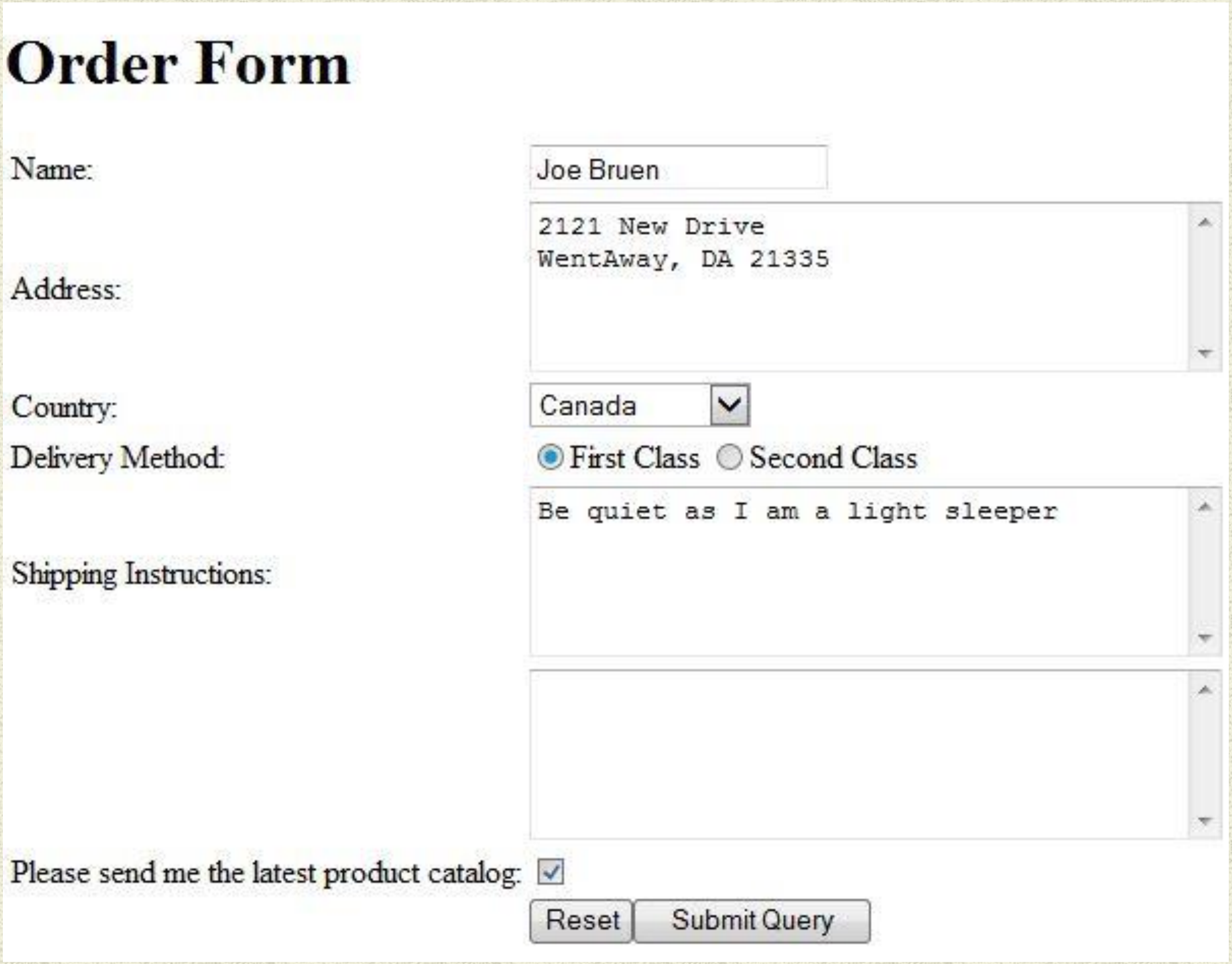
Web Application Servlet

- In the Java 2 platform, web components provide dynamic extension capabilities for a web server. A Java Servlet is a type of web component.



1. Client sends an HTTP request to the web server.
2. Web server converts the request into an HTTPServletRequest object which is delivered to the Servlet
3. Servlet interacts with JavaBeans components
4. or a database to generate dynamic content
5. Servlet generates an HTTPServletResponse object
6. Web server converts this object to an HTTP response and returns it to the client

A Simple Order Form



Order Form

Name:	Joe Bruen
Address:	2121 New Drive WentAway, DA 21335
Country:	Canada
Delivery Method:	[✓ First Class] Second Class
Shipping Instructions:	Be quiet as I am a light sleeper
Please send me the latest product catalog:	✓

Servlet Code doGet Example

```
response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        writer.println("<html>");
        writer.println("<head>");
        writer.println("<title>" + TITLE + "</title></head>");
        writer.println("<body><h1>" + TITLE + "</h1>");
        writer.println("<form method='post'>");
        writer.println("<table>");
        writer.println("<tr>");
        writer.println("<td>Name:</td>");
        writer.println("<td><input name='name'/></td>");
        writer.println("</tr>");
        writer.println("<tr>");
        writer.println("<td>Address:</td>");
        writer.println("<td><textarea name='address' "
                + "cols='40' rows='5'></textarea></td>");
        writer.println("</tr>");
```

Continued

```
•            writer.println("<tr>");
•           writer.println("<td>Country:</td>");
•           writer.println("<td><select name='country'>");
•           writer.println("<option>United States</option>");
•           writer.println("<option>Canada</option>");
•           writer.println("</select></td>");
•           writer.println("</tr>");
•           writer.println("<tr>");
•           writer.println("<td>Delivery Method:</td>");
•           writer.println("<td><input type='radio' " +
•                   "name='deliveryMethod'"
•                   + " value='First Class'/>First Class");
•           writer.println("<input type='radio' " +
•                   "name='deliveryMethod' "
•                   + "value='Second Class'/>Second Class</td>");
•           writer.println("</tr>");
•           writer.println("<tr>");
•           writer.println("<td>Shipping Instructions:</td>");
•   writer.println("<td><textarea name='instruction' "
•                   + "cols='40' rows='5'></textarea></td>");
•           writer.println("</tr>");
```

And More

```
writer.println("<tr>");
        writer.println("<td>&nbsp;</td>");
        writer.println("<td><textarea name='instruction' "
                + "cols='40' rows='5'></textarea></td>");
        writer.println("</tr>");
        writer.println("<tr>");
        writer.println("<td>Please send me the latest " +
                "product catalog:</td>");
        writer.println("<td><input type='checkbox' " +
                "name='catalogRequest'/></td>");
        writer.println("</tr>");
        writer.println("<tr>");
        writer.println("<td>&nbsp;</td>");
        writer.println("<td><input type='reset'/>" +
                "<input type='submit'/></td>");
        writer.println("</tr>");
        writer.println("</table>");
        writer.println("</form>");
        writer.println("</body>");
        writer.println("</html>");
```

Demo: HelloWorldServlet in STS/Eclipse

- •Servlet (and JSP) development is typically done in an IDE. We use STS/Eclipse (Luna) in this course. Luna comes with JEE tools supporting HTML, CSS, JavaScript, Servlet API, JSP, and JSF, and has an embedded browser.

- •We also need a "helper application" to handle the creation of dynamic content – called an *application server.* Our application server is Pivotal tc server v3.0 [Tomcat].

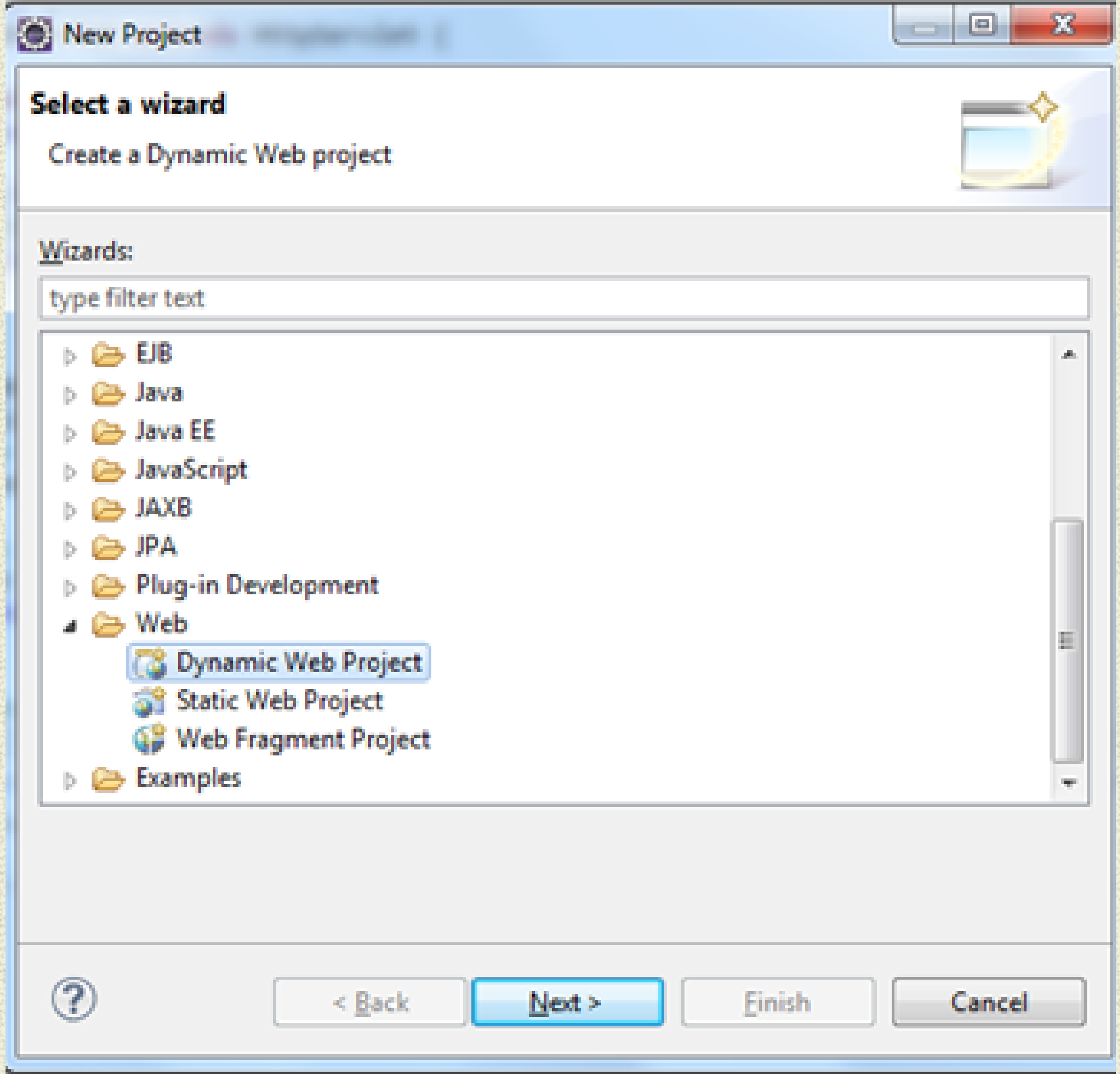
Demo: HelloWorldServlet in STS/Eclipse

Step 1: create a workspace in eclipse, right click new ---project ----…[other…]

Web ----choose Dynamic Web project

Then click:

Next



New Project

Select a wizard

Create a Dynamic Web project

Wizards:

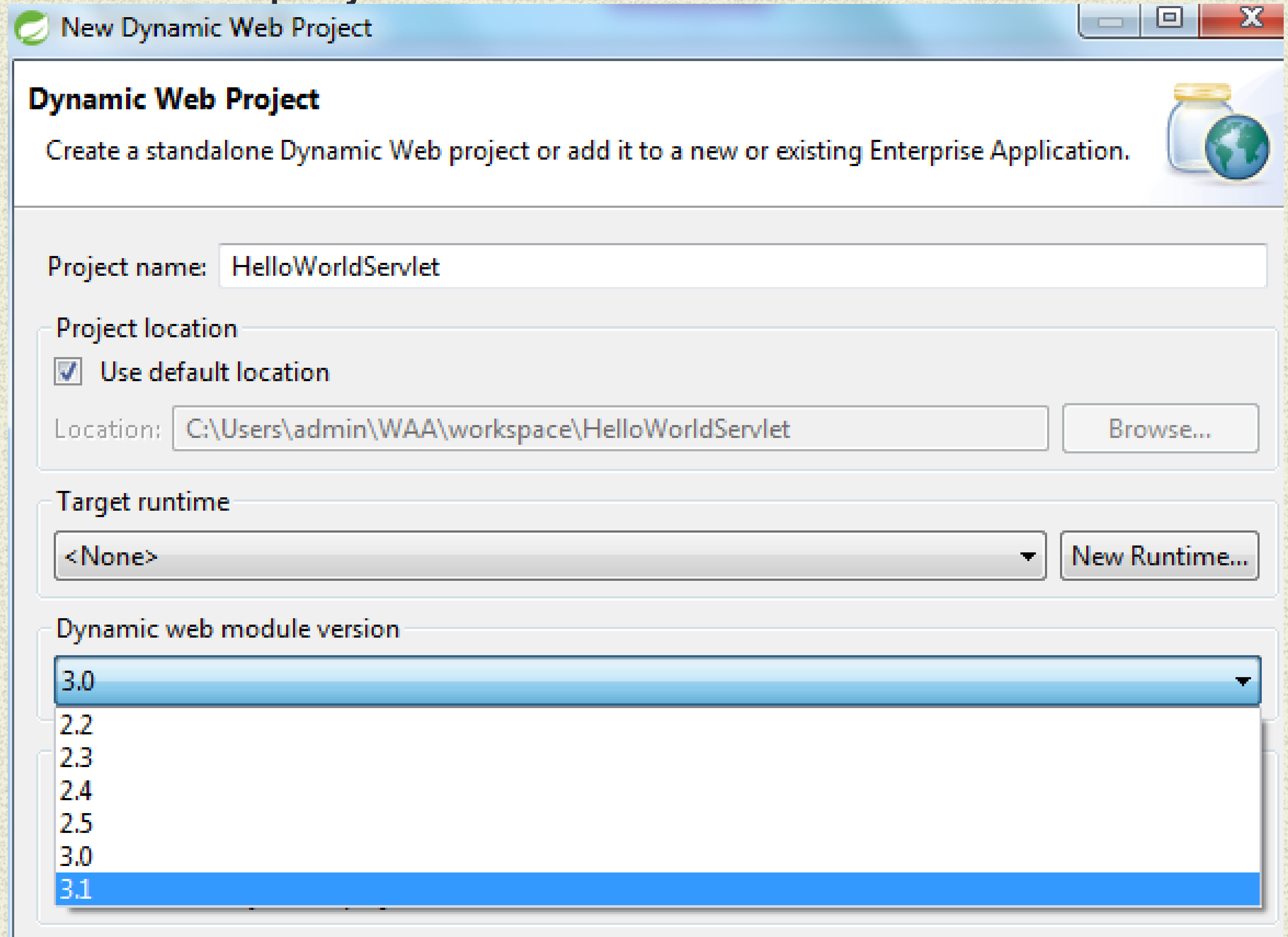
type filter text

- EJB
- Java
- Java EE
- JavaScript
- JAXB
- JPA
- Plug-in Development
- Web
  - Dynamic Web Project
  - Static Web Project
  - Web Fragment Project
- Examples

< Back | Next > | Finish | Cancel

Demo: HelloWorldServlet in STS/Eclipse

- Step 2: Name the project and click finish


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: HelloWorldServlet

Project location

[✓] Use default location

Location: C:\Users\admin\WAA\workspace\HelloWorldServlet

Browse...

Target runtime

<None>

New Runtime...

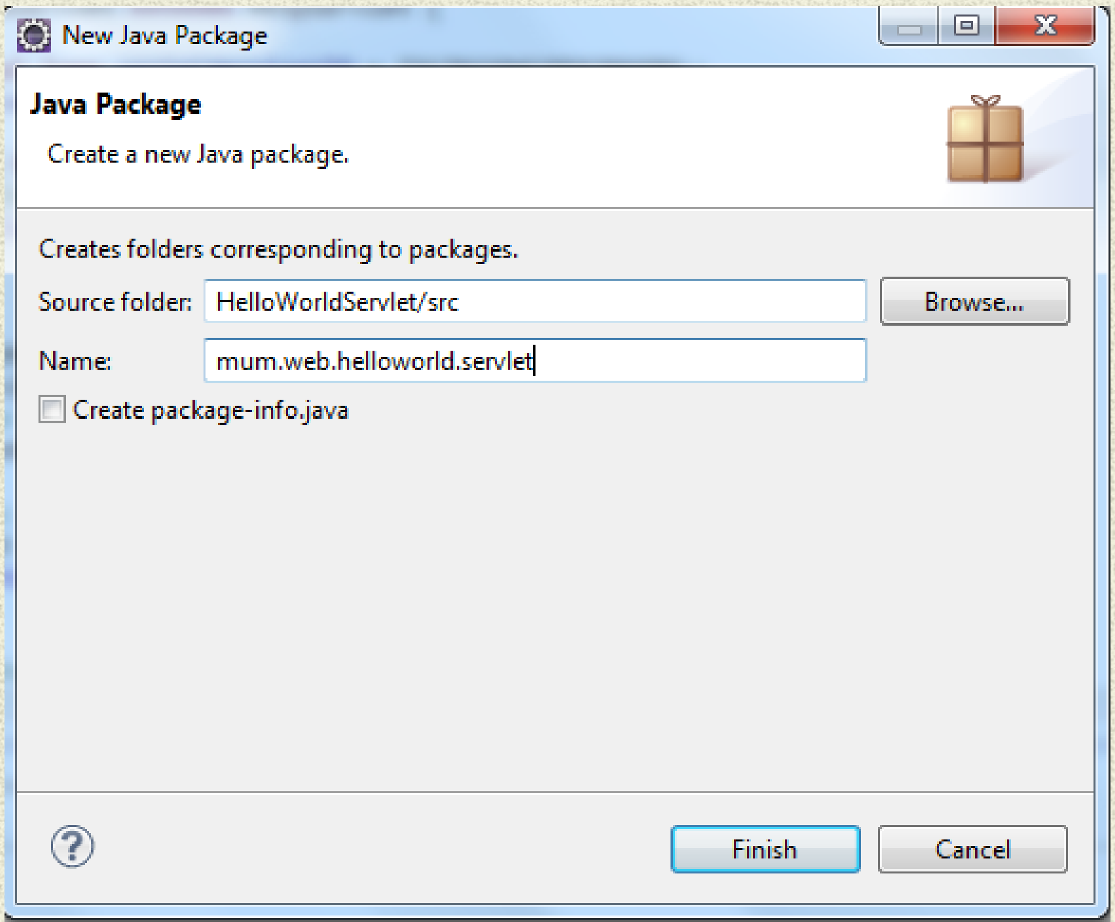
Dynamic web module version

3.0

2.2
2.3
2.4
2.5
3.0
3.1

Demo: HelloWorldServlet

- Step 3: Create a new package under src folder
  Click: Finish



New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

Source folder: HelloWorldServlet/src

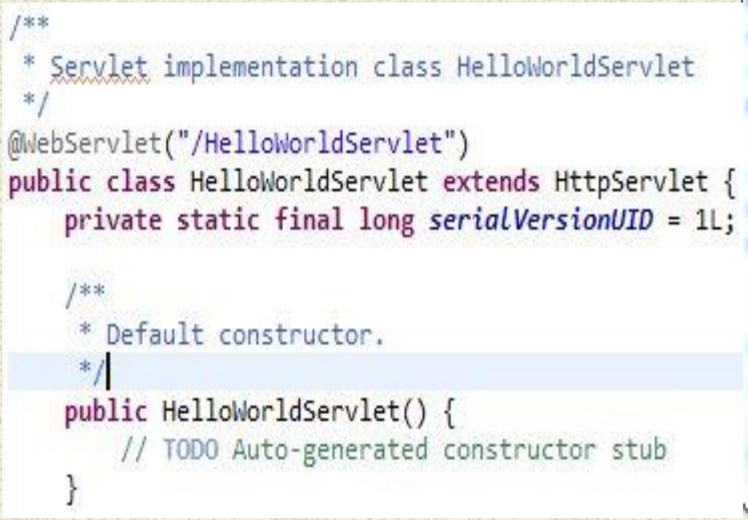
Name: mum.web.helloworld.servlet

☐ Create package-info.java

Demo: HelloWorldServlet

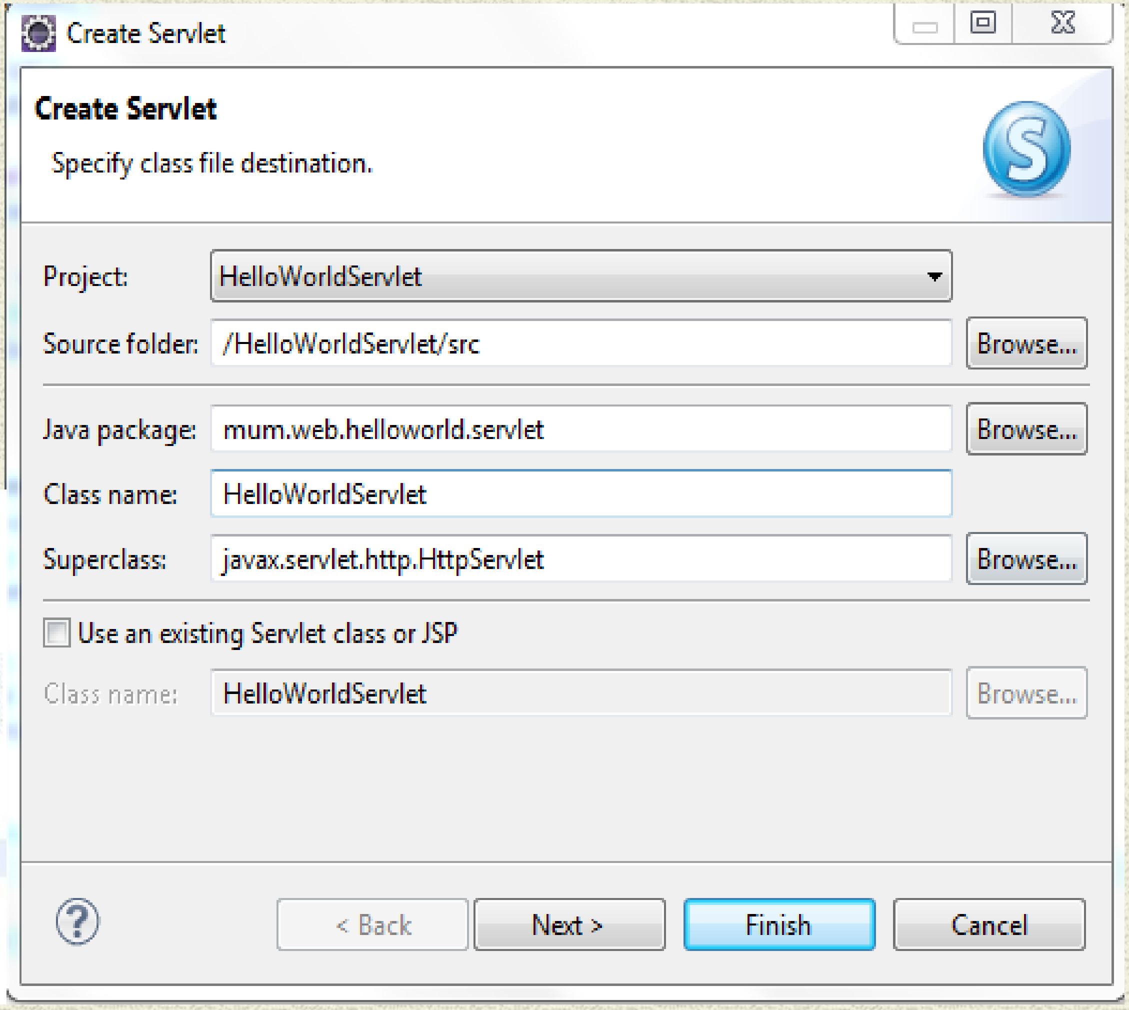
- **Step 4:** Right click on package new servlet : Name the class then Click
- On Finish

Eclipse by default inserts the annotation @WebServlet on the servlet class declaration. If you use a DD, you would delete it.



```java
/**
 * Servlet implementation class HelloWorldServlet
 */
@WebServlet("/HelloWorldServlet")
public class HelloWorldServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Default constructor.
     */
    public HelloWorldServlet() {
        // TODO Auto-generated constructor stub
    }
```



Create Servlet

Specify class file destination.

Project: HelloWorldServlet

Source folder: /HelloWorldServlet/src

Java package: mum.web.helloworld.servlet

Class name: HelloWorldServlet

Superclass: javax.servlet.http.HttpServlet

[ ] Use an existing Servlet class or JSP

Class name: HelloWorldServlet

Demo: HelloWorldServlet

- Step 5: do some thing with the servlet – in this example, we print out a 'Hello World' message.

```
//add message field to the servlet class
    private String message;

    // initialize the servlet
    public void init() throws ServletException {
         message = "Hello World";
     }
    //override the doGet method
     public void doGet(HttpServletRequest request,
HttpServletResponse response)
                  throws ServletException, IOException {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          out.println("<h1>" + message + "</h1>");
      }
```

Demo: HelloWorldServlet

- Step 6: Run the servlet using Pivotal tc Server. Right click HelloWorldServlet.java

  Run As

  Run on Server

  Choose Pivotal tc Server

  Finish

Demo: HelloWorldServlet

- Step 7: You can see the program running in Eclipse built-in browser. Or you can access it by the url http://localhost:8080/HelloWorldServlet/HelloWorldServlet from your own browser
- Note the composition of the url
  - http - protocol
  - localhost:8080 – local port number
  - First HelloWorldServlet – project name which becomes the *context root*
  - Second HelloWorldServlet – servlet name, you can change how the servlet is referenced using the annotation @WebServlet(name) or by creating a value in servlet-mapping in web.xml (more on this later)

Project Structure

- WebContent
  - Location of all Web resources, including HTML, JSP, graphic files, and so on. If the files are not placed in this directory (or in a subdirectory structure under this directory), the files will not be available when the application is executed on a server. The Web content folder represents the contents of the WAR file that will be deployed to the server.
- WEB-INF
  - Contains the supporting Web resources for a Web application, including the web.xml file and the classes and lib directories.
- /classes
  - This directory is for servlets, utility classes, and the Java compiler output directory. The classes in this directory are used by the application class loader to load the classes.
- /lib

  Library files that your Web application references. Any classes in .jar files placed in this directory will be available for your Web application

Main Point

Servlets are the basis of dynamic web applications. Servlets process information from a request object and generate new information in a response object. *For every action in nature there is always a reaction.*

Anatomy of a webapp

- The sample web app hello from today’s Lab contains the following files:

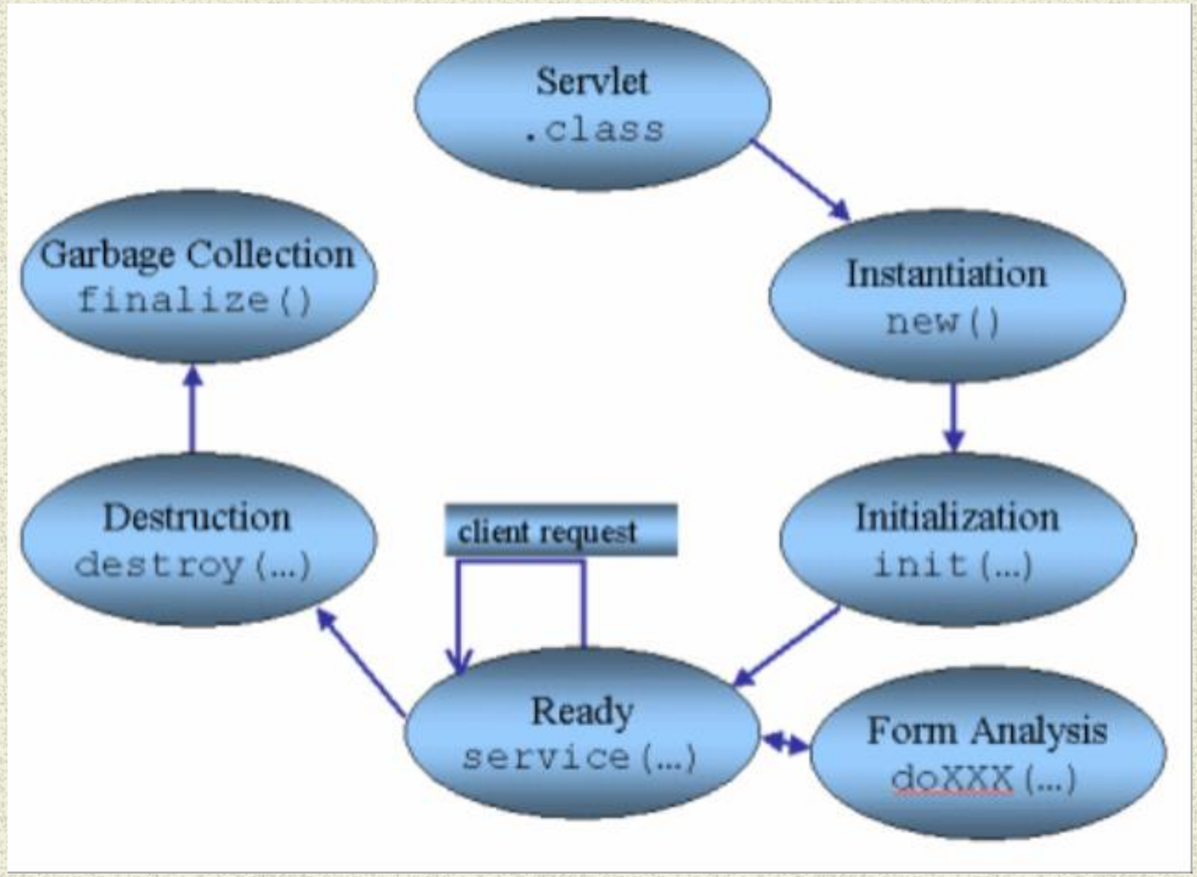
- mum.Hello.java – the servlet
- index.html – the “welcome” web page (invokes the servlet when the user submits)
- web.xml – configuration file – the Deployment Descriptor (DD)

- To understand the role of each we need to understand the underlying architecture further

The Container

- Servers that support servlets have as a helper app a *servlet container.*
- When a request comes to the web server, if the server sees the request is for a servlet, it passes the request data to the servlet container.
- The servlet container locates the servlet, creates request and response objects and passes them to the servlet, and returns to the web server the response stream that the servlet produces.
- The web server sends the response back to the client browser to be rendered.

Servlet Lifecycle



Three Names of a Servlet [web.xml]

- <servlet>
  - <servlet-name>hello</servlet-name>
  - <servlet-class>mum.Hello</servlet-class>
- </servlet>
- <servlet-mapping>
  - <servlet-name>hello</servlet-name>
  - <url-pattern>/hello</url-pattern>
- </servlet-mapping>
- **“servlet-name” is the internal name of the servlet**
- **“serlvet-class” is the Java name of the class**
- **“url-pattern” is the way the servlet is specified in the**
- **browser http:localhost:8080/HelloWorld/hello** *and the*
- **HTML form <form action=*"hello" method="get"*>**

URL Patterns

- Every character in a pattern must match the corresponding character in the URL path **EXACTLY**
- Two exceptions:
  - At the end of a pattern, /* matches any sequence of characters from that point forward.
  - The pattern *.*extension* matches any “file name” ending with *extension*.

Default Servlet Mapping:

<url-pattern>/</url-pattern> matches a request if no other pattern matches.

<url-pattern>/*</url-pattern> overrides all other servlets. Whatever request you fire, it will end up in that servlet.

URL Patterns [cont.]

Mapping rules Precedence order:

- Map exact URL

  <servlet-name>northroster</servlet-name>

  <url-pattern>/north/teamRoster.do</url-pattern>/

- Map wildcard paths

  <servlet-name>northclub</servlet-name>

  <url-pattern>/north/*</url-pattern>/

- Map extensions

  <servlet-name>northsouth</servlet-name>

  <url-pattern>*.do</url-pattern>/

- CATCHALL: Map to the default servlet

Welcome List [web.xml]

- <welcome-file-list>
-     <welcome-file>index.html</welcome-file>
- </welcome-file-list>

ServletConfig & ServletContext

- ServletConfig is a configuration object PER servlet to pass initialization information to a servlet during servlet init().
- ServletContext holds parameters that are initialization information for the entire application(i.e, every servlet in the application). ServletContext is also used during the application for application state management.

```
<web-app>
        <context-param>
                <param-name>applicationName</param-name>
                <param-value>Order Form</param-value>
        </context-param>
        <servlet>
                <servlet-name>order</servlet-name>
                <servletclass>mum.edu.cs.Order</servlet-class>
                <init-param>
                        <param-name>servletName</param-name>
                        <param-value>MVC2</param-value>
                </init-param>
        </servlet>
         <listener>
                 <listener-class>mum.edu.listener.OrderContextListener </listener-class>
        </listener>
```

OrderFormStartup Demo

- **Override Servlet init() – to process the Servlet init-param:**

```
public class Order extends HttpServlet {
// Store servlet init-param here
String servletName;
@Override
public void init( ) throws ServletException {
    servletName = getServletConfig().getInitParameter("servletName");
}
```

Implement Listener to process Application Context context-param:

```
 public class OrderContextListener implements ServletContextListener{
    public void contextInitialized(ServletContextEvent event){
   ServletContext servletContext = event.getServletContext();
          String applicationName =
                          servletContext.getInitParameter("applicationName");
  servletContext.setAttribute("applicationName", applicationName);
    }
```

Main Point

- Web containers manage the life cycle of servlets. *The unified field manages the entire universe.*