Web Servers:

The Web server is meant for keeping Websites. It Stores and transmits web documents (files). It uses the HTTP protocol to connect to other computers and distribute information. Example: IIS, Apache, Sun Java System Web Server.

Tomcat Server Installation:

Step-1: Download and install Tomcat. To perform this:

Goto http://tomcat.apache.org and choose *Downloads* under this choose Tomcat 8.0 where look for "8.0.{xx}" (where {xx} is the latest upgrade number). Or choose Binary Distributions ⇒ Core ⇒ "zip" package (e.g., "apache-tomcat-8.0.{xx}.zip", about 8 MB).

UNZIP into a directory of your choice. DO NOT unzip onto the Desktop (because its path is hard to locate). I suggest using "d:\myproject". Tomcat will be unzipped into directory "d:\myproject\apache-tomcat-8.0.{xx}". For ease of use, we shall shorten and rename this directory to "d:\myproject\tomcat". Take note of Your Tomcat Installed Directory. Hereafter, I shall refer to the Tomcat installed directory as <TOMCAT_HOME> (or <CATALINA_HOME> - "Catalina" is the codename for Tomcat 5 and above).

A better approach is to keep the original directory name, such as apache-tomcat-8.0.{xx}, but create a symlink called tomcat via command "mklink /D tomcat apache-tomcat-8.0.{xx}". Symlink is available in Windows Vista/7/8 only.

Installing Tomcat on Windows can be done easily using the Windows installer.

Installation as a service: Tomcat will be installed as a Windows service no matter what setting is selected. Using the checkbox on the component page sets the service as "auto" startup, so that Tomcat is automatically started when Windows starts. For optimal security, the service should be run as a separate user, with reduced permissions.

Java location: The installer will provide a default JRE to use to run the service. The installer uses the registry to determine the base path of a Java 6 or later JRE, including the JRE installed as part of the full JDK. When running on a 64-bit operating system, the installer will first look for a 64-bit JRE and only look for a 32-bit JRE if a 64-bit JRE is not found. It is not mandatory to use the default JRE detected by the installer. Any installed Java 6 or later JRE (32-bit or 64-bit) may be used.

Tray icon: When Tomcat is run as a service, there will not be any tray icon present when Tomcat is running. Note that when choosing to run Tomcat at the end of installation, the tray icon will be used even if Tomcat was installed as a service.

The installer will create shortcuts allowing starting and configuring Tomcat. It is important to note that the Tomcat administration web application can only be used when Tomcat is running.

Tomcat's Directories:

Tomcat installed directory contains the following sub-directories:

1. bin: contains the binaries; and startup script (startup.bat for Windows and startup.sh for Unix and Mac), shutdown script (shutdown.bat for Windows and shutdown.sh for Unix and Mac), and other binaries and scripts.
2. conf: contains the system-wide configuration files, such as server.xml, web.xml, context.xml, and tomcat-users.xml.

3. lib: contains the Tomcat's system-wide JAR files, accessible by all webapps. You could also place external JAR file (such as MySQL JDBC Driver) here.
4. logs: contains Tomcat's log files. You may need to check for error messages here.
5. webapps: contains the webapps to be deployed. You can also place the WAR (Webapp Archive) file for deployment here.
6. work: Tomcat's working directory used by JSP, for JSP-to-Servlet conversion.
7. temp: Temporary files.

STEP-2: Create an Environment Variable JAVA_HOME

Now we need to create an environment variable called "JAVA_HOME" and set it to your JDK installed directory.

First, take note of your JDK installed directory. The default is "c:\Program Files\Java\jdk1.7.0_{xx}", where {xx} is the latest upgrade number. It is important to verify your JDK installed directory, via the "Computer", before you proceed further.

Start a CMD shell, and issue the command "set JAVA_HOME" to check if variable JAVA_HOME has been set:

> set JAVA_HOME
Environment variable JAVA_HOME not defined

If JAVA_HOME is set, check if it is set to your JDK installed directory correctly otherwise, goto next step. To set the environment variable JAVA_HOME in Windows 2000/XP/Vista/7/8: Push "Start" button ⇒ Control Panel ⇒ System ⇒ (Vista/7/8) Advanced system settings ⇒ Switch to "Advanced" tab ⇒ Environment Variables ⇒ System Variables ⇒ "New" (or "Edit" for modification) ⇒ In "Variable Name", enter "JAVA_HOME" ⇒ In "Variable Value", enter your JDK installed directory (e.g., "c:\Program Files\Java\jdk1.7.0_{xx}").

To verify, RE-START a CMD shell (need to refresh the environment) and issue:

> set JAVA_HOME
JAVA_HOME=c:\Program Files\Java\jdk1.7.0_{xx}

STEP-3: Configure Tomcat Server

The Tomcat configuration files are located in the "conf" sub-directory of your Tomcat installed directory, e.g. "d:\myproject\tomcat\conf" (for Windows). There are 4 configuration XML files:

server.xml

web.xml

context.xml

tomcat-users.xml

Make a BACKUP of the configuration files before you proceed.

Use a programming text editor (e.g., NotePad++, NotePad, TextPad for Windows) to open the configuration file "server.xml", under the "conf" sub-directory of Tomcat installed directory.

The default TCP port number configured in Tomcat is 8080, you may choose any number between 1024 and 65535, which is not used by an existing application. We shall choose 9999 for example.

Locate the following lines, and change port="8080" to port="9999".

<!-- A "Connector" represents an endpoint by which requests are received and responses are returned. Documentation at :

Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)

Java AJP Connector: /docs/config/ajp.html

APR (HTTP/AJP) Connector: /docs/apr.html

Define a non-SSL HTTP/1.1 Connector on port 8080

-->

<Connector port="9999" protocol="HTTP/1.1"

connectionTimeout="20000"

redirectPort="8443" />

In order to test the installation, whether it is done correctly or not, enter the address http://localhost:8080/ in the address bar of the browser and look for the index page of the tomcat server is loaded.

Introduction to Servlets:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically. Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

1. Performance is significantly better.
2. Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
3. Servlets are platform-independent because they are written in Java.
4. The full functionality of the Java class libraries is available to a Servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Lifecycle of a Servlet:

A Servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a Servlet:

1. The Servlet is initialized by calling the init () method.
2. The Servlet calls service() method to process a client's request.
3. The Servlet is terminated by calling the destroy() method.
4. Finally, Servlet is garbage collected by the garbage collector of the JVM.

The init method is designed to be called only once. It is called when the Servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets.

The Servlet is normally created when a user first invokes a URL corresponding to the Servlet, but you can also specify that the Servlet be loaded when the server is first started.

When a user invokes a Servlet, a single instance of each Servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the Servlet.

The init method definition looks like this:

public void init() throws ServletException { // Initialization code... }

The service() method is the main method to perform the actual task. The Servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a Servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method:

public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException { }

A GET request results from a normal request for a URL or from an HTML form that has no method specified and it should be handled by doGet() method.

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { // Servlet code }

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { // Servlet code }

The destroy() method is called only once at the end of the life cycle of a Servlet. This method gives your Servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the Servlet object is marked for garbage collection. The destroy method definition looks like this:

public void destroy() { // Finalization code... }

JSDK:

JSDK (Java Servlet Development Kit) is a package containing all the classes and interfaces needed to develop Servlets. JSDK also contains a web server and Servlet engine to test your creations. The Servlet engine provided in JSDK is a basic one(but free). There are many other Servlet engines much more robust and can be interfaced with most major web servers of the market.

To install JSDK double-click on the executable under Windows. Simply copy the file tree in the root directory of the JDK, no configuration is necessary. To verify that the installation was performed correctly, simply start the servletrunner utility, the Servlet engine included in the JSDK, that is to say, a basic server running on port 8080. If JSDK utility is correctly installed servletrunner should return the following lines:

servletrunner
servletrunner starting with settings :
port = 8080
backlog = 50
max handlers = 100
timeout = 5000
servletdir = ./examples
document dir = ./examples

servlet propfile = ./examples/servlet.properties

Example:

```
import java.io.*;
import javax.servlet.*;
public class HelloServlet extends GenericServlet
{
public void service(ServletRequest req, ServletRespone res) throws ServletException, IOException
{
res.setCcontentType("text/html");
printerWriter pw = res.getWriter( );
pw.println("Hello");
pw.close( );
}
}
```

The Servlet API:

The Servlet API contains two packages 1) javax.servlet and 2) javax.servlet.http

javax.servlet Package:

The javax.servlet package contains a number of classes and interfaces that describe and define the contracts between a Servlet class and the runtime environment provided for an instance of such a class by a conforming Servlet container.

Interface	Description
Servlet	Declares life cycle methods for a servlet
ServletConfig	Allows Servlets to get initialization parameters
ServletContext	Enables Servlets to log events
ServletRequest	Used to read data from a client request
ServletResponse	Used to read data to a client response

Interface *Summary*

Filter *A filter is an object that performs filtering tasks on either the request to a resource (a Servlet or static content), or on the response from a resource, or both.*

FilterChain *A FilterChain is an object provided by the Servlet container to the developer giving a view into the invocation chain of a filtered request for a resource.*

FilterConfig *A filter configuration object used by a Servlet container to pass information to a filter during initialization.*

RequestDispatcher *Defines an object that receives requests from the client and sends them to any resource (such as a Servlet, HTML file, or JSP file) on the server.*

Servlet *Defines methods that all Servlets must implement.*

ServletConfig *A Servlet configuration object used by a Servlet container to pass information to a Servlet during initialization.*

ServletContext *Defines a set of methods that a Servlet uses to communicate with its Servlet container, for example, to get the MIME type of a file, dispatch requests, or write to a log file.*

ServletContextAttributeListener *Implementations of this interface receive notifications of changes to the attribute list on the Servlet context of a web application.*

ServletContextListener *Implementations of this interface receive notifications about changes to the Servlet context of the web application they are part of.*

ServletRequest *Defines an object to provide client request information to a Servlet.*

ServletRequestAttributeListener *A ServletRequestAttributeListener can be implemented by the developer interested in being notified of request attribute changes.*

ServletRequestListener *A ServletRequestListener can be implemented by the developer interested in being notified of requests coming in and out of scope in a web component.*

ServletResponse *Defines an object to assist a Servlet in sending a response to the client.*

Class	Description
GenericServlet	Implements the Servlet and ServletConfig
ServletInputStream	Provides an input stream for reading requests from a client
ServletOutputStream	Provides an output stream for writing responses to a client.
ServletException	Indicates that a servlet error occurred.

Following are the interfaces and their methods

Servlet Interface:

void destroy	Called when the servlet is unloaded
ServletConfig getServletConfig()	Returns a ServletConfig object that contains any initialization parameters
String getServletInfo	Returns a string describing the servlet
void init()	Called when the servlet is initialized
void service	Called to process a request from a client

ServletConfig Interface:

ServletContext getServletContext	Returns the context for this servlet
String getInitParmeter(String param)	Returns the value of the initialization parameter name param
getInitParameterNames()	Returns all initialization parameter names

ServletContext Interface:

getAtrribute(String attr)	Returns the value of server attributes named attr.
String getServiceInfo()	Returns information about the server.
Servlet getServlet(String sname)	Returns the servlet named sname.
getServletNames()	Returns the names of Servlets in the server

ServletRequest Interface:

String getParameter(String pname)	Returns the value of the parameter named pname
getParameterNames()	Returns the parameter names for this request
String[ ] getParameterValues()	Returns the parameter values for this request
String getProtocol()	Returns a description of the protocol
String getServerName()	Returns the name of the server
Int getServerPort()	Returns the port number.

ServletResponse Interface:

PrinterWriter getWriter()	Returns a PrintWriter that can be used to write character data to the response
ServletOutputStream getOutputStream()	Returns a ServletOutputStream that can be used to write binary data to the response

Following are the classes and their methods

GenericServlet class:

This class implements Servlet and ServletConfig interfaces

ServletInputStream class:

The ServletInputStream class extends InputStream. It is implemented by the server and provides an input stream that a servlet developer can use to read the data from a client request. In addition to this, one more method is added which returns the actual number of bytes read

int readLine(byte[ ] buffer, int offset, int size)

ServletOutputStream class:

ServletOutputStream class extends OutputStream. It defines the print() and println() methods, which output data to the stream.

ServletException class:

This class indicates that a servlet problem has occurred. The class has the following constructor

ServletException( )
ServletException(String s)

Reading Servlet Parameters:

The following Servlet program shows the reading servlet parameters:

```
import java.io.*;
import javax.servlet.*;
Public class ParameterServlet extends GenericServlet
{
Public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
{
PrintWriter pw=res.getWriter;
Enumeration e = req.getParameterNames();
While(e.hasMoreElements())
{
String pname = (String)e.nextElement();
pw.print(pname + " =");
String pvalue=req.getParameter(pname);
pw.println(pvalue);
}
pw.close();
}
```

```
}
```

Reading Initialization Parameters:

The following Servlet program shows the reading initialization parameters:

```
import java.io.*;
import javax.servlet.*;
public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException
{
ServletConfig sc= getServletConfig();
res.setContentType("text/html");
PrintWriter pw=res.getWriter();
pw.println(" Name : "+ sc.getInitParameter("name"));
pw.close();
}
}
```