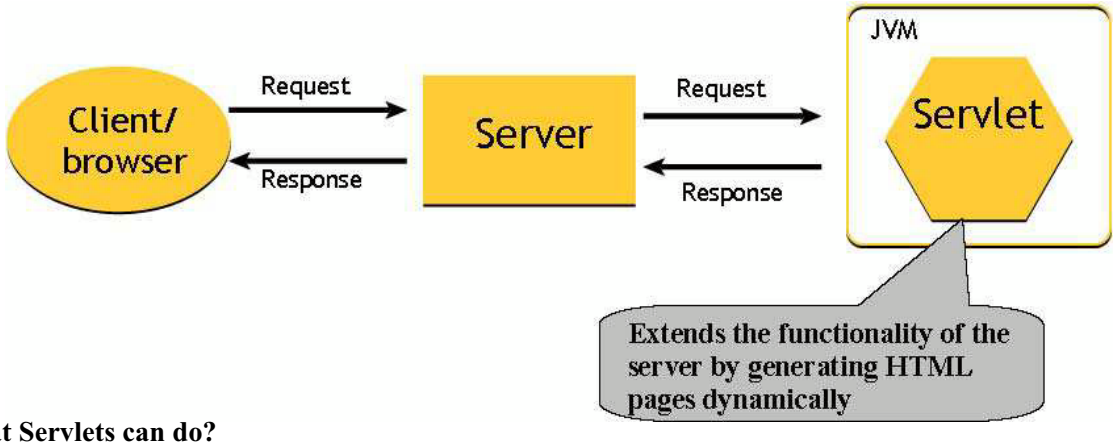
Java Servlets

Servlets are java technology's answer to CGI programming. CGI was widely used for generating dynamic content before Servlets arrived. They were programs written mostly in C,C++ that run on a web server and used to build web pages.
As you can see in the figure below, a client sends a request to web server, server forwards that request to a servlet, servlet generates dynamic content, mostly in the form of HTML pages, and returns it back to the server, which sends it back to the client. Hence we can say that servlet is extending the functionality of the webserver (The job of the earlier servers was to respond only to request, by may be sending the required html file back to the client, and generally no processing was performed on the server)



What Servlets can do?

- Servlets can do anything that a java class can do. For example, connecting with database, reading/writing data to/from file etc.
- Handles requests sent by the user (clients) and generates response dynamically (normally HTML pages).
- The dynamically generated content is send back to the user through a webserver (client)

Servlets vs. other SSP technologies

The java's servlet technology has following advantage over their counter parts:

Convenient

Servlets can use the whole java API e.g. JDBC. So if you already know java, why learn Perl or C. Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and sending HTTP headers, handling cookies and tracking session etc and many more utilities

Efficient

With traditional CGI, a new process is started for each request while with servlets each request is handled by a lightweight java thread, not a heavy weight operating system process. (more on this later)

Powerful

Java servlets let you easily do several things that are difficult or impossible with regular CGI. For example, servlets can also share data among each other

Portable

Since java is portable and servlets is a java based technology therefore they are generally portable across web servers

Inexpensive

There are numbers of free or inexpensive web servers available that are good for personal use or low volume web sites. For example Apache is a commercial grade webserver that is absolutely free. However

some very high end web and application servers are quite expensive e.g BEA weblogic. We'll also use Apache in this course

Software Requirements

To use java servlets will be needed

- J2SE
- Additional J2EE based libraries for servlets such as servlet-api.jar and jsp-api.jar. Since these libraries are not part of J2SE, you can download these APIs separately. However these APIs are also available with the web server you'll be using.
- A capable servlet web engine (webserver)

Jakarta Servlet Engine (Tomcat)

Jakarta is an Apache project and tomca subprojects. Apache Tomcat is an open source web server, which is used as an official r mentation of Java Servlets and Java Server Pages technologies.

Tomcat is developed in an open and participatory environment and released under the Apache software license

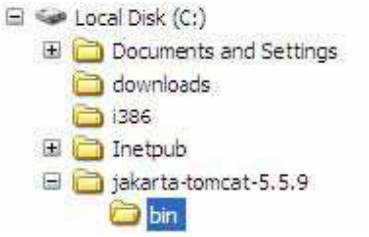
Environment Setup

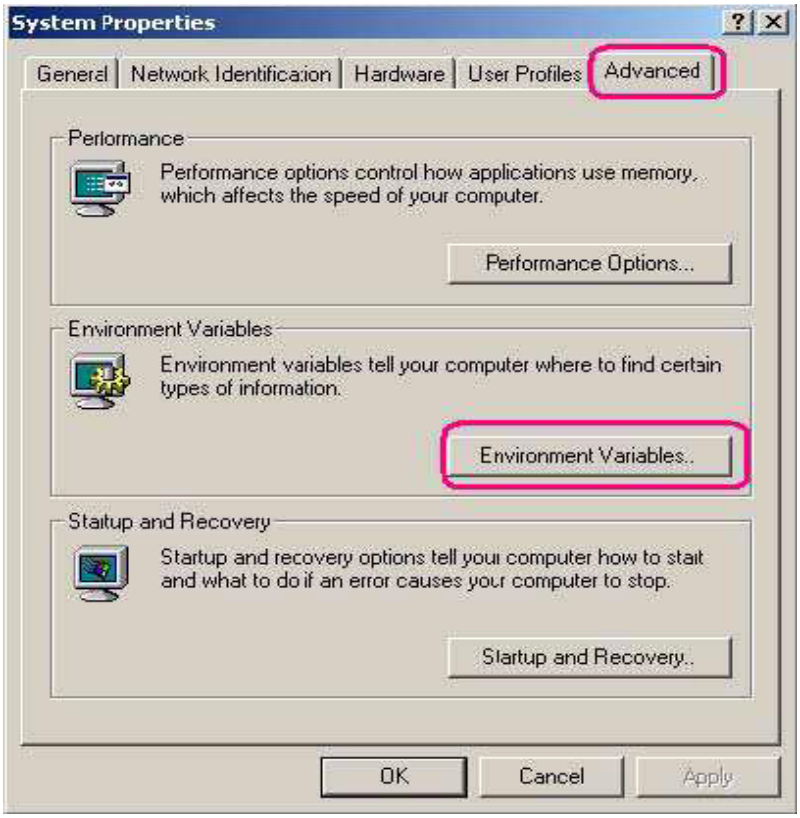
To work with servlets and JSP technologies, you first need to set up the environment. Tomcat installation can be performed in two different ways **(a)** using .zip file **(b**sing .exe file. This setup process is broken down into the following steps:

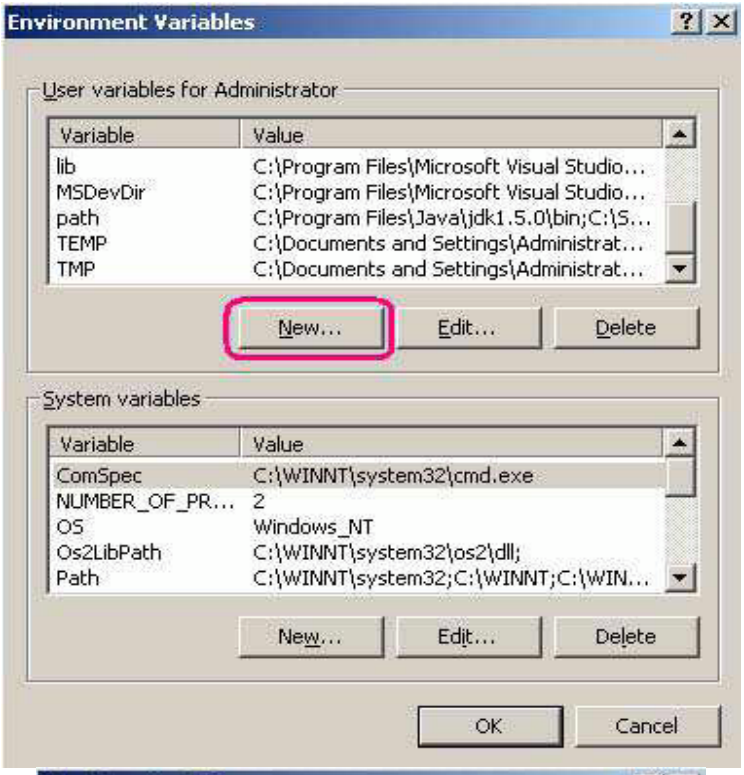
1 Download the Apache Tomcat Serv
ch as:

-The C:\jakarta-tomcat-5.5.9 folder is generally referred as **root** directory or CATALINA_HOME

Note: After extraction**,** make sure C:\jakarta-tomcat-5.5.9 contains a bin subdirectory. Sometimes students create their own directory and unzip the file there such as C:\jakarta-tomcat-5.5.9\jakarta-tomcat-5.5.9.This causes problems while giving path information



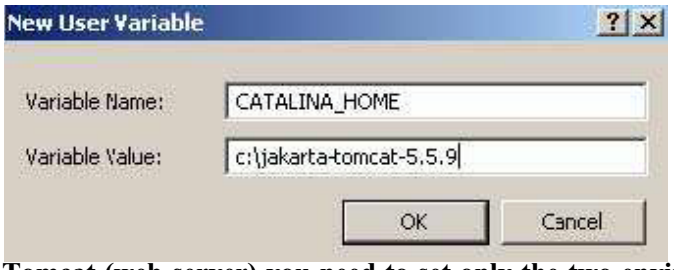

System Properties
General
Network Identification
Hardware
User Profiles
Advanced
Performance
Performance options control how applications use memory, which affects the speed of your computer.
Performance Options...
Environment Variables
Environment variables tell your computer where to find certain types of information.
Environment Variables...
Startup and Recovery
Startup and recovery options tell your computer how to start and what to do if an error causes your computer to stop.
Startup and Recovery...
OK
Cancel
Apply



4. Set the CATALINA_HOME variable

CATALINA_HOME is used to tell the system about the root directory of the TOMCAT. There are various files (classes, exe etc) needed by the system to run. CATALINA_HOME is used to tell your system (in this case your web server Tomcat) where the required files are.

- To Set the CATALINA_HOME environment variable, create another User Variable.
- Type CATALINA_HOME as the name of the environment variable.
- Its value should be the path till your top-level Tomcat directory. If you have unzipped the Tomcat in C drive. It should be C:\jakarta-tomcat-5.5.9.This is shown below:
- Press Ok button to finish



Note: To run Tomcat (web server) you need to set only the two environment variables and these are JAVA_HOME & CATALINA_HOME

5. Set the CLASSPATH variable

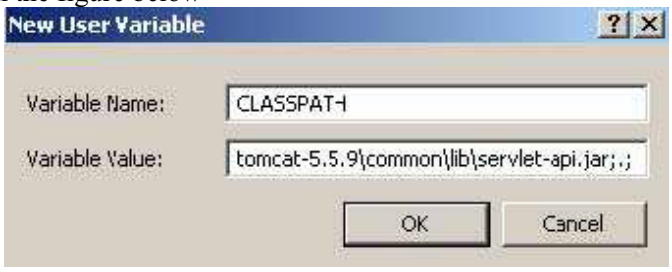
Since servlets and JSP are not part of the Java 2 platform, standard edition, you have to identify the servlet classes to the compiler. The server already knows about the servlet classes, but the compiler (i.e., javac) you use for compiling source files of servlet does not. So if you don't set your CLASSPATH, any attempt to compile servlets, tag libraries, or other classes that use the servlet API will fail with error messages about unknown classes.

- To Set the CLASSPATH environment variable, create another User Variable.
- Type CLASSPATH as the name of the environment variable.
- Its value should be the path for servlet-api.jar and jsp-api.jar. These file can be found on following path:

    - C:\jakarta-tomcat-5.5.9\common\lib**servlet-api.jar**
    - C:\jakarta-tomcat-5.5.9\common\lib**jsp-api.jar**
    - Both these api's are specified as values with semicolon between them. Remember to add semicolon dot semicolon (;.;) at the end too. For example

Classpath = C:\jakarta-tomcat-5.5.9\common\lib\servlet-api.jar**;**C:\jakarta-tomcat 5.5.9\common\lib\jsp-api.jar**;.;**

This is also shown in the figure below



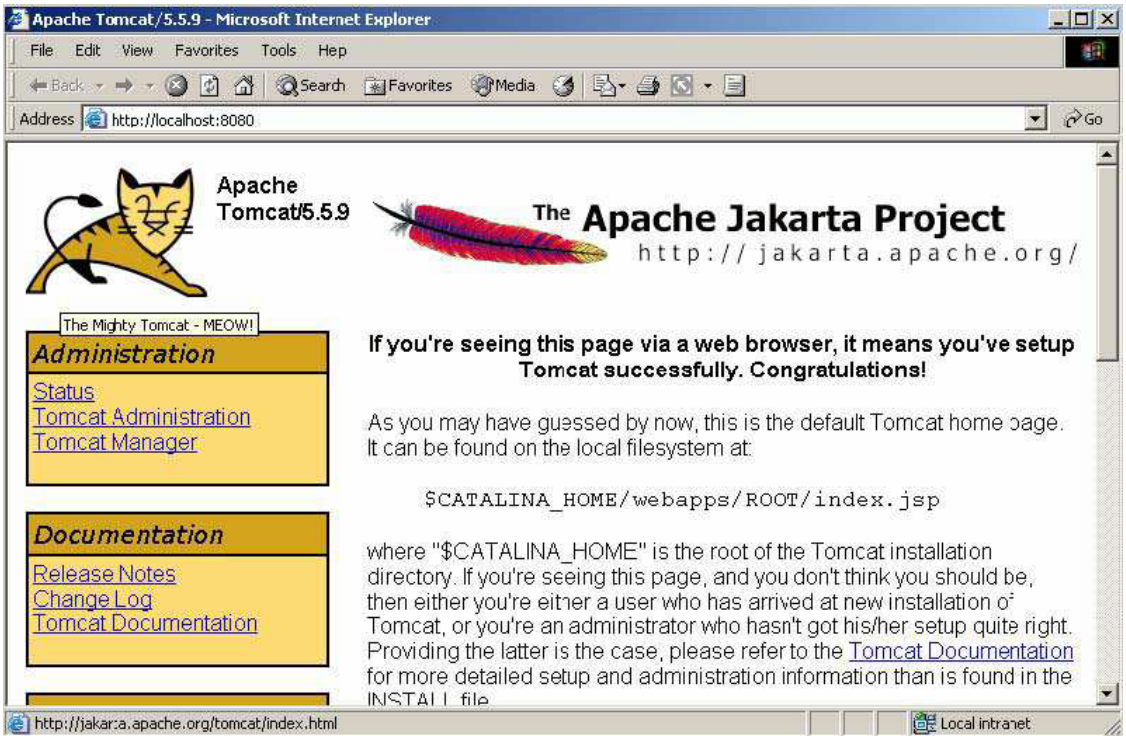
— Press OK button to finish the setting of CLASSPATH variable

6. Test the server

Before making your own servlets and JSP, verify that the server is working properly. Follow these steps in order to do that:

- Open the C:\jakarta-tomcat-5.5.9\bin folder and locate the startup.batfile.
- Double clicking on this file will open up a DOS window, which will disappear, and another DOS window will appear, the second window will stay there. If it does not your paths are not correctly set.
- Now to check whether your server is working or not, open up a browser window and type http://localhost:8080. This should open the default page of Tomcat as shown in next diagram:

Note: If default page doesn't displayed, open up an internet explorer window, move on to Tools Æ Internet Options Æ Connections LAN Settings. Make sure that option of "Bypass proxy server for local addresses" is unchecked.



There is another easier way to carry out the environment setup using .exe file. However, it is strongly recommended that you must complete the environment setup using .zip file to know the essential fundamentals.

Environment Setup Using .exe File

Let's look at the steps involved to accomplish the environment setup using .exefile.

1. Download the Apache Tomcat Server

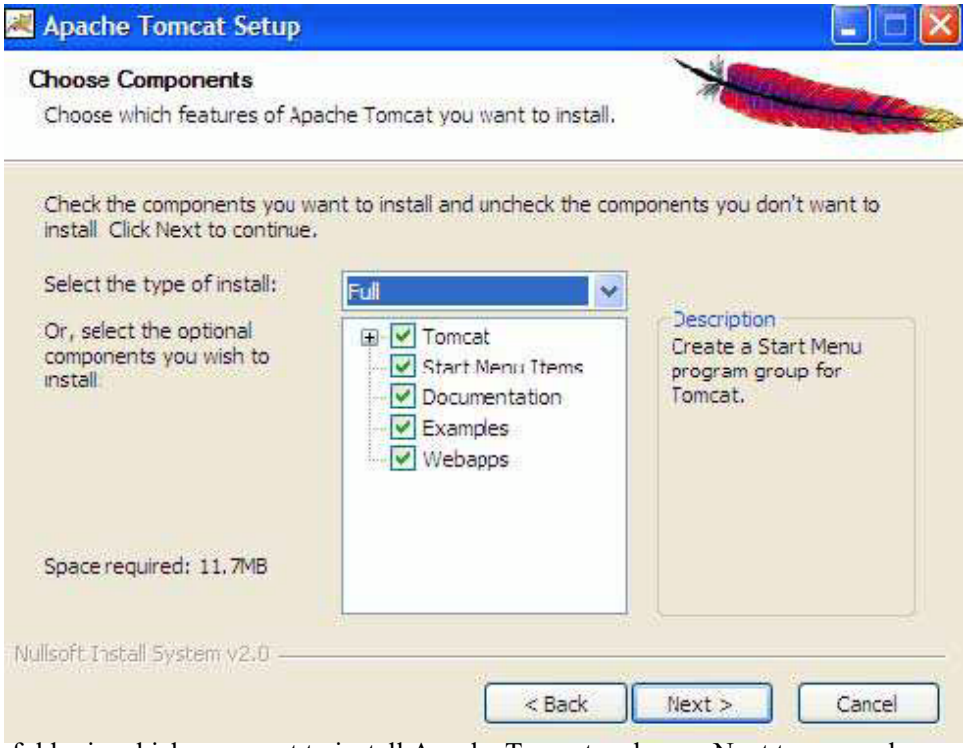
From the http://tomcat.apache.org, download the .exe file for the current release (e.g. jakarta-tomcat-5.5.9.zip) on your C:\ drive. There are different releases available on site. Select to download Windows executable (.exe) file from Binary Distributions Core section.

Note: J2SE 5.0 must be installed to use the 5.5.9 version of tomcat.

2. Installing Tomcat using .exe file

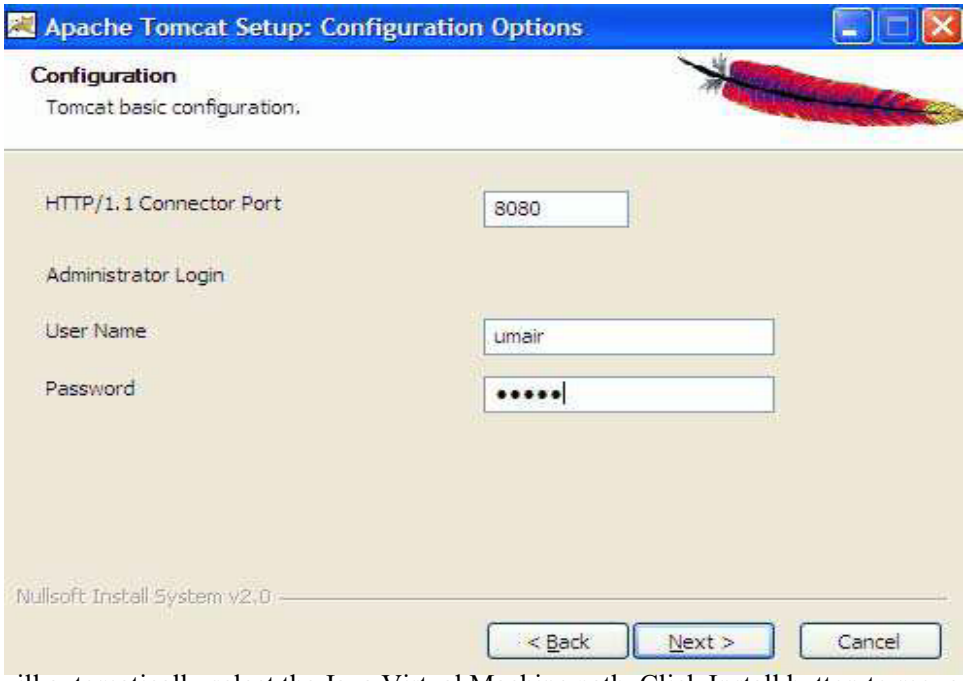
-Run the .exefile by double clicking on it.

-Moving forward in setup, you will reach to the following window -Select install type "Full" and press Next button to proceed.



-Choose the folder in which you want to install Apache Tomcat and press Next to proceed.

-The configuration window will be opened. Leave the port unchanged (since by default web servers run on port 8080, you can change it if you really want to). Specify the user name & password in the specified fields and press Next button to move forward. This is also shown in the diagram coming next:

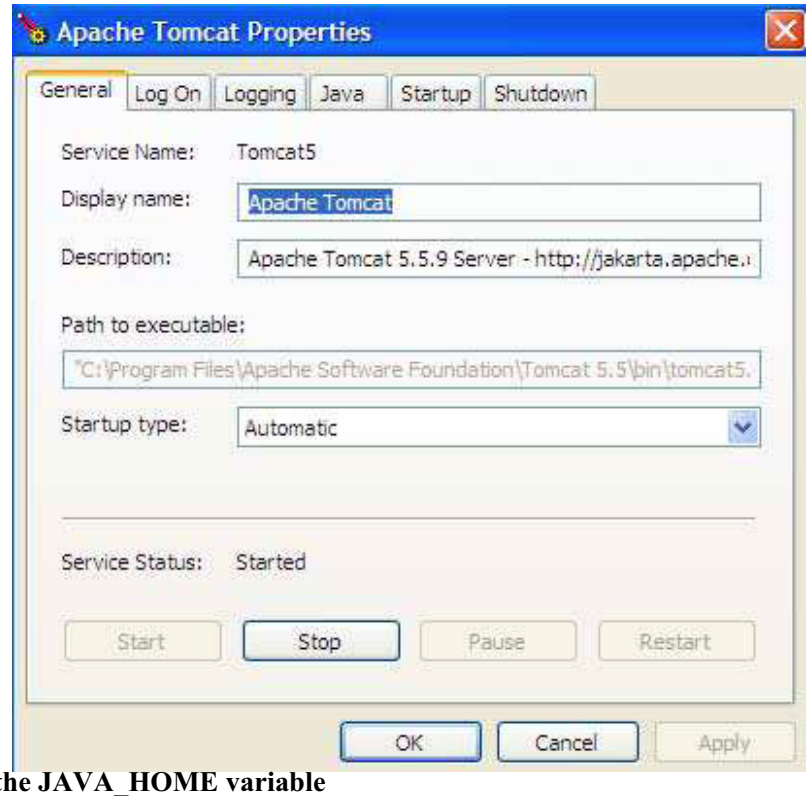


-The setup will automatically select the Java Virtual Machine path. Click Install button to move ahead.

-Finish the setup with the Run Apache Tomcat option selected. It will cause the tomcat server to run in quick launch bar as shown in diagram below. The Apache Tomcat shortcuts will also added to Programs menu.

Apache Tomcat

-Double clicking on this button will open up Apache Tomcat Properties window. From here you can start or stop your web server. You can also configure many options if you want to. This properties window is shown below:



3. Set the JAVA_HOME variable

Choosing .exe mode does not require completing this step.

4. Set the CATALINA_HOME variable

Choosing .exe mode does not require completing this step.

5. Set the CLASSPATH variable

Same as step 5 of .zip installation mode

6. Test the server

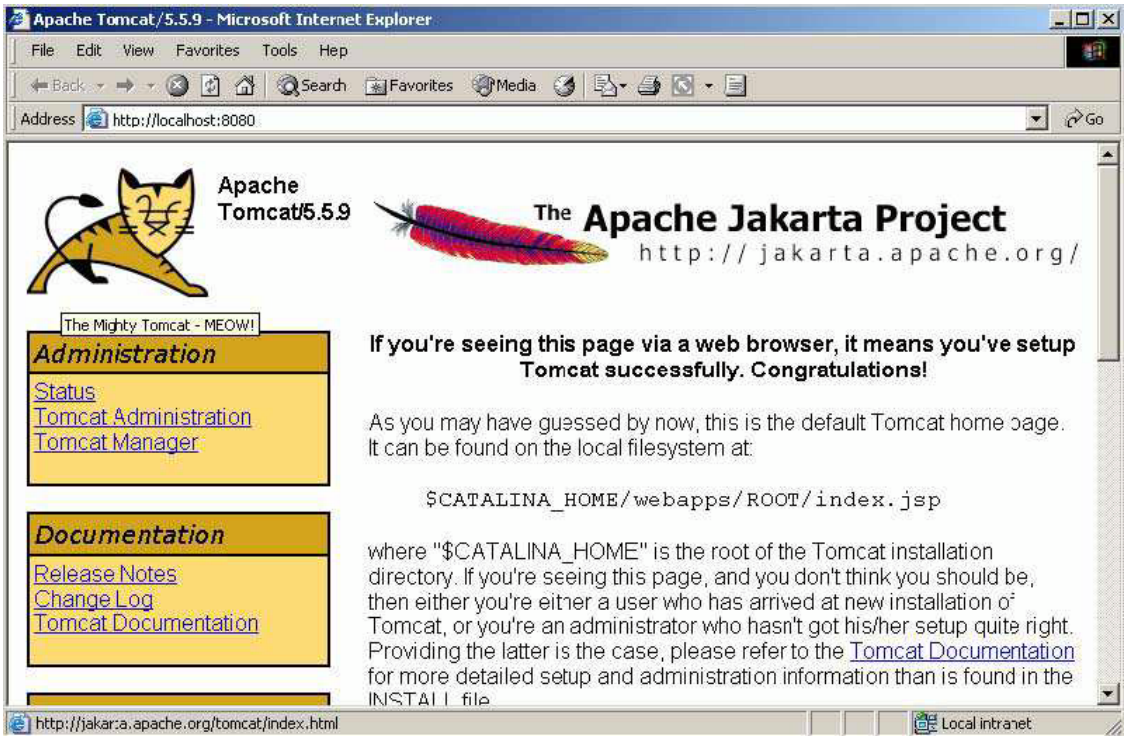
If tomcat installation is made using .exe file, follow these steps

-Open the Apache Tomcat properties window by clicking on the Apache Tomcat button from Quick Launch.

-Start the tomcat server if it is not running by clicking on Start button.

-Open up a browser window and type http://localhost:8080. This should open the default page of Tomcat as shown in the next diagram:

Note: If default page doesn't displayed, open up an internet explorer window, move on to Tools Æ Internet Options Æ Connections LAN Settings. Make sure that option of "Bypass proxy server for local addresses" is unchecked.



References:

. Java Servlet & JSP tutotrial http://www.apl.jhu.edu/~hall/**java**/Servlet-Tutorial/

Creating a Simple
Web Application in Tomcat

In this handout, we'll discuss the standard tomcat directory structure, a pre-requisite for building any web application. Different nuts and bolts of Servlets will also be discussed. In the later part of this handout, we'll also learn how to make a simple web application using servlet.

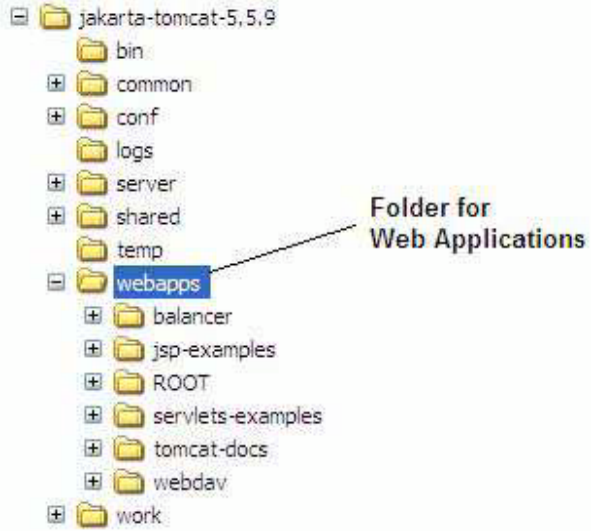
Standard Directory Structure of a J2EE Web Application

A web application is defined as a hierarchy of directories and files in a standard layout. Such hierarchies can be used in two forms

- Unpack
  - Where each directory & file exists in the file system separately
  - Used mostly during development

- Pack
  - Known as Web Archive (WAR) file
  - Mostly used to deploy web applications

The webapps folder is the top-level Tomcat directory that contains all the web applications deployed on the server. Each application is deployed in a separate folder often referred as "context".



To make a new application e.g myapp in tomcat you need a specific folder hierarchy.

-Create a folder named myapp in *C:\jakarta-tomcat-5.5.9\webapps* folder. This name will also appear in the URL for your application. For example
http://localhost:8080/myapp/index.html

-All JSP and html files will be kept in main application folder (C:\jakartatomcat-5.5.9\webapps\myapp)

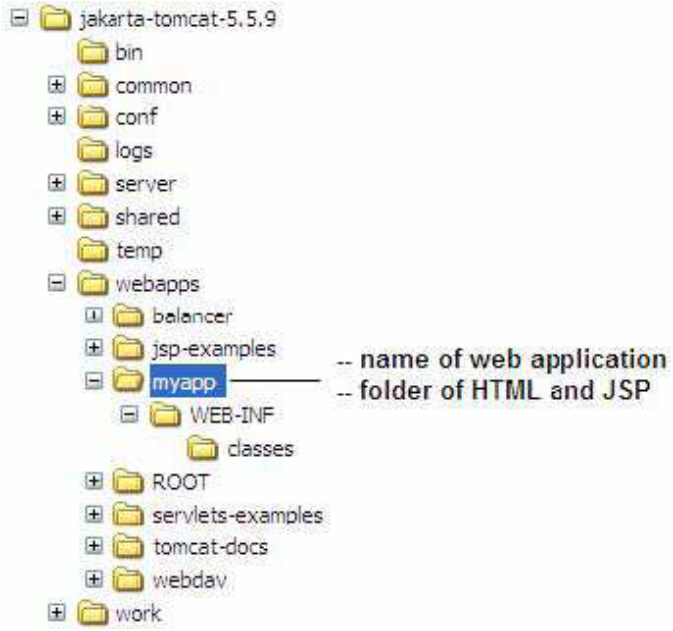
-Create another folder inside myapp folder and change its name to WEB-INF. Remember WEB-INF is case sensitive and it is **not** WEB_INF

-Configuration files such as web.xml will go in WEB-INF folder (C:\jakarta-tomcat-5.5.9\webapps\myapp\WEB-INF)

-Create another folder inside WEB-INF folder and change its name to classes. Remember classes name is also case sensitive.

-Servlets and Java Beans will go in classes folder (C:\jakarta-tomcat5.5.9\webapps\myapp\WEB-INF\classes)

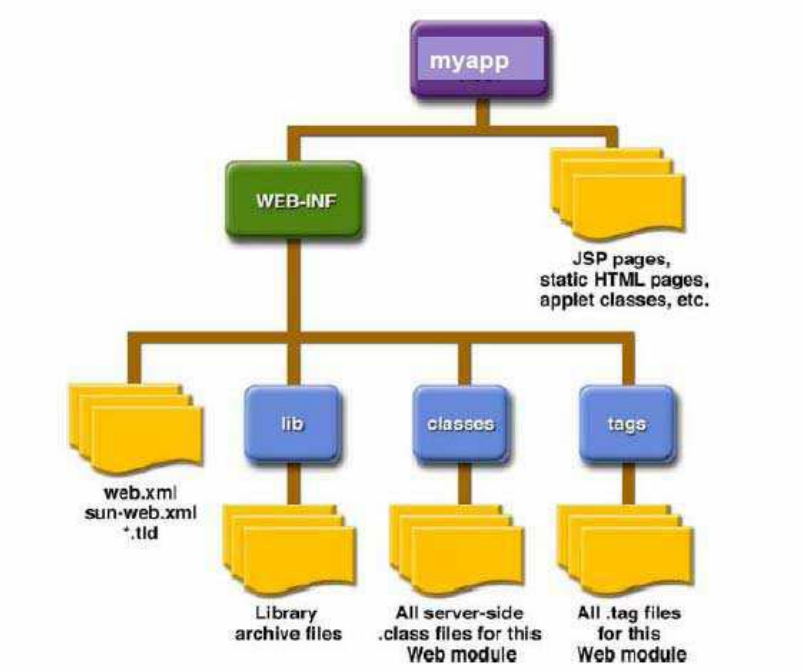
That's the minimum directory structure required in order to get started. This is also shown in the figure below:



-To test application hierarchy, make a simple html file e.g. index.html file. Write some basic HTML code into it and save it in main application directory i.e.
C:\jakarta-tomcat-5.5.9\webapps\myapp\

-Restart the server and access it by using the URL
http://localhost:8080/myapp/index.html

-A more detailed view of the Tomcat standard directory structure is given below.



-Here you can see some other folders like lib& tags under the WEB-INF.
-The lib folder is required if you want to use some achieve files (.jar). For example an API in jar format that can help generating .pdf files.
-Similarly tags folder is helpful for building custom tags or for using .tagfiles.

Note: Restart Tomcat every time you create a new directory structure, a servlet or a java bean so that it can recognize it. For JSP and html files you don't have to restart the server.

Writing Servlets

Servlet Types

-Servlet related classes are included in two main packages javax.servletand javax.servlet.http.
-Every servlet must implement the javax.servlet.Servlet interface, it contains the servlet's life cycle methods etc. (Life cycle methods will be discussed in next handout)
-In order to write your own servlet, you can subclass from GernericServlet or HttpServlet

GenericServlet class

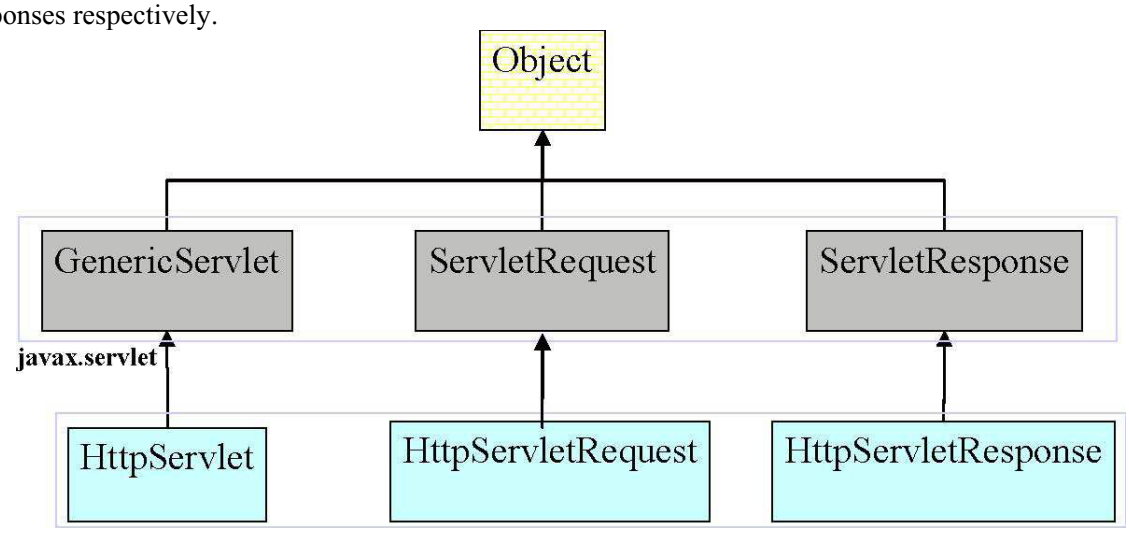
- Available in javax.servlet package
- Implements javax.servlet.Servlet
- Extend your class from this class if you are interested in writing protocol independent servlets

HttpServlet class

- Available in javax.servlet.http package
- Extends from GenericServletclass
- Adds functionality for writing HTTP specific servlets as compared to GernericServlet
- Extend your class from HttpServlet, if you want to write HTTP based servlets

Servlet Class Hierarchy

The Servlet class hierarchy is given below. Like all java classes GenericServletalso inherits from Object class. Apart from GenericServlet and HttpServlet classes, ServletRequest, HttpServletRequest, ServeltResponse and HttpServletResponse are also helpful in writing a servlet.
As you can guess ServletRequest & ServletResponse are used in conjunction with GenericServlet. These classes are used for processing protocol independent requests and generating protocol independent responses respectively.



HttpServletRequest & HttpServletRespose are used for processing HTTP protocol specific requests and generating HTTP specific response. Obviously these classes will be used in conjunction with HttpServet class, which means you are making a HTTP protocol specific servlet.

Types of HTTP requests

HTTP supports different types of request to be sent over to server. Each request has some specific purpose. The most important ones are **get** & **post**. Given below a brief overview of each request type is given. You can refer to RFC of HTTP for further details.

-GET: Requests a page from the server. This is the normal request used when browsing web pages.
-POST: This request is used to pass information to the server. Its most common use is with HTML forms.
-PUT: Used to put a new web page on a server.
-DELETE: Used to delete a web page from the server.
-OPTIONS: Intended for use with the web server, listing the supported options.
-TRACE: Used to trace servers

GET & POST, HTTP request types

Some details on GET and POST HTTP request types are given below.
. **GET**
-Attribute-Value pair is attached with requested URL after '?'.
-For example if attribute is 'name' and value is 'ali' then the request will be http://www.gmail.com/register**?name=ali**
-For HTTP based servlet, override doGet() methods of HttpServlet class to handle these type of requests.
. **POST**
-Attribute-Value pair attached within the request body. For your reference HTTP request diagram is given below again:

-Override doPost() method of HttpServlet class to handle POST type requests.

Steps for making a Hello World Servlet

To get started we will make a customary "HelloWorldServlet". Let's see what are the steps involved in writing a servlet that will produce "Hello World"

1. Create a directory structure for your application (i.e. helloapp). This is a one time process for any application
2. Create a HelloWorldServlet source file by extending this class from HttpServlet and overriding your desired method. For example doGet() or doPost().
3. Compile it (If get error of not having required packages, check your class path)
4. Place the class file of HelloWorldServlet in the classes folder of your web application (i.e. myapp).

Note: If you are using packages then create a complete structure under classes folder

5. Create a deployment descriptor (web.xml) and put it inside WEB-INF folder
6. Restart your server if already running
7. Access it using Web browser

Example Code: HelloWorldServlet.java

```
//File HelloWorldServlet.java

// importing required packages
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// extending class from HttpServelt
public class HelloWorldServlet extends HttpServlet {

        /* overriding doGet() method because writing a URL in the browserby default generate request of
            GET type
            As you can see, HttpServletRequest andHttpServletResponse are passed to this
            method. Theseobjects will help in processing of HTTP request andgenerating
            response for HTTP
            This method can throw ServletException or IOException, so wemention these exception
            types after method signature
       */
        public void doGet(HttpServletRequest request,HttpServletResponse response)
        throws ServletException, IOException
        {
                /* getting output stream i.e PrintWriter from responseobject by calling getWriter
                    method on it
                  As mentioned, for generating response, we will
                  useHttpServletResponse object*/
                PrintWriter out = response.getWriter();
                /* printing Hello World in the browser using PrintWriter
                    object. You can also write HTML like
                            out.println("<h1> Hello World </h1>") */
                out.println("Hello World! ");
        } // end doGet()
} // end HelloWorldServlet
```

Example Code: web.xml

eXtensible Markup Language (xml) contains custom defined tags which convey information about the content. To learn more about XML visit http://ww.w3schools.com.

Inside web.xml, the **<web-app>** is the root tag representing the web application. All other tags come inside of it.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
        <servlet> <servlet-name> HelloWorldServlet </servlet-name> <servlet-class>
                HelloWorldServlet </servlet-class>
        </servlet>
          <servlet-mapping><servlet-name> HelloWorldServlet </servlet-
                name><url-pattern> /myfirstservlet </url-
                pattern></servlet-mapping> </web-app>
```

The **<servlet>** tag represents one's servlet name and its class. To specify the name of servlet, **<servlet-name>** tag is used. Similarly to specify the class name of servlet (it is the same name you used for making a servlet), **<servlet-class>**tag is used.

Note: It is important to note here that you can specify any name for a servlet inside **<servlet-name>** tag. This name is used for referring to servlet in later part of web.xml. You can think of it as your id assigned to you by your university while you have actually different name (like **<servlet-class>**).
Next we will define the servlet mapping. By defining servlet mapping we are specifying URL to access a servlet. **<servlet-mapping>** tag is used for this purpose.
Inside **<servlet-mapping>** tag, first you will write the name of the servlet for which you want to specify the URL mapping using **<servlet-name>** tag and then you will define the URL pattern using **<url-pattern>** tag. Notice the forward slash (/) is used before specifying the url. You can specify any name of URL. The forward slash indicates the root of your application.

<url-pattern> **/myfirstservlet** </url-pattern>

Now you can access HelloWorldServelt (if it is placed in myapp application) by giving the following url in the browser

http://localhost:8080/**myapp/myfirstservlet**

Note: Save this web.xml file by placing double quotes("web.xml") around it as you did to save .java files.

Compiling and Invoking Servlets

- **-**Compile HelloWorldServlet.java using javac command.
- **-**Put HelloWorldServlet**.**class in C:\jakarta-tomcat5.5.9/webapps/myapp/WEB-INF/classes folder
- **-**Put web.xml file in C:\jakarta-tomcat5.5.9/webapps/myapp/WEB-INF folder
- **-**Invoke your servlet by writing following URL in web browser. Don't forget to restart your tomcat server if already running

  http://localhost:8080/myapp/myfirstservlet

Note: By using IDEs like netBeans® 4.1, you don't have to write web.xml by yourself or even to worry about creating directory structure and to copy files in appropriate locations. However manually undergoing this process will strengthen your concepts and will help you to understand the underlying mechanics ☺.

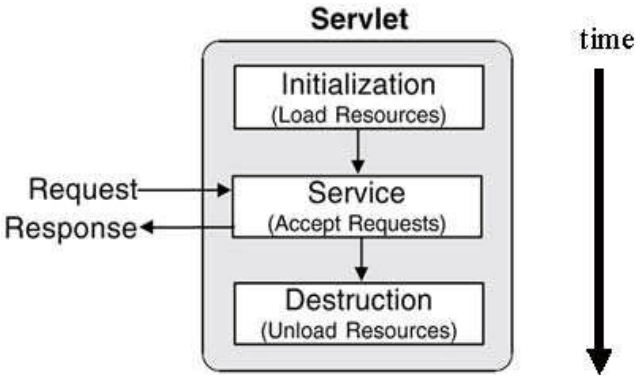
Servlets Lifecycle

In the last handout, we have seen how to write a simple servlet. In this handout we will look more specifically on how servlets get created and destroyed. What different set of method are invoked during the lifecycle of a typical servlet.
The second part consists on reading HTML form data through servlet technology. This will be explored in detail using code example

Stages of Servlet Lifecycle

A servlet passes through the following stages in its life.
1 Initialize
2 Service
3 Destroy

As you can conclude from the diagram below, that with the passage of time a servlet passes through these stages one after another.



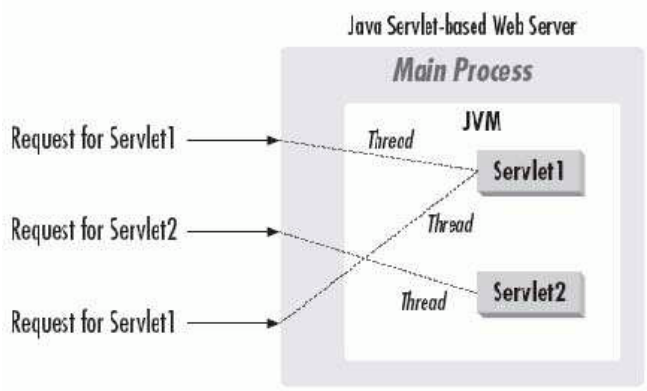
1. Initialize

When the servlet is first created, it is in the initialization stage. The webserver invokes he init() method of the servlet in this stage. It should be noted here that init() is only called once and is not called for each request. Since there is no constructor available in Servlet so this urges its use for one time initialization (loading of resources, setting of parameters etc) just as the init() method of applet.
Initialize stage has the following characteristics and usage

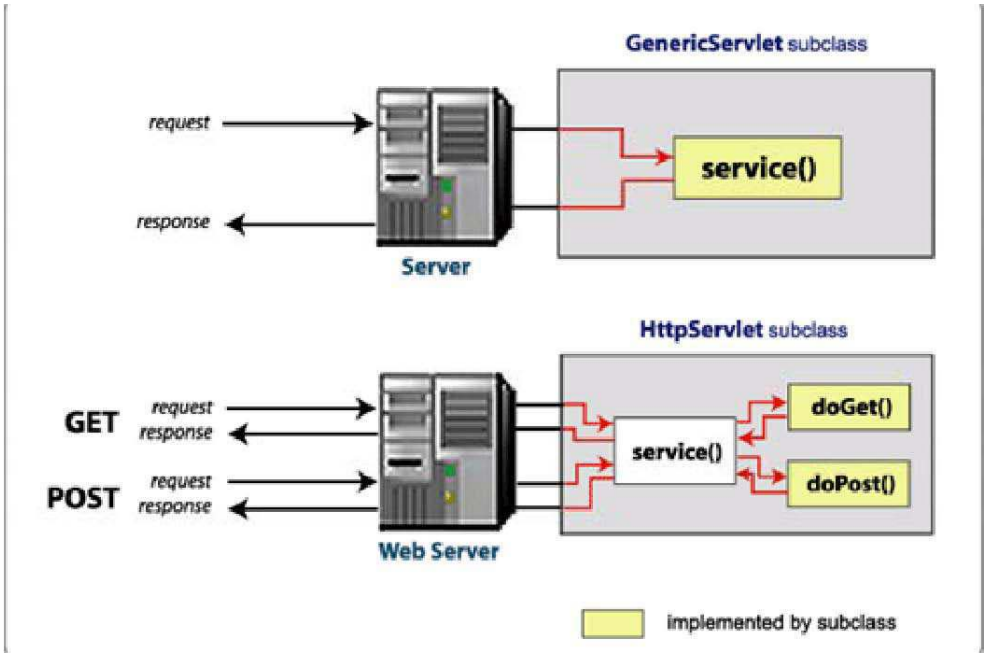
- Executed once, when the servlet gets loaded for the first time
- Not called for each client request
- The above two points make it an ideal place to perform the startup tasks which are done in constructor in a normal class.

2. Service

The service() method is the engine of the servlet, which actually processes the client's request. On every request from the client, the server spawns a new thread and calls the service() method as shown in the figure below. This makes it more efficient as compared to the technologies that use single thread to respond to requests.



The figure below show both versions of the implementation of service cycle. In the upper part of diagram, we assume that servlet is made by sub-classing from GenericServlet. (Remember, GenericServlet is used for constructing protocol independent servlets.). To provide the desired functionality, service() method is overridden. The client sends a request to the web server; a new thread is created to serve this request followed by calling the service() method. Finally a response is prepared and sent back to the user according to the request.



The second part of the figure illustrates a situation in which servlet is made using HttpServlet class. Now, this servlet can only serves the HTTP type requests. In these servlets doGet() and doPost() are overridden to provide desired behaviors. When a request is sent to the web server, the web server after creating a thread, passes on this request to service() method. The service() method checks the HTTP requests type (GET, POST etc) and calls the doGet() or doPost() method depending on how the request is originally sent. After forming the response by doGet() or doPost() method, the response is sent back to the service() method that is finally sent to the user by the web server.

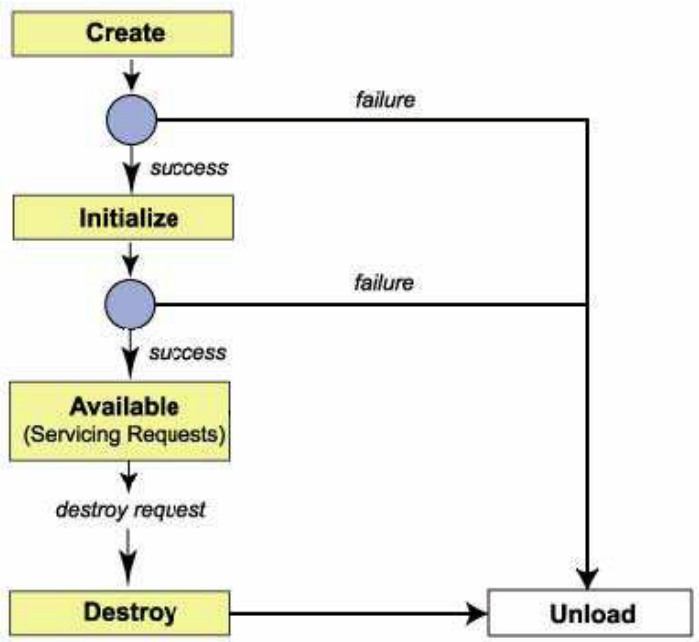
3. Destroy

The web server may decide to remove a previously loaded servlet instance, perhaps because it is explicitly asked to do so by the server administrator, or perhaps servlet container shuts down or the servlet is idle for a long time, or may be the server is overloaded. Before it does, however it calls the servlets destroy() method. This makes it a perfect spot for releasing the acquired resources.

Summary

- A Servlet is constructed and initialized. The initialization can be performed inside of init() method.
- Servlet services zero or more requests by calling service() method that may decide to call further methods depending upon the Servlet type (Generic or HTTP specific)
- Server shuts down, Servlet is destroyed and garbage is collected

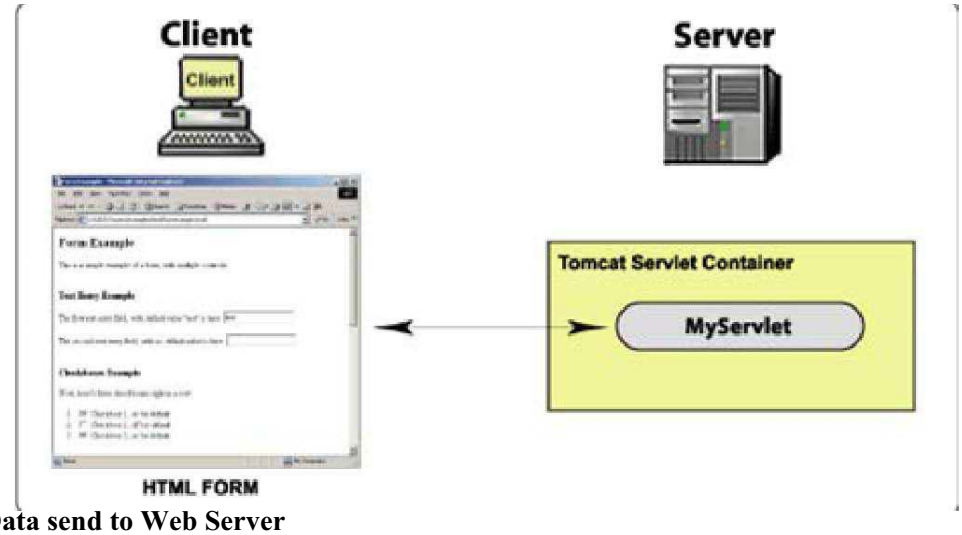
The following figure can help to summarize the life cycle of the Servlet



The web sever creates a servlet instance. After successful creation, the servlet enters into initialization phase. Here, init() method is invoked for once. In case web server fails in previous two stages, the servlet instance is unloaded from the server.
After initialization stage, the Servlet becomes available to serve the clients requests and to generate response accordingly. Finally, the servlet is destroyed and unloaded from web server.

Reading HTML Form Data Using Servlets
In the second part, the required concepts and servlet technology is explored in order to read HTML form data. To begin with, let's first identify in how many ways a client can send data

HTML & Servlets
Generally HTML is used as a Graphics User Interface for a Servlet. In the figure below, HTML form is being used as a GUI interface for MyServlet. The data entered by the user in HTML form is transmitted to the MyServlet that can process this data once it read out. Response may be generated to fulfil the application requirements.



Types of Data send to Web Server

When a user submits a browser request to a web server, it sends two categories of data:

. **Form Data**

Data, that the user explicitly type into an HTML form. For example: registration information provided for creating a new email account.

. **HTTP Request Header Data**

Data, which is automatically, appended to the HTTP Request from the client for example, cookies, browser type, and browser IP address.

Based on our understanding of HTML, we now know how to create user forms. We also know how to gather user data via all the form controls: text, password, select, checkbox, radio buttons, etc. Now, the question arises: if I submit form data to a Servlet, how do I extract this form data from servlet? Figuring this out, provides the basis for creating interactive web applications that respond to user requests.

Reading HTML Form Data from Servlet

Now let see how we can read data from "HTML form" using Servlet. The HttpServletRequest object contains three main methods for extracting form data submitted by the user:

. **getParameter(String name)**

-Used to retrieve a single form parameter and returns String corresponding to name specified.
-Empty String is returned in the case when user does not enter any thing in the specified form field.
-If the name specified to retrieve the value does not exist, it returns null.
Note: You should only use this method when you are sure that the parameter has only one value. If the parameter might have more than one value, use getParamterValues().

. **getParameterValues(String name)**

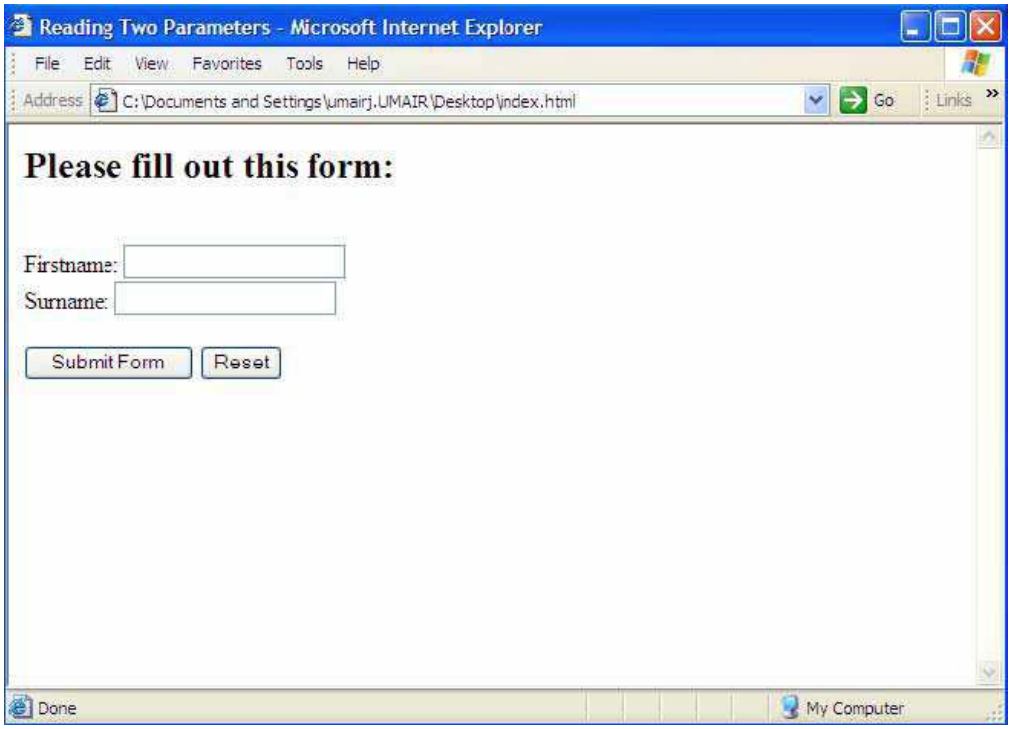
-Returns an array of Strings objects containing all of the given values of the given request parameter.
-If the name specified does not exist, null is returned

. **getParameterNames()**

-If you are unsure about the parameter names, this method will be helpful
-It returns Enumeration of String objects containing the names of the parameters that come with the request.
-If the request has no parameters, the method returns an empty Enumeration.

Note: All these methods discussed above work the same way regardless of the request type(GET or POST). Also remember that form elements are case sensitive for example, "userName" is not the same as the "username."

Example Code: Reading Form Data using Servlet
This example consists of one HTML page (**index.html**), one servlet **(MyServlet.java**) and one xml file **(web.xml)** file. The HTML page contains two form parameters: firstName and surName. The Servlet extracts these specific parameters and echoes them back to the browser after appending "Hello".
Note: The example given below and examples later in coming handouts are built using netBeans®4.1. It's important to note that tomcat server bundled with netBeans® runs on 8084 port by default.
index.html



Let's have a look on the HTML code used to construct the above page.

```
<html>
        <head>
        <title> Reading Two Parameters </title>
        </head>

        <body>
        <H2> Please fill out this form: </H2>
        <FORM METHOD="GET"
        ACTION="http://localhost:8084/paramapp/formservlet"NAME="myform" >
        <BR> Firstname:
        <INPUT TYPE = "text" NAME="firstName">
        <BR> Surname:
        <INPUT TYPE = "text" NAME="surName">
         <BR> <INPUT TYPE="submit" value="Submit Form"> <INPUT TYPE="reset"
        value="Reset">
        </FORM>
        </body>
</html>
```

Let's discuss the code of above HTML form. As you can see in the <FORM> tag, the attribute METHODis set to "GET". The possible values for this attribute can be GET and POST. Now what do these values

mean?

- Setting the method attribite to “GET” means that we want to send the HTTP request using the GET method which will evantually activate the doGet() method of the servlet. In the GET method the information in the input fields entered by the user, merges with the URL as the query string and are visible to the user.
- Setting METHOD value to “POST” hides the entered information from the user as this information becomes the part of request body and activates doPost() method of the servlet.

Attribute ACTION of <FROM> tag is set to **http://localhost:8084/paramapp/formservlet**. The form data will be transmitted to this URL. paramapp is the name of web application created using netBeans. formservlet is the value of <url-pattern> defined in the web.xml. The code of web.xml is given at the end. The NAME attribute is set to “myform” that helps when the same page has more than one forms. However, here it is used only for demonstration purpose.
To create the text fields where user can enter data, following lines of code come into play

```
<INPUT TYPE = “text” NAME="firstName">
<INPUT TYPE = “text” NAME="surName">
```

Each text field is distinguished on the basis of name assigned to them. Later these names also help in extracting the values entered into these text fields.

MyServlet.java
Now lets take a look at the servlet code to which HTML form data is submitted.

```
import java.io.*;import javax.servlet.*;import
javax.servlet.http.*;
   public class MyServlet extends HttpServlet{
   public void doGet(HttpServletRequest req,
                                                    HttpServletResponse res)throws ServletException,
      IOException{
          // reading first name parameter/textfield
          String fName = req.getParameter(“firstName”);
          // reading surname parameter/textfield
          String sName = req.getParameter(“surName”);
          // gettting stream from HttpServletResponse objectPrintWriter out = res.getWriter();
          out.println("Hello: " + fName + “ “ +sName ");
          out.close();
      }
}// end FormServlet
```

We started the code with importing three packages.

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

These packages are imported to have the access on PrintWriter, HttpServlet, HttpServletRequest, HttpServletResponse, ServletException and IOException classes.

The class MySevlet extends from HttpServlet to inherit the HTTP specific functionality. If you recall HTML code (index.html) discussed above, the value of mehtod attribute was set to “GET”. So in this case, we only need to override doGet()Method.

Entering inside doGet() method brings the crux of the code. These are:

```
String fName = req.getParameter(“firstName”);
```

String sName = req.getParameter(**“surName”**);

Two String variables fName and sName are declared that receive String values returned by getParameter() method. As discussed earlier, this method returns Stringcorresponds to the form parameter. Note that the values of name attributes of input tags used in index.html have same case with the ones passed to getParameter() methods as parameters. The part of HTML code is reproduced over here again:

<INPUT TYPE = “text” NAME="firstName">
<INPUT TYPE = “text” NAME="surName">

In the last part of the code, we get the object of PrintWriter stream from the object of HttpServletResponse. This object will be used to send data back the response. Using PrintWriter object (out), the names are printed with appended “Hello” that becomes visible in the browser.

web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app>
        <servlet>
                <servlet-name> FormServlet </servlet-name>
                <servlet-class> MyServlet </servlet-class>
        </servlet>

        <servlet-mapping>
                <servlet-name> FormServlet </servlet-name>
                <url-pattern> /formservlet </url-pattern>
        </servlet-mapping>
</web-app>
```

The <servlet-mapping> tag contains two tags <servlet-name> and <urlpatteren> containing name and pattern of the URL respectively. Recall the value of action attribute of the <form> element in the HTML page. You can see it is exactly the same as mentioned in **<**url-pattern> tag.

http://localhost:8084/paramapp/**formservlet**

References:

- Java API documentation
- Core Servlets and JSP by Marty Hall