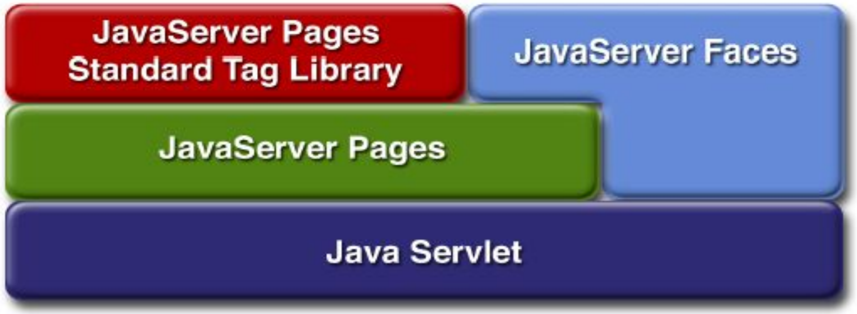
HNDIT 4232: Enterprise Architecture

Web-based application development and state management

Web Applications

- web application (sometimes shortened to web app) is a collection of.
  - Servlets
  - Java server pages (JSPS)
  - HTML documents
  - Images
  - Templates, and other web resources



that are set up in such a way as to be portable deployed across any servlet-enabled web server.

- i.e. All the files under
- server_root/webapps/ROOT belong to a single web application (the root one)

Web Modules

- In the J2EE architecture, web components and static web content files such as images are called web resources.
- A web module is specification. In addition to web components and web resources, a web module can contain other files: the smallest deployable and usable unit of web resources.
- A J2EE web module corresponds to a web application as defined in the Java Servlet

Web Modules

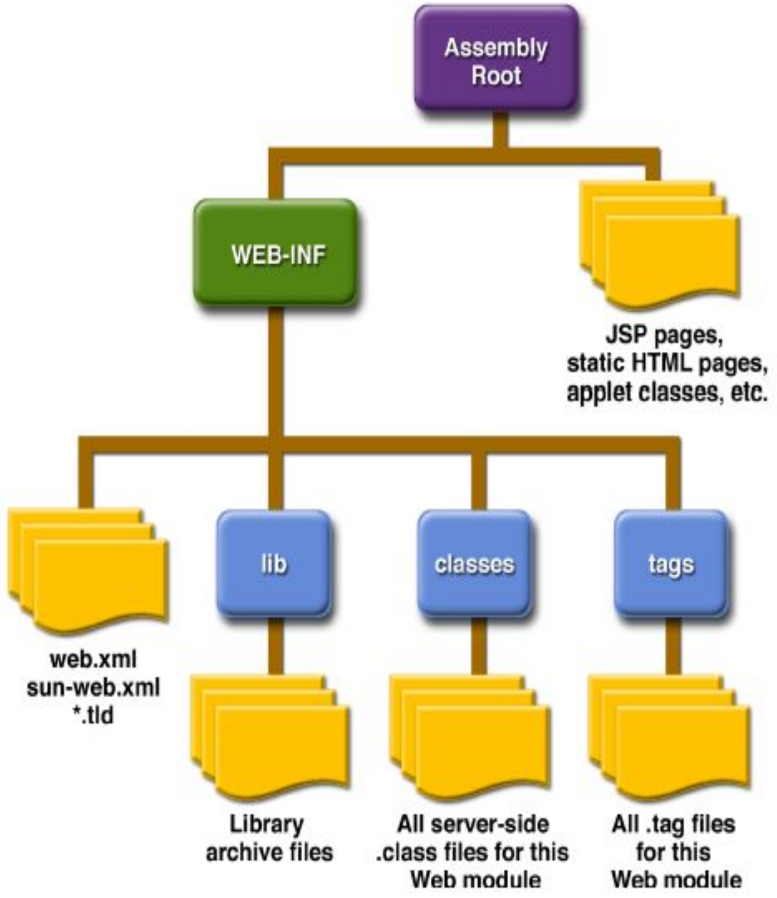

Assembly Root
WEB-INF
JSP pages, static HTML pages, applet classes, etc.
web.xml sun-web.xml *.tld
lib
classes
tags
Library archive files
All server-side .class files for this Web module
All .tag files for this Web module

Figure 3–5 Web Module Structure

- A web module has a specific structure. The top-level directory of a web module is the document root of the application. The document root is where JSP pages, client-side classes and archives, and static web resources, such as images, are stored.
- The document root contains a subdirectory named /WEB-INF/, which contains the following files and directories:
- • web.xml: The web application deployment descriptor
- • Tag library descriptor files
- • classes: A directory that contains *server-side classes*: servlets, utility classes, and JavaBeans components
- • tags: A directory that contains tag files, which are implementations of tag libraries
- • lib: A directory that contains JAR archives of libraries called by server side classes

The WEB-INF Directory

- The WEB-INF directory is special. The files inside WEB-INF are not served directly to the client, instead, they contain java classes and configuration information for the web app
- The WEB-INF/classes directory contains the class files for this web app's servlets and support classes.
- WEB-INF/lib contains classes stored in JAR files.

The Development Descriptor

- The web.xml file in the WEB –INF directory is known as a deployment descriptor.
- This file contains configuration information about the web app in which it resides. It’s an XML file with a standardized DTD.
- Everything between <web-app>and </web-app>, provide information to the server about this web application.
- Tags in a web.xml are order dependent. For example, the <servlet-class> to ensure everything work.
-

Web App Deployment

- Packaged web apps are archived into one file (.war file) and deployed on servlet containers.
- Since the file structure is pre-agreed among the vendors, the effort required to port web app running on the servlet container of one vender to another is minimal.
- However, if you use vendor specific extensions in the web app development, porting would not be so easy.

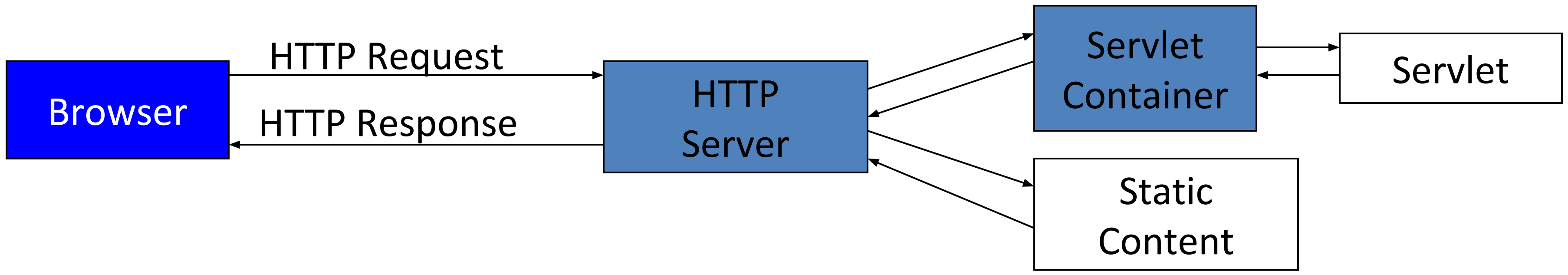
Web Archives (wars)

- To simplify deployment, web app files can be bundled in to a single archive file and developed to another server merely by placing the archive file in to a specific directory.
- These archive files files have the extension, war. Which stands for web application archive.
- WAR files are actually JAR files (creted using the jar utility) saved with an alternate extension.

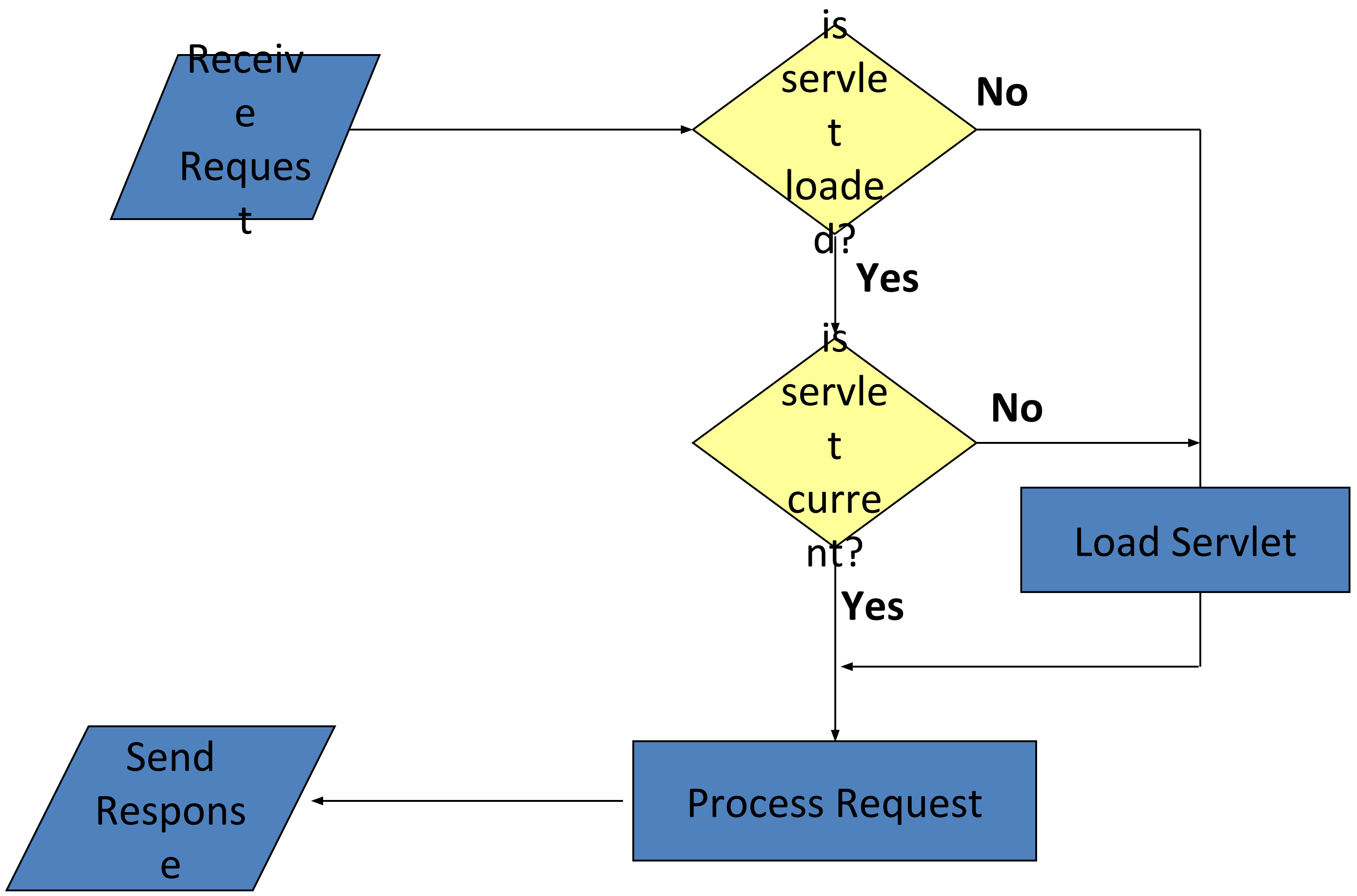
Servlet - Definition

- **A *servlet* is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model**.
- They can be loaded dynamically into and run by a special web server
- This servlet-aware web server, is known as servlet container
- Servlets interact with clients via a request-response model based on HTTP
- Therefore, a servlet container must support HTTP as the protocol for client requests and server responses
- The *javax.servlet* and *javax.servlet.http* packages provide interfaces and classes for writing servlets.
- All servlets must implement the Servlet interface, which defines life-cycle methods.

Servlet Container Architecture



How Servlet Works



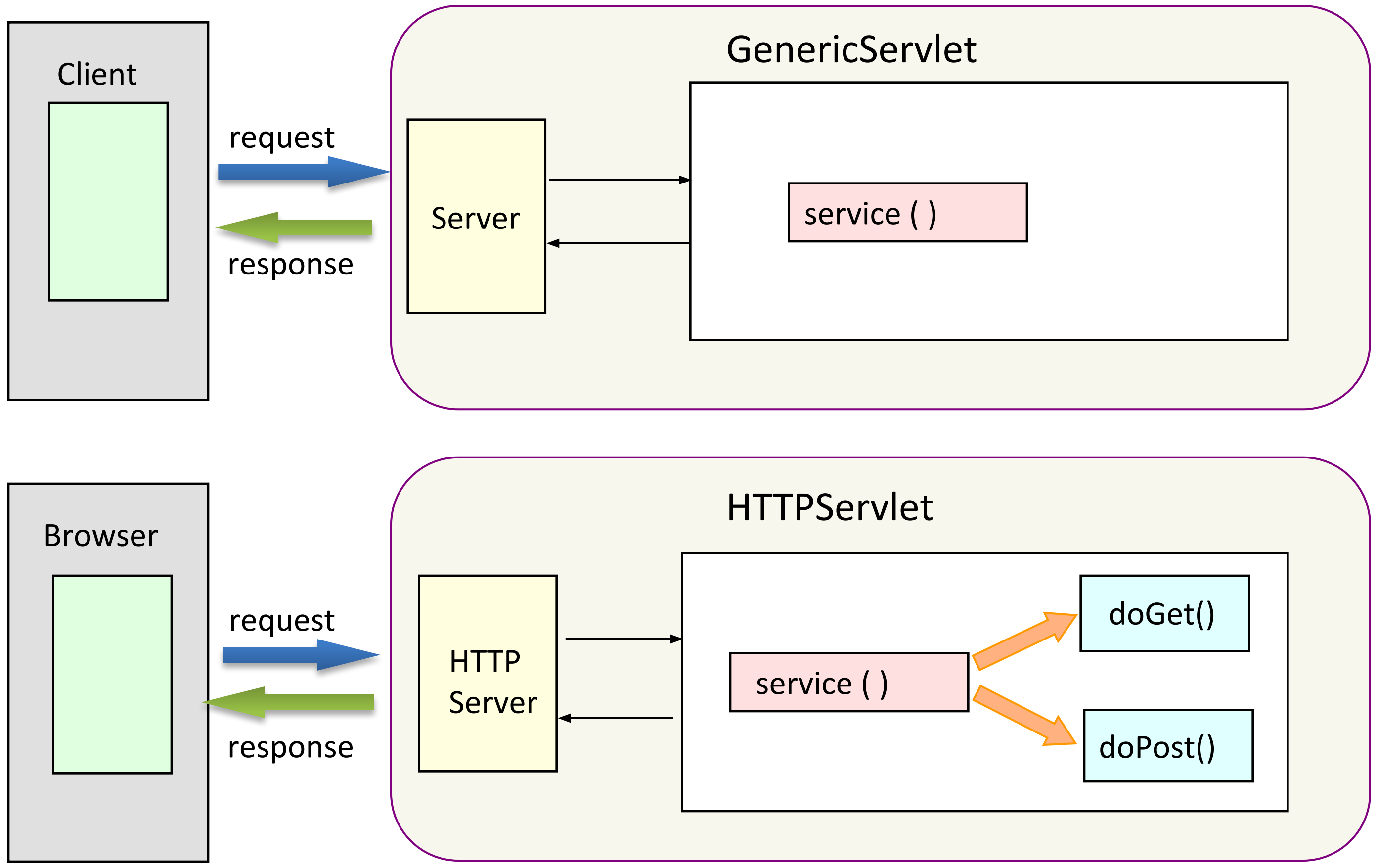
Servlet API

- Every servlet must implement javax.servlet.Servlet interface
- Most servlets implement the interface by extending one of these classes
  - javax.servlet.GenericServlet
  - javax.servlet.http.HttpServlet
- The API provides support in four categories :
  - Servlet lifecycle management.
  - Access to the servlet context.
  - Utility classes.
  - HTTP – specific support classes.

Servlet API

- ***javax.servlet. GenericServlet***
  - A generic Servlet should override its service() method to handle requests as appropriate for the Servlet.
  - The service() method accepts two parametes :
    - A request object and
    - A response object
- ***javax.servlet. http.HttpServlet***
  - An HTTP Servlet does not override the service() method. Instead , it overrides doGet() to handle GET requests and doPost() to handle POST request.
  - The service method of HttpServlet handles the setup and dispatching to all the doXxx() methods, which is why it usually should not be overridden.
  - An HTTP servlet can also override doPut() , doDelete() requests.However , HTTP servlet usually don’t override doTrace() or doOptions() methods as the default implementations are almost sufficient.

Generic Servlet Vs. HTTP Servlet

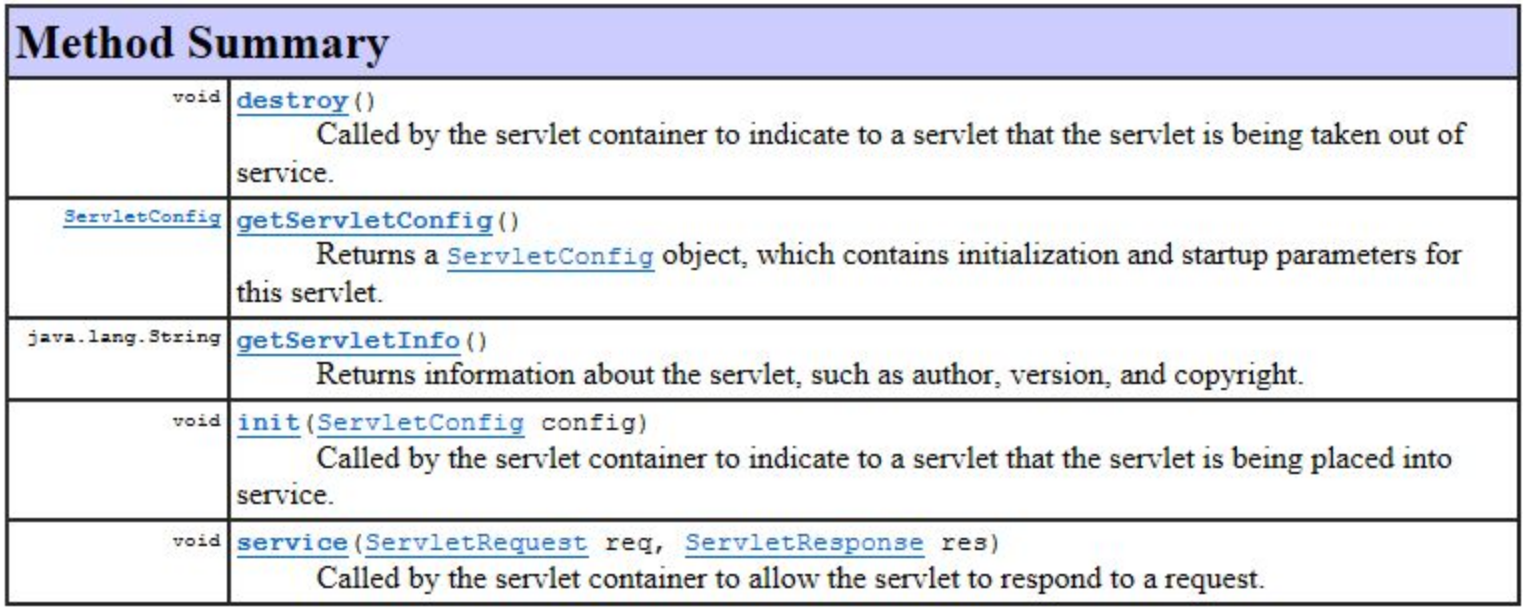


Interface javax.servlet.Servlet

- The Servlet interface defines methods
  - to initialize a servlet
  - to receive and respond to client requests
  - to destroy a servlet and its resources
  - to destroy a servlet and its resources
  - to return basic information about itself, such as its author, version and copyright

- Developers need to directly implement this interface only if their servlets cannot (or choose not to) inherit from GenericServlet or HttpServlet.

Generic Servlet - Methods



Method Summary	
void	destroy() Called by the servlet container to indicate to a servlet that the servlet is being taken out of service.
ServletConfig	getServletConfig() Returns a ServletConfig object, which contains initialization and startup parameters for this servlet.
java.lang.String	getServletInfo() Returns information about the servlet, such as author, version, and copyright.
void	init(ServletConfig config) Called by the servlet container to indicate to a servlet that the servlet is being placed into service.
void	service(ServletRequest req, ServletResponse res) Called by the servlet container to allow the servlet to respond to a request.

Servlet Functions

- Act as a middle-tier process
- Act as proxies to client process such as Applets.
- Protocol Support- Any protocol that follows a request/response computing model can be implemented by a servlet. This could include : HTTP , SMTP, POP, FTP , etc.

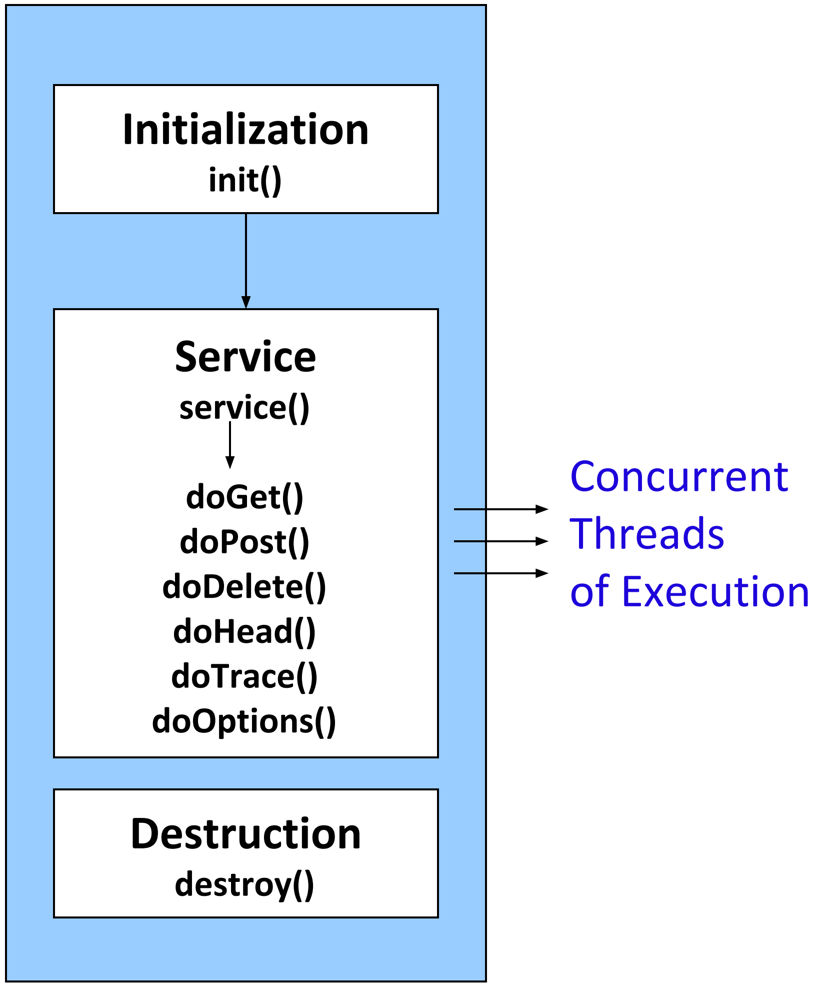
Characteristics of Servlets

- Servlets are efficient
- Servlets are portable
- Servlets are robust
- Servlets are persistence

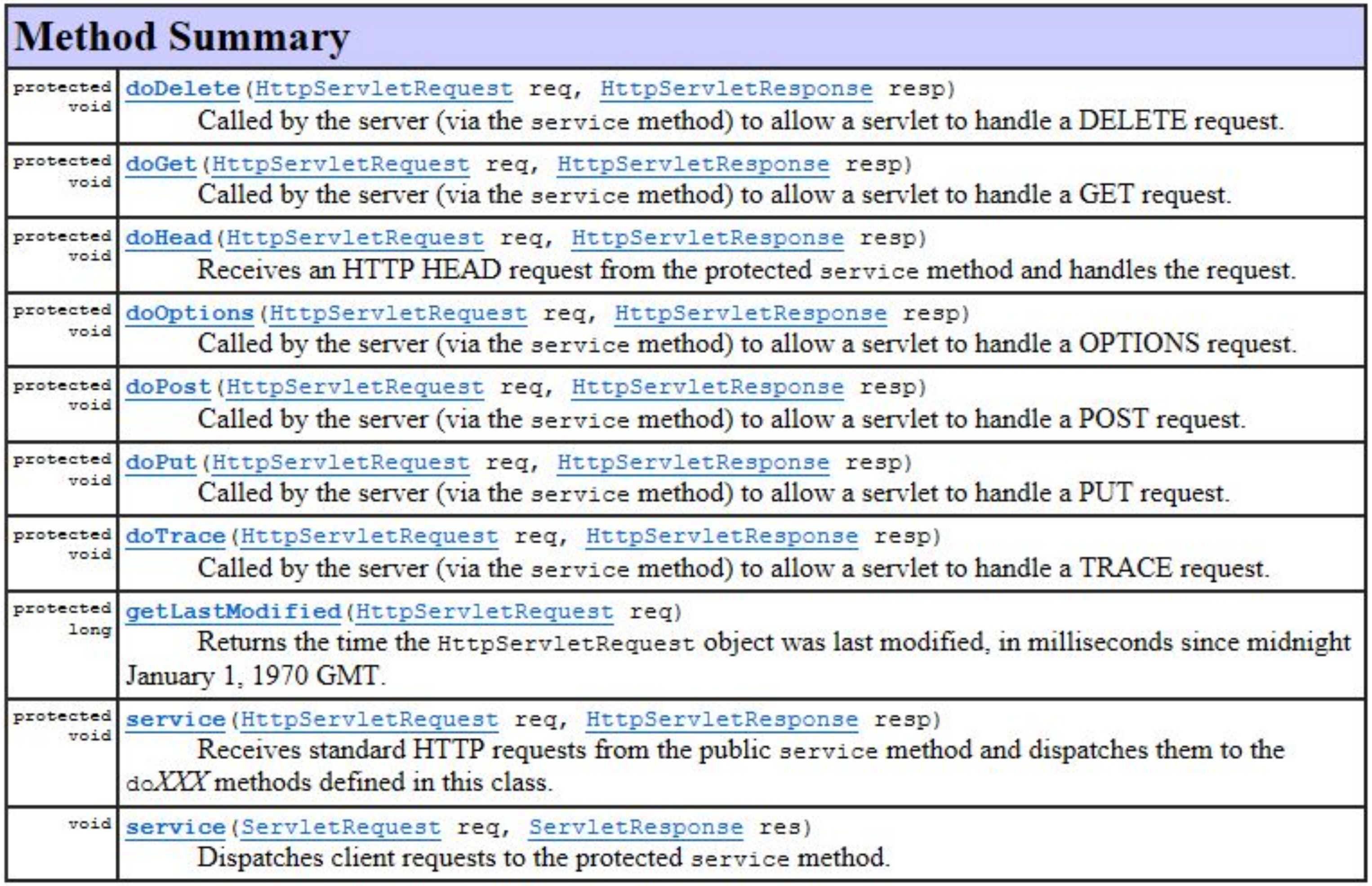
Servlet Container

- Servlets run inside a special container called the servlet container as part of the same process as the web server itself.
- The servlet container is responsible for initializing , invoking, and destroying each servlet instance.
- Three primary tasks of the servlet container :
  - Creating the Request Object.
  - Creating the Response Object.
  - Invoking the service method of the servlet, passing the request and response objects.

Servlet Life Cycle



HTTPServlet - Methods



Method Summary	
protected void	`doDelete(HttpServletRequest req, HttpServletResponse resp)` Called by the server (via the `service` method) to allow a servlet to handle a DELETE request.
protected void	`doGet(HttpServletRequest req, HttpServletResponse resp)` Called by the server (via the `service` method) to allow a servlet to handle a GET request.
protected void	`doHead(HttpServletRequest req, HttpServletResponse resp)` Receives an HTTP HEAD request from the protected `service` method and handles the request.
protected void	`doOptions(HttpServletRequest req, HttpServletResponse resp)` Called by the server (via the `service` method) to allow a servlet to handle a OPTIONS request.
protected void	`doPost(HttpServletRequest req, HttpServletResponse resp)` Called by the server (via the `service` method) to allow a servlet to handle a POST request.
protected void	`doPut(HttpServletRequest req, HttpServletResponse resp)` Called by the server (via the `service` method) to allow a servlet to handle a PUT request.
protected void	`doTrace(HttpServletRequest req, HttpServletResponse resp)` Called by the server (via the `service` method) to allow a servlet to handle a TRACE request.
protected long	`getLastModified(HttpServletRequest req)` Returns the time the `HttpServletRequest` object was last modified, in milliseconds since midnight January 1, 1970 GMT.
protected void	`service(HttpServletRequest req, HttpServletResponse resp)` Receives standard HTTP requests from the public `service` method and dispatches them to the `do`*XXX* methods defined in this class.
void	`service(ServletRequest req, ServletResponse res)` Dispatches client requests to the protected `service` method.

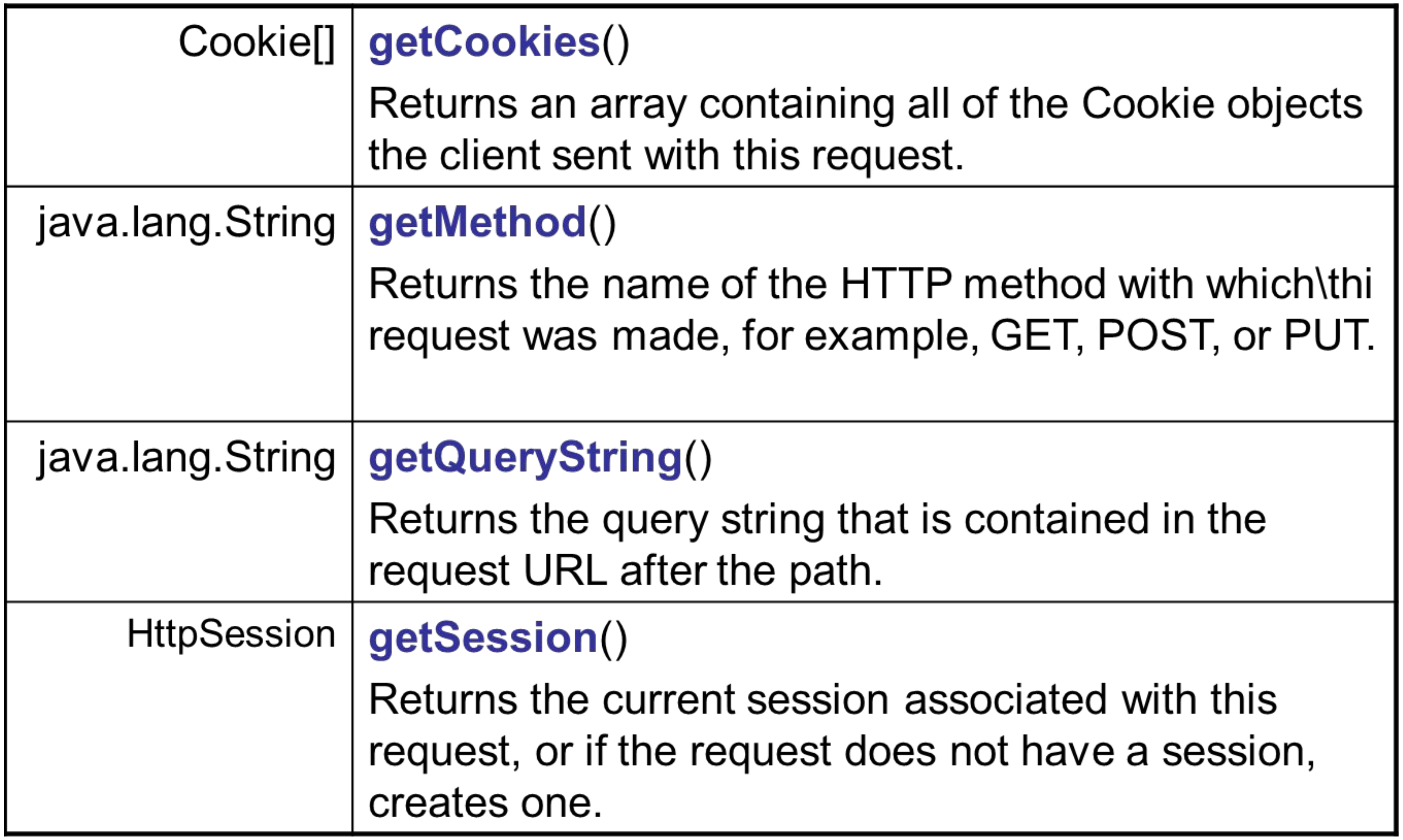
Servlet Request Objects

- Provides client request information to a servlet
- The servlet container creates a servlet request object and passes it as an argument to the servlet's service method
- The ServletRequest interface define methods to retrieve data sent as client request:
  - Parameter name and values
  - Attributes
  - Input stream
- HTTPServletRequest extends the ServletRequest interface to provide request information for HTTP servlets

HTTPServletRequest - Methods

Enumeration	**getParameterNames()** an Enumeration of String objects, each String containing the name of a request parameter; or an empty Enumeration if the request has no parameters
java.lang.String[]	**getParameterValues** (java.lang.String name) Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
java.lang.String	**getParameter** (java.lang.String name) Returns the value of a request parameter as a String, or null if the parameter does not exist.

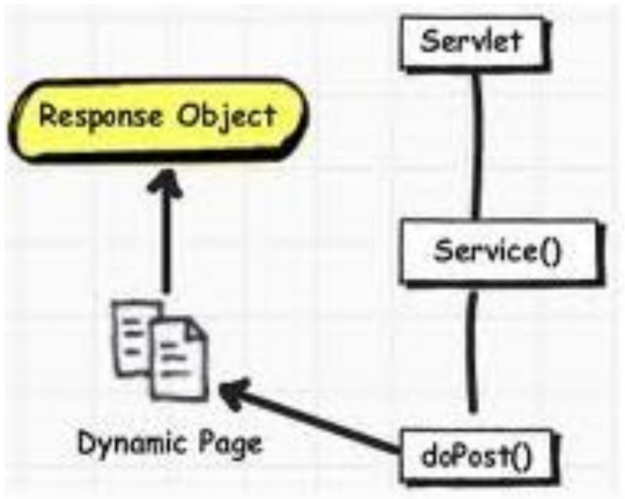
HTTPServletRequest - Methods



Cookie[]	**getCookies**() Returns an array containing all of the Cookie objects the client sent with this request.
java.lang.String	**getMethod**() Returns the name of the HTTP method with which\thi request was made, for example, GET, POST, or PUT.
java.lang.String	**getQueryString**() Returns the query string that is contained in the request URL after the path.
HttpSession	**getSession**() Returns the current session associated with this request, or if the request does not have a session, creates one.

Servlet Response Objects

- Defines an object to assist a servlet in sending a response to the client
- The servlet container creates a ServletResponse object and passes it as an argument to the servlet's service method



HttpServletResponse - Methods

java.io.PrintWriter	**getWriter**() Returns a PrintWriter object that can send character text to the client
void	**setContentType** (java.lang.String type) Sets the content type of the response being sent to the client. The content type may include the type of character encoding used, for example, text/html; charset=ISO-8859-4
int	**getBufferSize**() Returns the actual buffer size used for the response

Write the Servlet Code

- Servlets implement the javax.servlet.Servlet interface
- Because most servlets extend web servers that use the HTTP protocol to interact with clients, the most common way to develop servlets is by specializing the javax.servlet.http.HttpServlet class
- The HttpServlet class implements the Servlet interface by extending the GenericServlet base class, and provides a framework for handling the HTTP protocol.
- Its service() method supports standard HTTP requests by dispatching each request to a method designed to handle it.

Step by step servlet Designing

- Import packages which contain classes that are use by servlet
- Declare the servlet class which extends javax.servlet.http.HTTPServlet.
- Over ride HTTPServlet doGet (), doPost()
- In doGet() or doPost() use set contentType () of the HTTPServlet response object to set the contain type of the response that we are going to send. All response headers must be set before a PrintWriter is requested to write body data to the response.
- Now obtain a PrintWriter object from ServletResponse

  PrintWriter out=res.getWriter ();
- Use the PrintWriter to write a text or text/html eg: out.println("Hello");
- Close the PrintWriter after finishing writing to it. The web server closes the PrintWriter automatically when the service called return. An explicit call is done when you want to do some post processing after the response to the client has been fully written. out.close ();
- If you wish to give some information like the servlet name, version, author, & copyright notice etc, over ride the getServletInfo() which is suppose to return information about the servlet.

Servlet Example

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

public class MyServlet extends HttpServlet{

  protected void doGet(HttpServletRequest req,HttpServletResponse res){
    res.setContentType( "text/html");
    PrintWriter out = res.getWriter();
    out.println( "<HTML><HEAD><TITLE> Hello You!” +
                "</Title></HEAD>” +
                "<Body> Hello You!!!</BODY></HTML>" );
    out.close();
  }

  public String getServletInfo(){
     return "MyServlet version 1";
  }
}
```

An Example Servlet

Lines 1 to 3 import some packages which contain many classes which are used by the Servlet (almost every Servlet needs classes from these packages)

```
1:   import java.io.*;
2:   import javax.servlet.*;
3:   import javax.servlet.http.*;
```

The Servlet class is declared in line 4. Our Servlet extends javax.servlet.http.HttpServlet, the standard base class for HTTP Servlets

```
4:   public class MyServlet
    extends HttpServlet{
```

```
 5:     protected void doGet(HttpServletRequest req,
                   HttpServletResponse res)
 ..  ...
12:     }
```

In lines 5 through 12 HttpServlet's doGet method is getting overridden

An Example Servlet

In line 6 we use a method of the HttpServletResponse object to set the content type of the response that we are going to send.

All response headers must be set *before* a PrintWriter or ServletOutputStream is requested to write body data to the response.

```
6: res.setContentType("text/html");
```

In line 7 we request a PrintWriter object to write text to the response message.

```
7: PrintWriter out =
    res.getWriter();
```

In lines 8, 9 and 10 we use the PrintWriter to write the text of type text/html (as specified through the content type).

```
 8: out.println( "<HTML><HEAD><TITLE> Hello You!” +
                "</Title></HEAD>” +
10:             "<Body> HelloYou!!!</BODY></HTML>" );
```

An Example Servlet

The PrintWriter gets closed in line 15 when we are finished writing to it.

```
11: out.close();
```

In lines 18 through 21 we override the getServletInfo() method which is supposed to return information about the Servlet, e.g. the Servlet name, version, author and copyright notice.

This is not required for the function of the HelloClientServlet but can provide valuable information to the user of a Servlet who sees the returned text in the administration tool of the Web Server.

```
13:   public String getServletInfo(){
14:       return "MyServlet version 1";
15:   }
```

Compile the Servlet

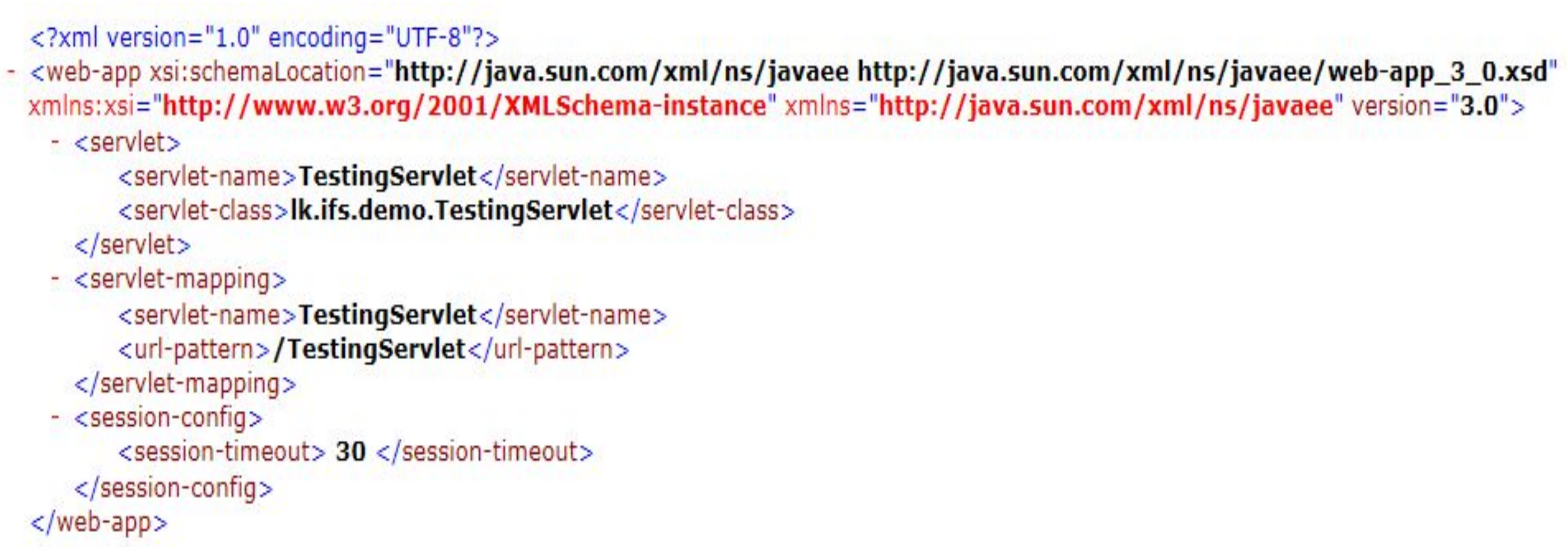
- Compile the Servlet class
- The resulting MyServlet.class file should go under myApp/WEB-INF/classes

Deploy the Servlet

- In the Servlet container each application is represented by a servlet context
- each servlet context is identified by a unique path prefix called context path
  - For example our application is identified by /myApp which is a directory under webapps
- The remaining path is used in the selected context to find the specific Servlet to run, following the rules specified in the deployment descriptor

Deployment Descriptor

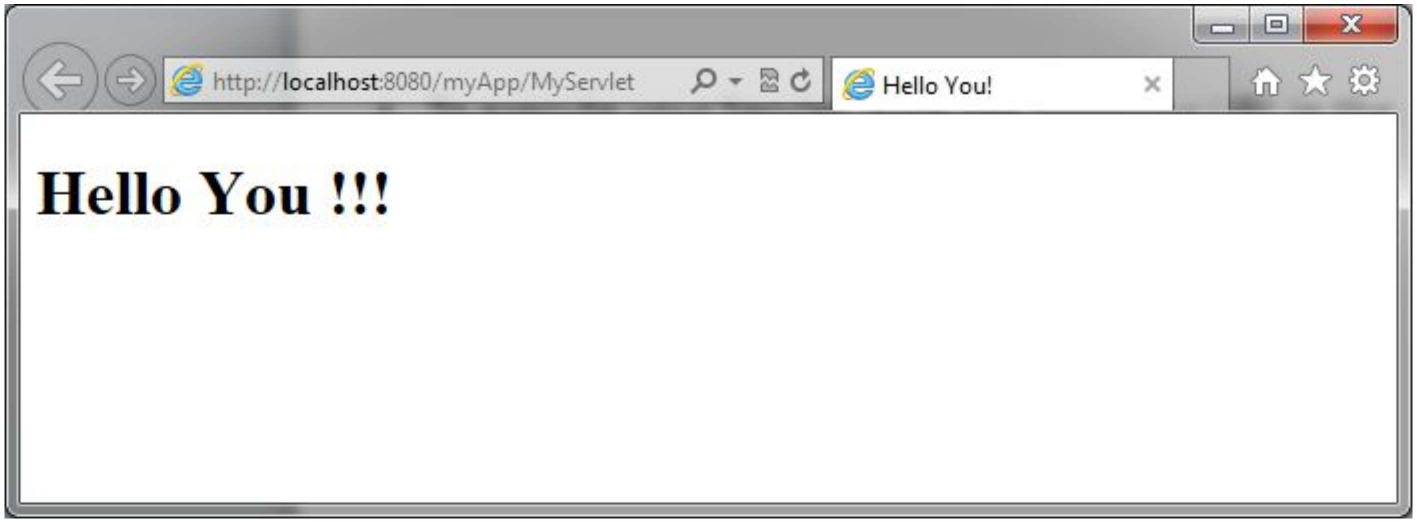
- The deployment descriptor is a XML file called web.xml that resides in the WEB-INF directory within an application.



```
  <?xml version="1.0" encoding="UTF-8"?>
- <web-app xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee" version="3.0">
  - <servlet>
        <servlet-name>TestingServlet</servlet-name>
        <servlet-class>lk.ifs.demo.TestingServlet</servlet-class>
    </servlet>
  - <servlet-mapping>
        <servlet-name>TestingServlet</servlet-name>
        <url-pattern>/TestingServlet</url-pattern>
    </servlet-mapping>
  - <session-config>
        <session-timeout> 30 </session-timeout>
    </session-config>
  </web-app>
```

Run the Servlet

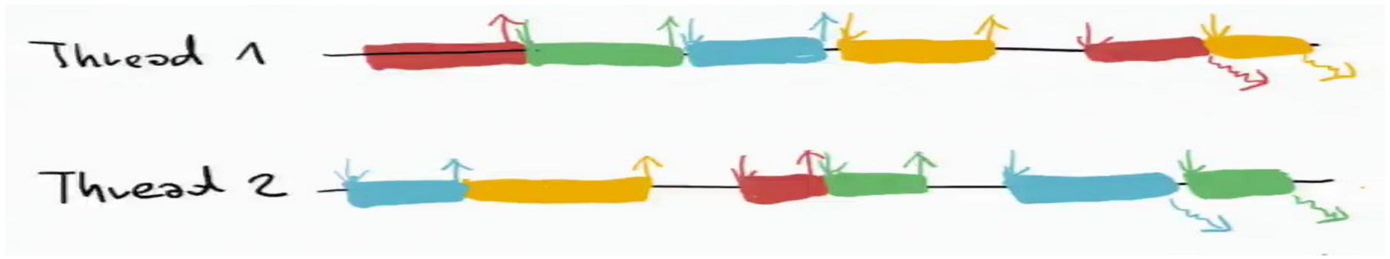
- To execute your Servlet, type the following URL in the Browser's address field:
  - http://localhost:8080/myApp/MyServlet



```
Hello You !!!
```

Running service() on a single thread

- By default, servlets written by specializing the HttpServlet class can have multiple threads concurrently running its service() method.



- If you would like to have only a single thread running a service method at a time, then your servlet should also implement the SingleThreadModel interface.
  - This does not involve writing any extra methods, merely declaring that the servlet implements the interface.

Example

```
public class SurveyServlet extends HttpServlet implements
    SingleThreadModel
{
    /* typical servlet code, with no threading concerns
     * in the service method. No extra code for the
     * SingleThreadModel interface. */
     ...
}
```

More About Servlets

Session Tracking

Persistent information

- A server site typically needs to maintain two kinds of persistent (remembered) information:
  - Information about the session
    - A session starts when the user logs in or otherwise identifies himself/herself, and continues until the user logs out or completes the transaction (for example, makes a purchase)
  - Information about the user
    - User information must generally be maintained much longer than session information (for example, remembering a purchase)
    - This information must be stored on the server, for example on a file or in a database

Server capabilities

- Servlets, like Applets, can be trusted or untrusted
  - A servlet can use a unique ID to store and retrieve information about a given session
  - User information usually requires a login ID and a password
  - Since servlets don’t quit between requests, *any* servlet can maintain information in its internal data structures, as long as the server keeps running
  - A *trusted* servlet can read and write files on the server, hence can maintain information about sessions and users even when the server is stopped and restarted
  - An untrusted servlet will lose *all* information when the servlet or server stops for any reason
    - This is sometimes good enough for session information
    - This is almost never good enough for user information

Session tracking

- HTTP is stateless: When it gets a page request, it has *no memory* of any previous requests from the same client
  - This makes it difficult to hold a "conversation"
    - Typical example: Putting things one at a time into a shopping cart, then checking out--each page request must somehow be associated with previous requests
  - The server must be able to keep track of multiple conversations with multiple users
- Session tracking is keeping track of what has gone before in this particular conversation
  - Since HTTP is stateless, it does not do this for you
  - You have to do it yourself, in your servlets

Session tracking solutions

- Cookies are small files that the servlet can store on the client computer, and retrieve later
- URL rewriting: You can append a unique ID after the URL to identify the user
- Hidden <form> fields can be used to store a unique ID
- Java’s Session Tracking API can be used to do most of the work for you

Hidden <form> fields

- <input type="hidden" name="sessionID" value="...">
- Advantage:
  - Requires the least knowledge: All you need to know is how to read and write parameters
- Disadvantages:
  - Not kept across sessions, so useless for maintaining persistent information about a user
  - Since the session ID must be incorporated into every HTML page, every HTML page must be dynamically generated
- There’s not much more to say about using hidden form fields, since you should already know enough to do it

Cookies

- A cookie is a small bit of text sent to the client that can be read again later
  - Limitations (for the protection of the client):
    - Not more than 4KB per cookie (more than enough in general)
    - Not more than 20 cookies per site
    - Not more than 300 cookies total
- Cookies are *not* a security threat
- Cookies *can be* a privacy threat
  - Cookies can be used to customize advertisements
  - Outlook Express allows cookies to be embedded in email
  - A servlet can read your cookies
    - Incompetent companies might keep your credit card info in a cookie
    - Netscape lets you refuse cookies to sites other than that to which you connected

Using cookies

- import javax.servlet.http.*;
- Constructor: Cookie(String name, String value)
- Assuming ***request*** is an HttpServletRequest and ***response*** is an HttpServletResponse,
  - ***response***.addCookie(cookie);
  - Cookie[ ] cookies = ***request***.getCookies();
    - String name = cookies[i].getName();
    - String value = cookies[i].getValue();
- There are, of course, many more methods in the HttpServletRequest, HttpServletResponse, and Cookie classes in the javax.servlet.http package

Some more Cookie methods

- public void setComment(String ***purpose***)
  - public String getComment()
- public void setMaxAge(int ***expiry***)
  - public int getMaxAge()
  - Max age in seconds after which cookie will expire
  - If expiry is negative, delete when browser exits
  - If expiry is zero, delete cookie immediately
- setSecure(boolean ***flag***)
  - public boolean getSecure()
  - Indicates to the browser whether the cookie should only be sent using a secure protocol, such as HTTPS or SSL

More HttpServletRequest methods

- public HttpSession getSession()
  - Gets the session object for this request (or creates one if necessary)
- public Enumeration getHeaderNames()
  - Gets an Enumeration of all the field names in the HTTP header
- public String getHeader(String name)
  - Given the header name, return its value
- public int getIntHeader(String name)
  - Given the header name, return its value as an int
  - Returns -1 if no such header
  - Could throw a NumberFormatException
- public Enumeration getHeaders(String ***name***)
  - Given the header name, return an Enumeration of all its values

The Session Tracking API

- The session tracking API is in javax.servlet.http.HttpSession and is built on top of cookies
- To use the session tracking API:
  - Create a session:
    - HttpSession session = ***request***.getSession();
      - Returns the session associated with this request
      - If there was no associated session, one is created
  - Store information in the session and retrieve it as needed:
    - session.setAttribute(***name***, ***value***);
    - Object ***obj*** = getAttribute(***name***);
- Session information is automatically maintained across requests