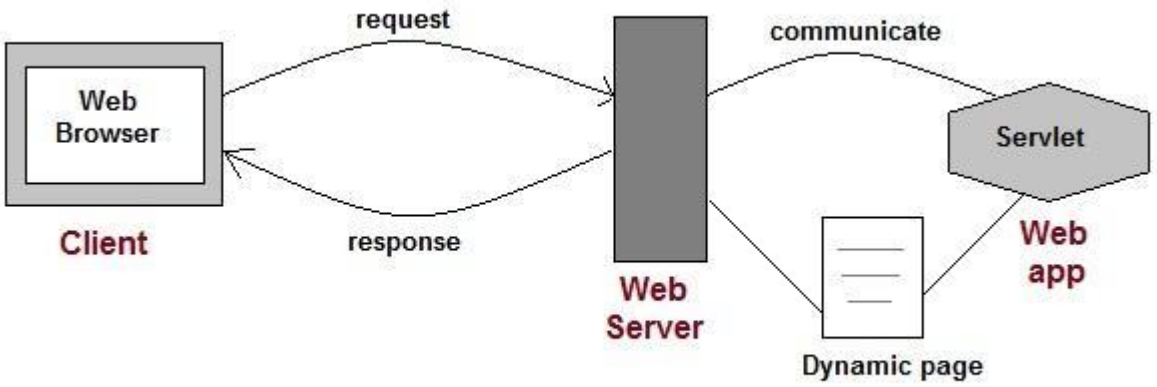
Introduction to Servlet

Servlet Technology is used to create web applications. **Servlet** technology uses Java language to create web applications.

Web applications are helper applications that resides at web server and build dynamic web pages. A dynamic page could be anything like a page that randomly chooses picture to display or even a page that displays the current time.

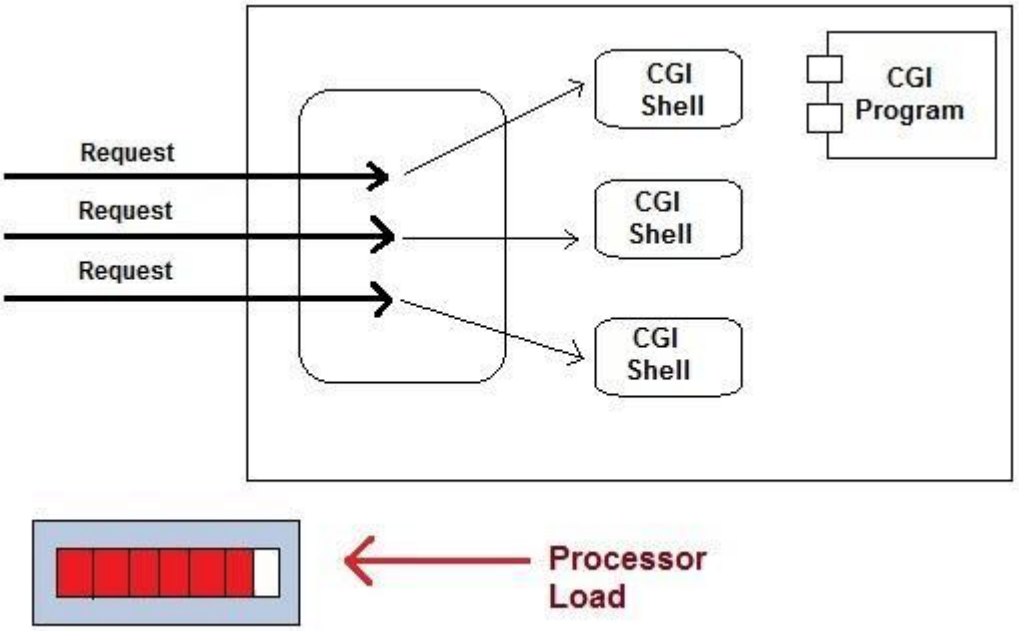


As Servlet Technology uses Java, web applications made using Servlet are **Secured**, **Scalable** and **Robust**.

CGI (Common Gateway Interface)

Before Servlets, CGI(Common Gateway Interface) programming was used to create web applications. Here's how a CGI program works :

- User clicks a link that has URL to a dynamic page instead of a static page.
- The URL decides which CGI program to execute.
- Web Servers run the CGI program in seperate OS shell. The shell includes OS enviroment and the process to execute code of the CGI program.
- The CGI response is sent back to the Web Server, which wraps the response in an HTTP response and send it back to the web browser.



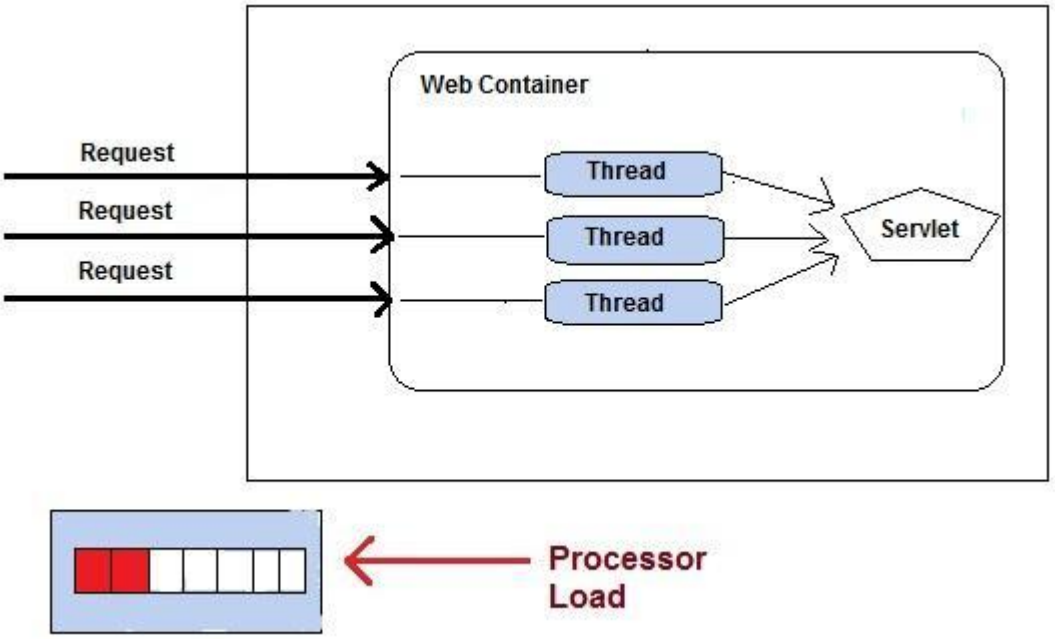
Drawbacks of CGI programs

- High resposne time because CGI programs execute in their own OS shell.
- CGI is not scalable.
- CGI programs are not always secure or object-oriented.
- It is Platform dependent.

Because of these disadvantages, developers started looking for better CGI solutions. And then Sun Microsystems developed **Servlet** as a solution over traditional CGI technology.

Advantages of using Servlets

- Less response time because each request runs in a separate thread.
- Servlets are scalable.
- Servlets are robust and object oriented.
- Servlets are platform independent.



Servlet API

Servlet API consists of two important packages that encapsulates all the important classes and interface, namely :

- **javax.servlet**
- **javax.servlet.http**

Some Important Classes and Interfaces of javax.servlet

INTERFACES	CLASSES
Servlet	ServletInputStream
ServletContext	ServletOutputStream
ServletConfig	ServletRequestWrapper
ServletRequest	ServletResponseWrapper
ServletResponse	ServletRequestEvent
ServletContextListener	ServletContextEvent
RequestDispatcher	ServletRequestAttributeEvent
SingleThreadModel	ServletContextAttributeEvent
Filter	ServletException
FilterConfig	UnavailableException
FilterChain	GenericServlet
ServletRequestListener	

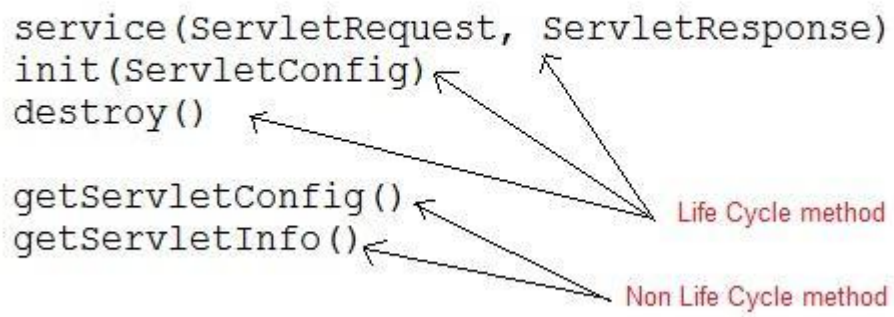
Some Important Classes and Interface of javax.servlet.http

CLASSES and INTERFACES

HttpServlet	HttpServletRequest
HttpServletResponse	HttpSessionAttributeListener
HttpSession	HttpSessionListener
Cookie	HttpSessionEvent

Servlet Interface

Servlet Interface provides five methods. Out of these five methods, three methods are **Servlet life cycle** methods and rest two are non life cycle methods.



GenericServlet Class

GenericServlet is an abstract class that provides implementation of most of the basic servlet methods. This is a very important class.

Methods of GenericServlet class

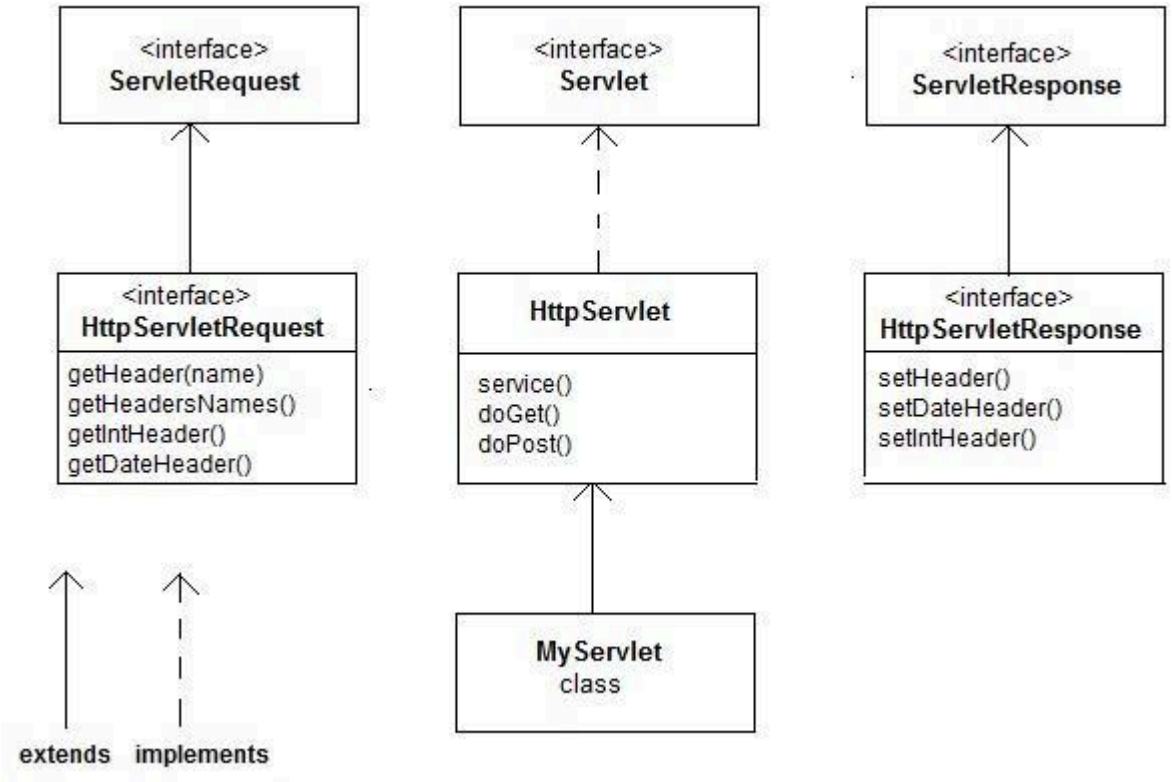
- `public void init(ServletConfig)`
- `public abstract void service(ServletRequest request,ServletResposne response)`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`
- `public ServletContext getServletContext()`
- `public String getInitParameter(String name)`
- `public Enumeration getInitParameterNames()`
- `public String getServletName()`
- `public void log(String msg)`
- `public void log(String msg, Throwable t)`

HttpServlet class

HttpServlet is also an abstract class. This class gives implementation of various `service()` methods of **Servlet** interface.

To create a servlet, we should create a class that extends **HttpServlet** abstract class. The Servlet class that we will create, must not override `service()` method. Our servlet class will override only the `doGet()` and/or `doPost()` methods.

The `service()` method of **HttpServlet** class listens to the Http methods (GET, POST etc) from request stream and invokes `doGet()` or `doPost()` methods based on Http Method type.



How a Servlet Application works

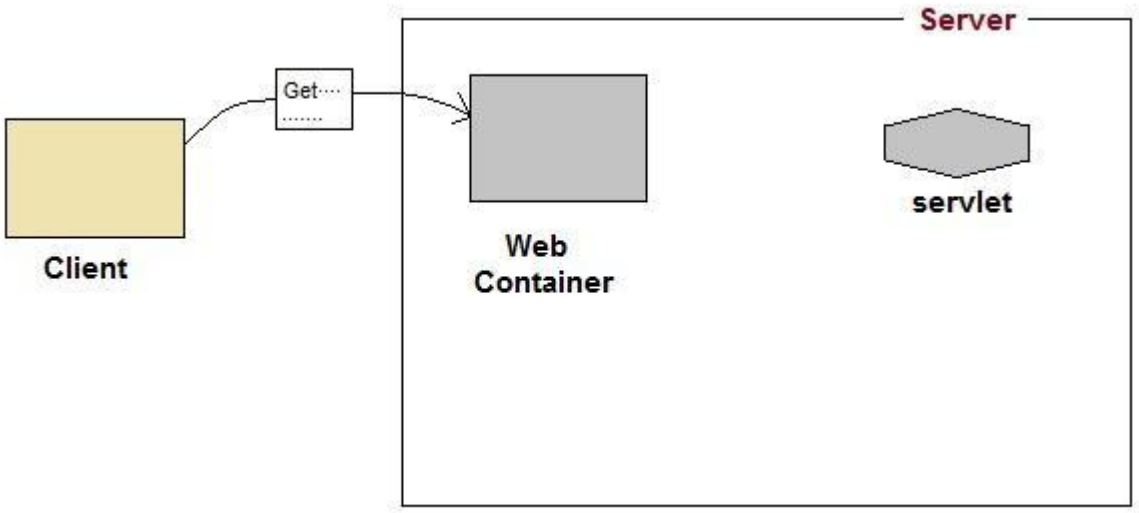
Web container is responsible for managing execution of servlets and JSP pages for Java EE application.

When a request comes in for a servlet, the server hands the request to the Web Container. **Web Container** is responsible for instantiating the servlet or creating a new thread to handle the request. Its the job of Web Container to get the request and response to the servlet. The container creates multiple threads to process multiple requests to a single servlet.

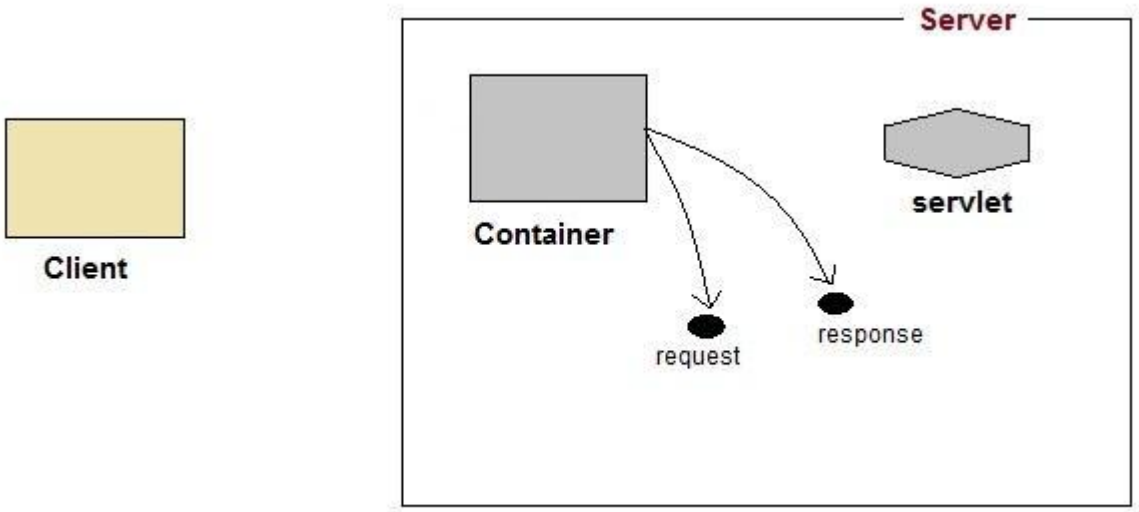
Servlets don't have a main() method. Web Container manages the life cycle of a Servlet instance.

Quick Revision on How a Servlet works

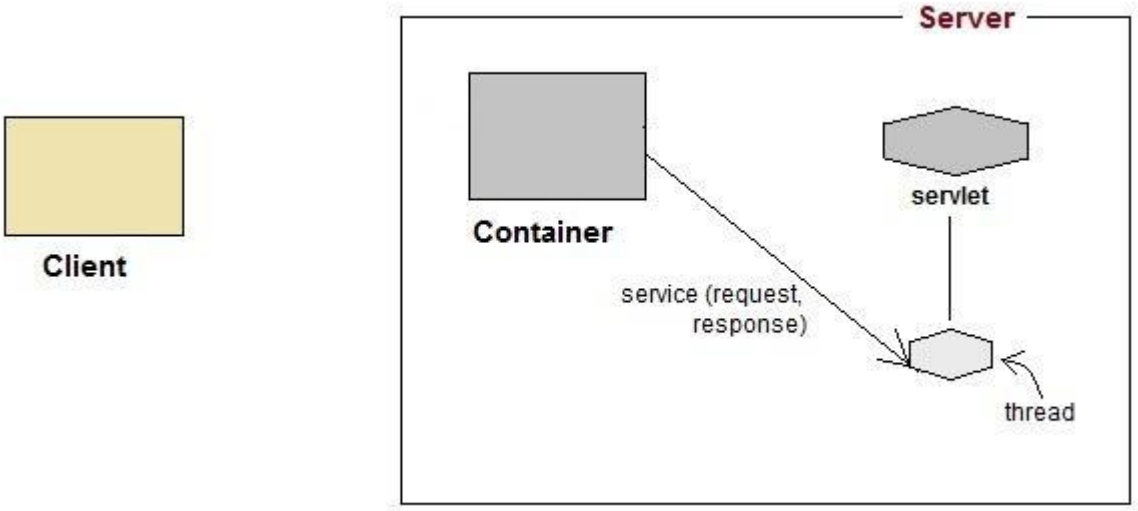
1. User sends request for a servlet by clicking a link that has URL to a servlet.



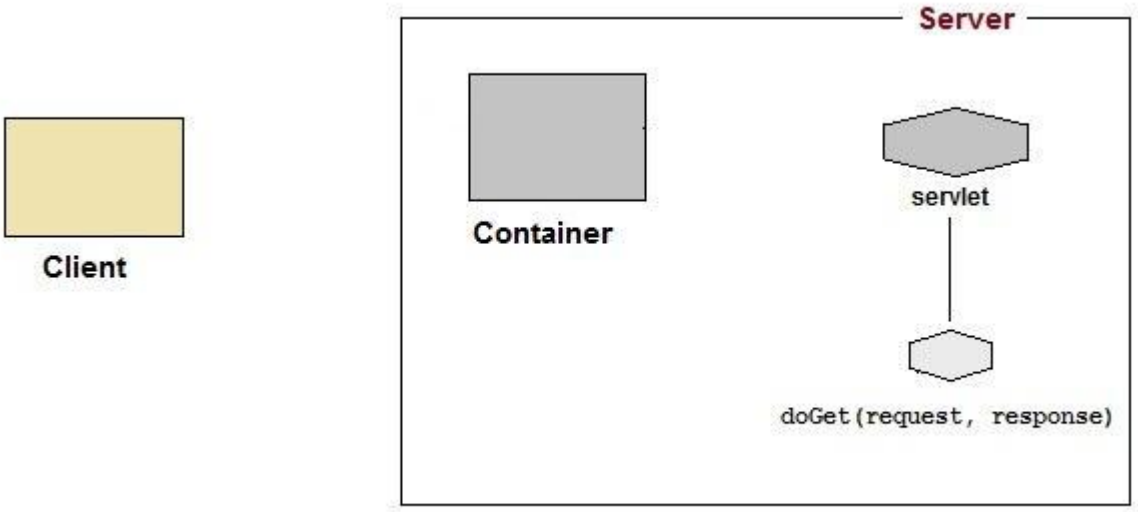
2. The container finds the servlet using **deployment descriptor** and creates two objects :
   a. **HttpServletRequest**
   b. **HttpServletResponse**



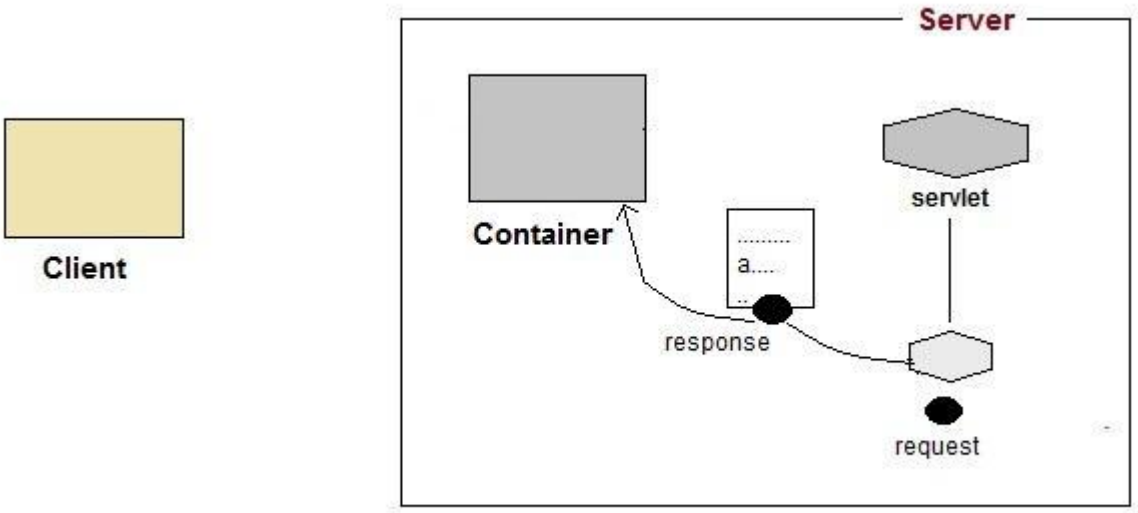
3. Then the container creates or allocates a thread for that request and calls the Servlet's `service()` method and passes the **request, response** objects as arguments.



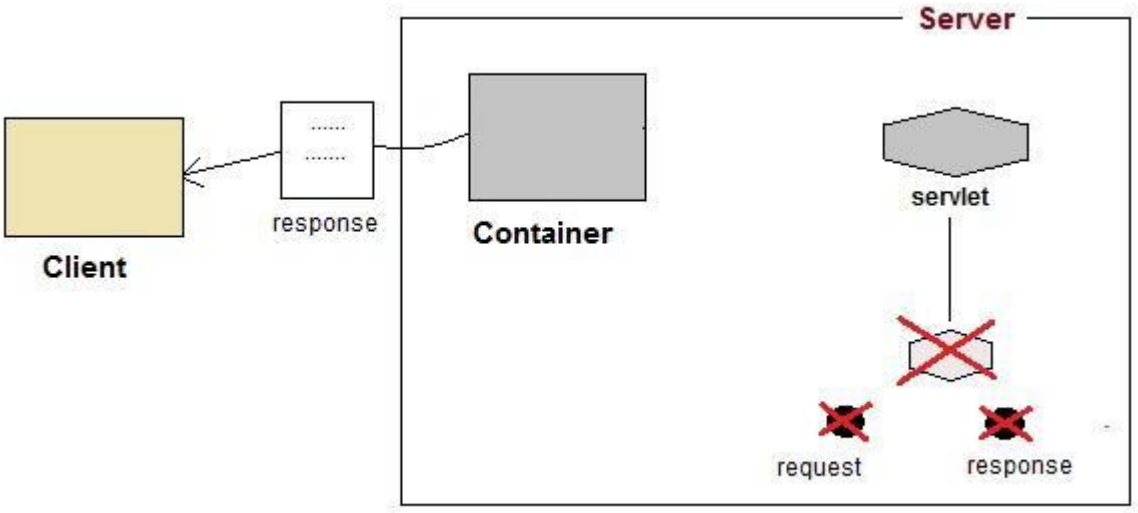
4. The `service()` method, then decides which servlet method, `doGet()` or `doPost()` to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the `service()` will call Servlet's `doGet()` method.



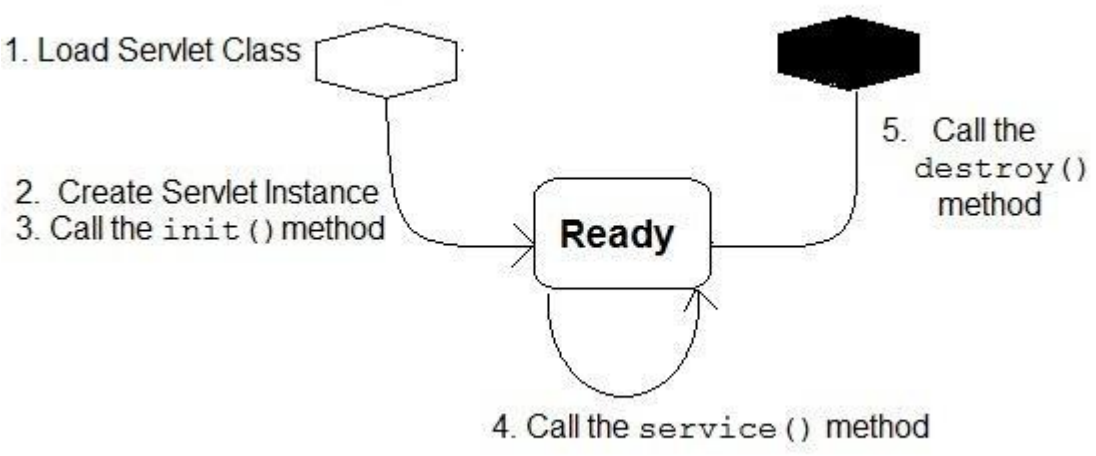
5. Then the Servlet uses response object to write the response back to the client.



6. After the `service()` method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.



Servlet Life Cycle



1. **Loading Servlet Class :** A Servlet class is loaded when first request for the servlet is received by the Web Container.
2. **Servlet instance creation :**After the Servlet class is loaded, Web Container creates the instance of it. Servlet instance is created only once in the life cycle.
3. **Call to the init() method :** `init()` method is called by the Web Container on servlet instance to initialize the servlet.

   Signature of init() method :

   ```
   public void init(ServletConfig config) throws ServletException
   ```

4. **Call to the service() method :** The containers call the `service()` method each time the request for servlet is received. The service() method will then call the `doGet()` or `doPost()` methos based ont eh type of the HTTP request, as explained in previous lessons.

   Signature of service() method :

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

5. **Call to destroy() method:** The Web Container call the `destroy()` method before removing servlet instance, giving it a chance for cleanup activity.

Steps to Create Servlet Application using tomcat server

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Download latest version of Tomcat Server and install it on your machine.

After installing Tomcat Server on your machine follow the below mentioned steps :

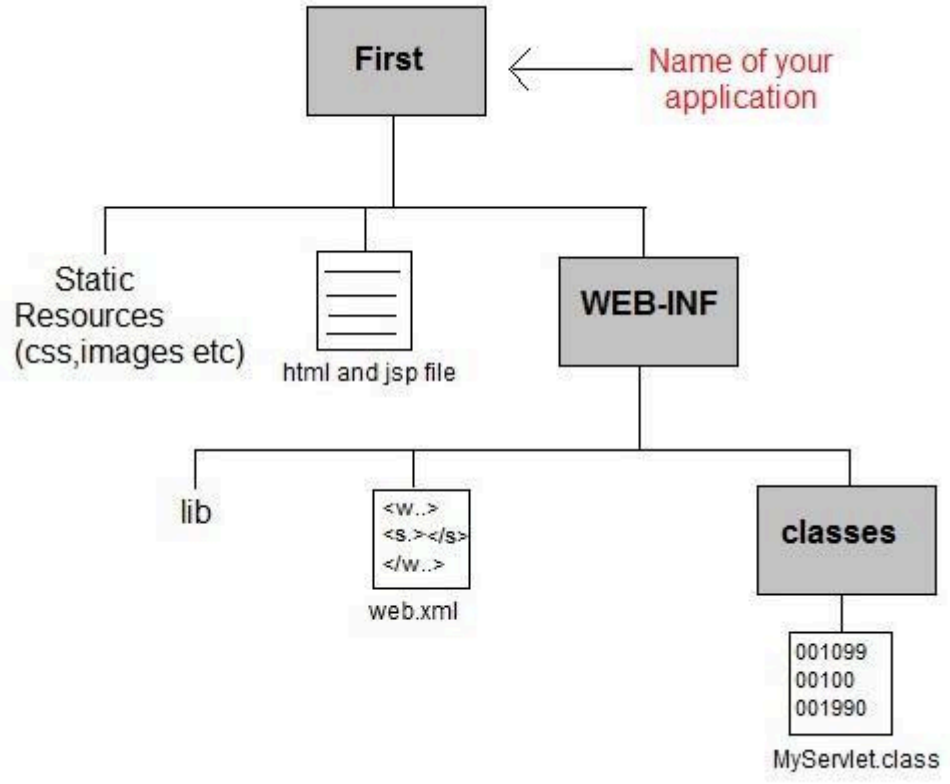
1. Create directory structure for your application.
2. Create a Servlet
3. Compile the Servlet
4. Create Deployment Descriptor for your application
5. Start the server and deploy the application

All these 5 steps are explained in details below, lets create our first Servlet Application.

---

1. Creating the Directory Structure

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside **Apache-Tomcat\webapps** directory. All HTML, static files(images, css etc) are kept directly under **Web application** folder. While all the Servlet classes are kept inside `classes` folder.

The `web.xml` (deployement descriptor) file is kept under `WEB-INF` folder.

Creating a Servlet

There are three different ways to create a servlet.

- By implementing **Servlet** interface
- By extending **GenericServlet** class
- By extending **HttpServlet** class

But mostly a servlet is created by extending **HttpServlet** abstract class. As discussed earlier **HttpServlet** gives the definition of `service()` method of the **Servlet** interface. The servlet class that we will create should not override `service()` method. Our servlet class will override only `doGet()` or `doPost()` method.

When a request comes in for the servlet, the Web Container calls the servlet's `service()` method and depending on the type of request the `service()` method calls either the `doGet()` or `doPost()` method.

NOTE: By default a request is **Get** request.

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

public MyServlet extends HttpServlet
{
 public void doGet(HttpServletRequest request,HttpServletResposne response)
                     throws ServletException
 {
   response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   out.println("<html><body>");
   out.println("<h1>Hello Readers</h1>");
   out.println("</body></html>");
 }
}
```

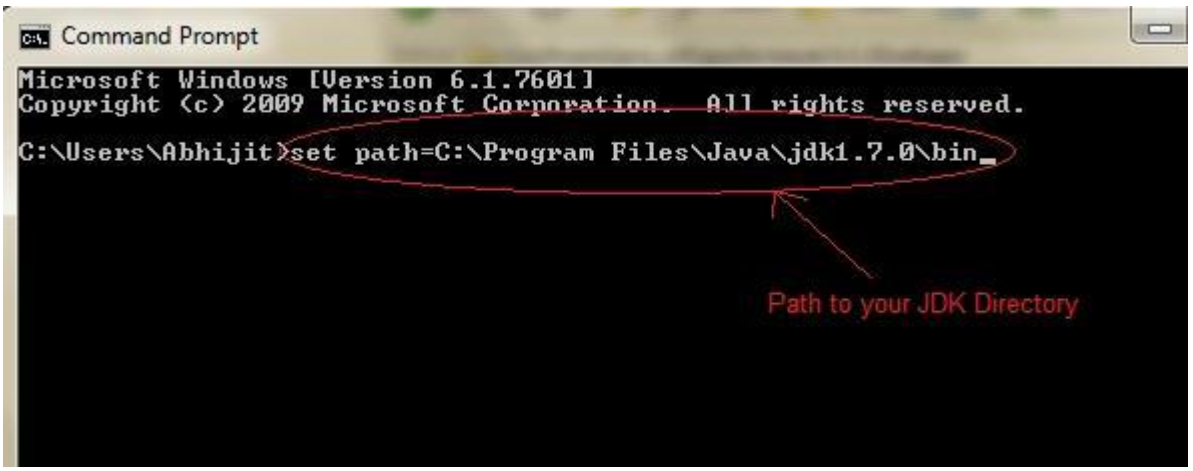
Write above code in a notepad file and save it as **MyServlet.java** anywhere on your PC. Compile it(explained in next step) from there and paste the class file into `WEB-INF/classes/` directory that you have to create inside **Tomcat/webapps** directory.

Compiling a Servlet

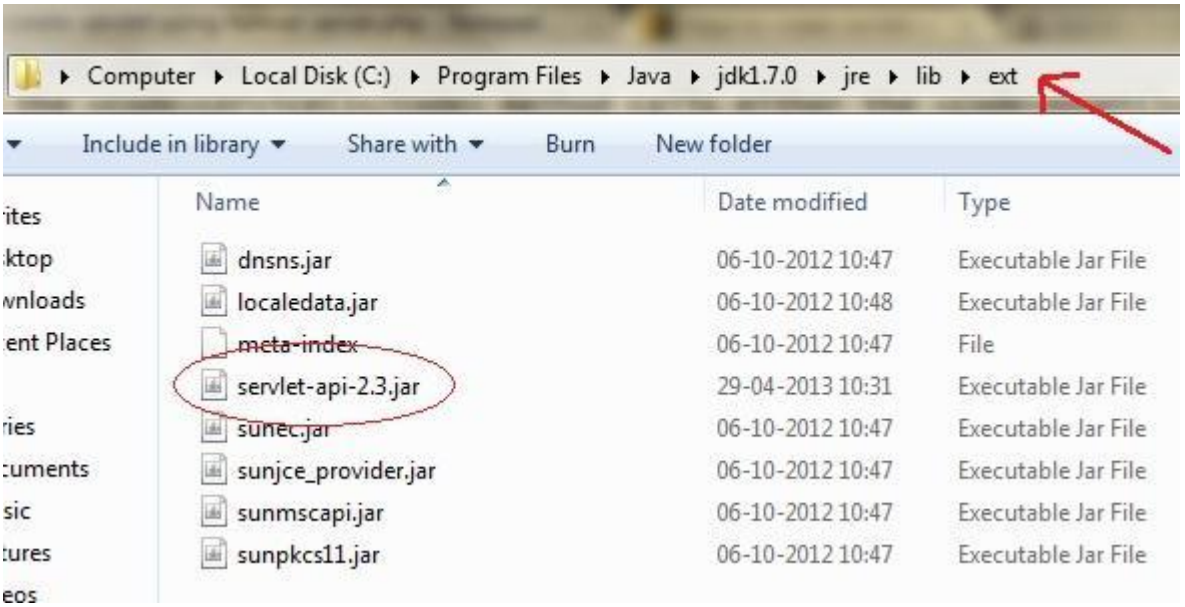
To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server `servlet-api.jar` file is required to compile a servlet class.

Steps to compile a Servlet

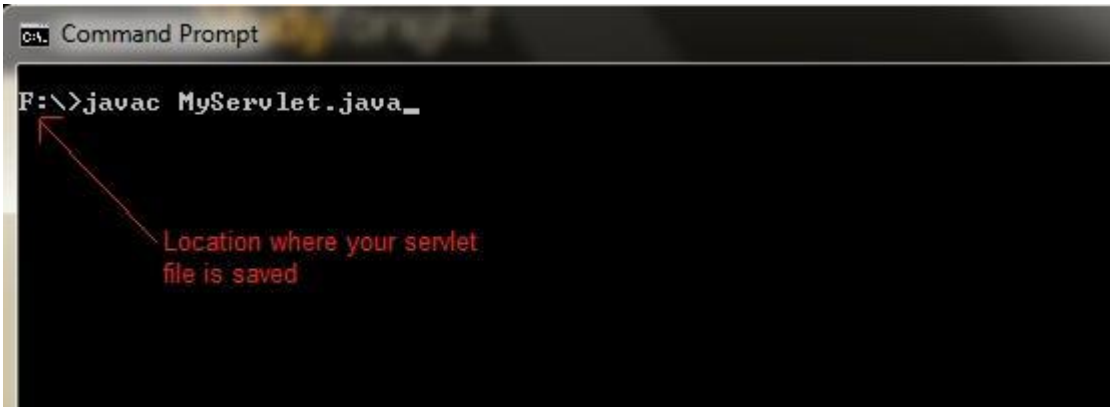
- Set the Class Path.



- Download **servlet-api.jar** file.
- Paste the servlet-api.jar file inside `Java\jdk\jre\lib\ext` directory.



- Compile the Servlet class.



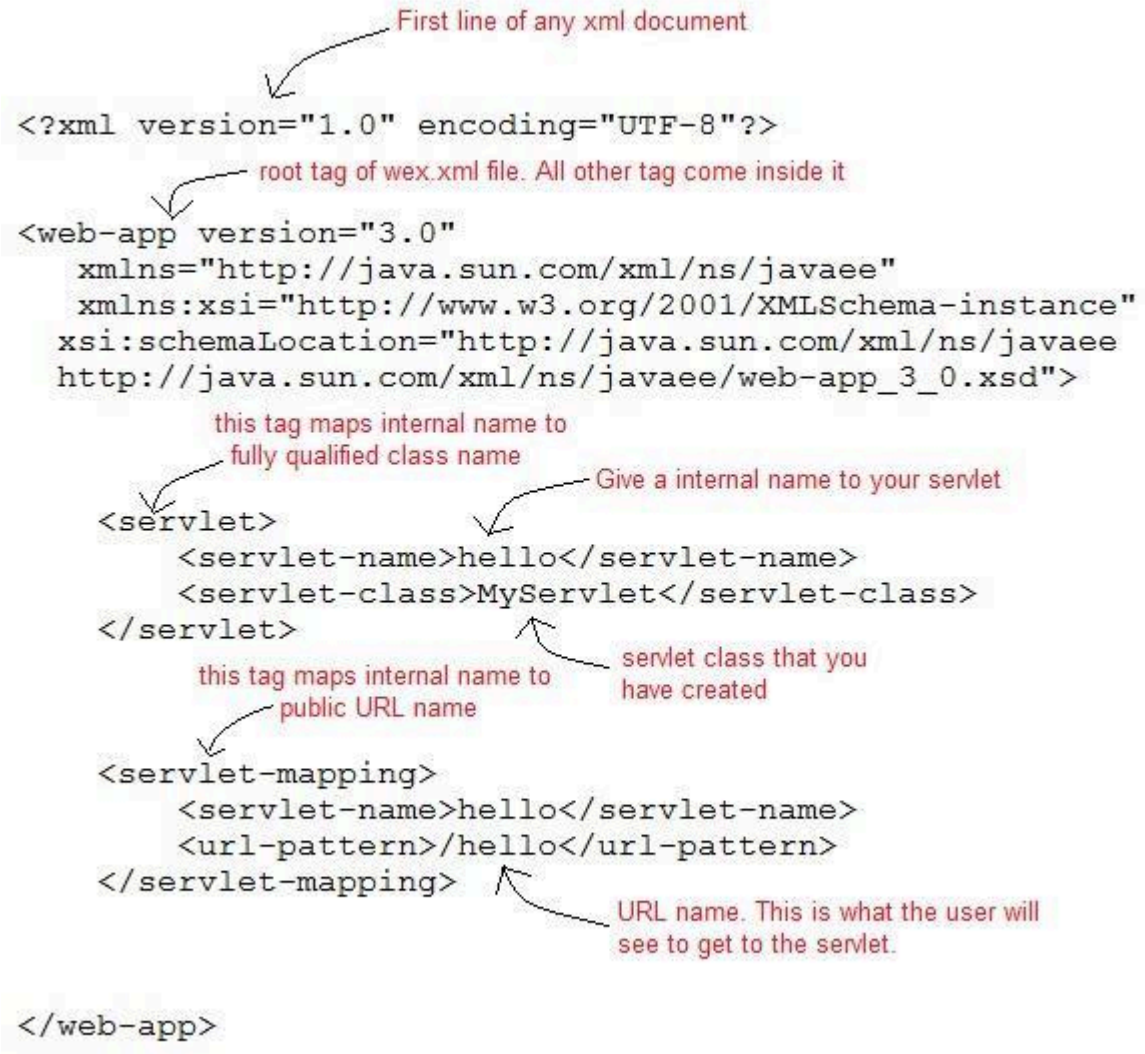
NOTE: After compiling your Servlet class you will have to paste the class file into `WEB-INF/classes/` directory.

Create Deployment Descriptor

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

- Mapping URL to Servlet class.
- Initializing parameters.
- Defining Error page.
- Security roles.
- Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple **web.xml** file for our web application.



Start the Server

Double click on the **startup.bat** file to start your Apache Tomcat Server.

Or, execute the following command on your windows machine using RUN prompt.

```
C:\apache-tomcat-7.0.14\bin\startup.bat
```

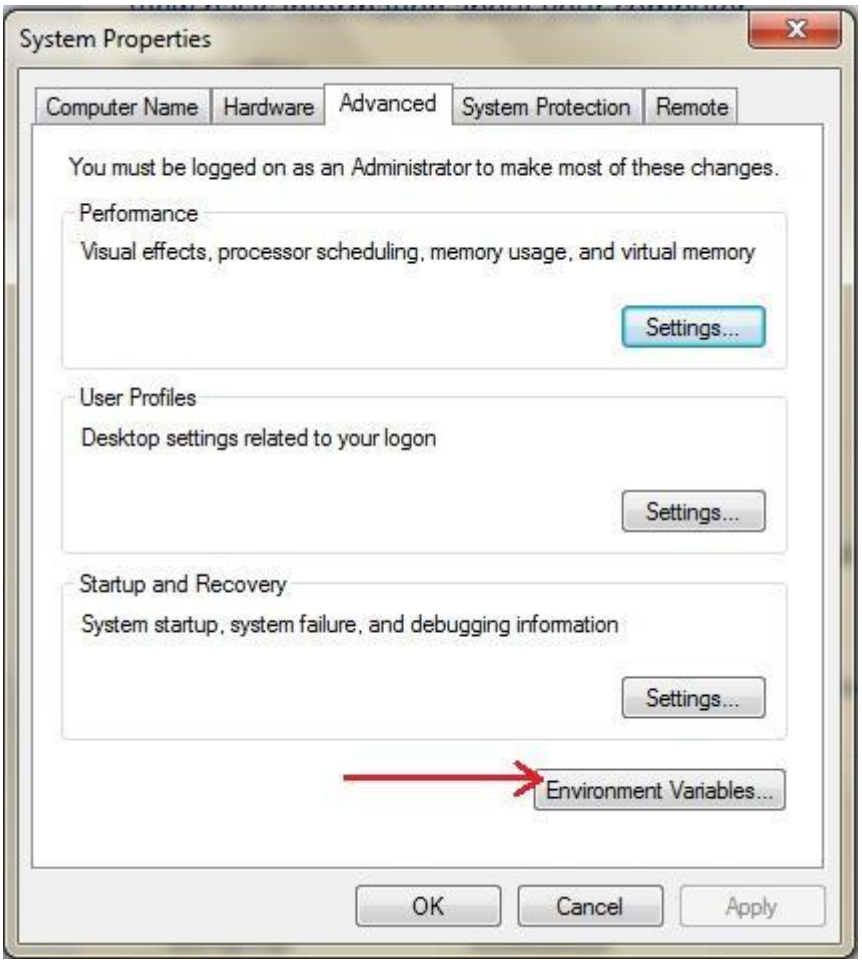
Starting Tomcat Server for the first time

If you are starting Tomcat Server for the first time you need to set JAVA_HOME in the Enviroment variable. The following steps will show you how to set it.

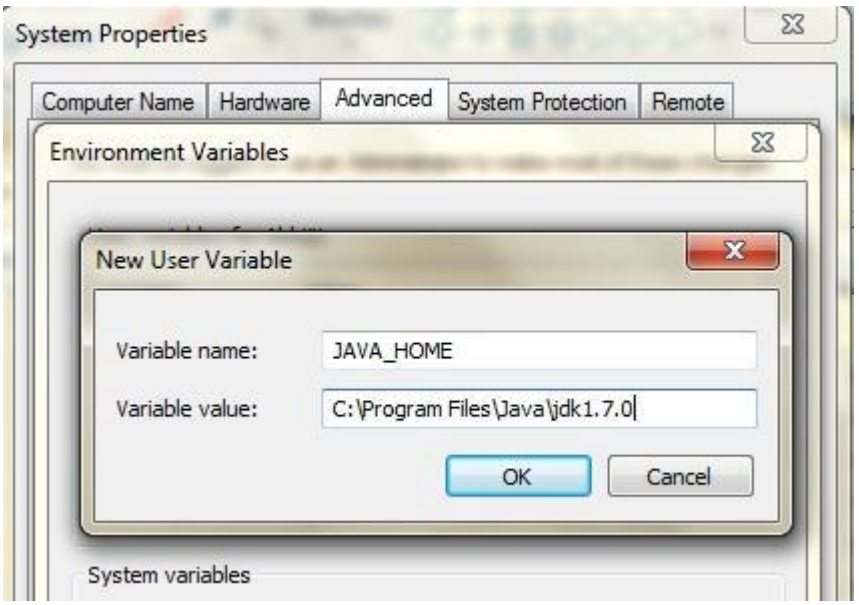
- Right Click on **My Computer**, go to **Properites**.



- Go to **Advanced** Tab and Click on **Enviroment Variables...** button.



- Click on **New** button, and enter **JAVA_HOME** inside Variable name text field and path of JDK inside Variable value text field. Click OK to save.



Run Servlet Application

Open Browser and type **http:localhost:8080/First/hello**

