**Cliente – Servidor**

**HTTP:**

El protocolo HTTP es el protocolo utilizado en Internet para realizar la “conversación” entre diferentes máquinas, denominadas cliente y servidor.

Sus siglas significan HyperText Transfer Protocol, y tiene como objetivo envolver el lenguaje HTML (HyperText Markup Language) y otras especificaciones para luego ser transportado hacia su destino.

**WEB SERVER:**

Representa al servidor que se encarga de procesar los pedidos de los distintos clientes. Tiene la capacidad de recibir y administrar pedidos, para determinar la respuesta a enviar.

Cuando decimos **cliente** nos referimos a un navegador web determinado.

En nuestro caso utilizamos el Servidor Apache Tomcat.

**Apache TOMCAT como WEB SERVER:**

<http://tomcat.apache.org/>

Les adjunto una guía para instalar e ingresar a la consola de Tomcat

https://www.youtube.com/watch?v=yunGR3hvxX4

**Navegador Web:**

Es el otro componente de la estructura CLIENTE-SERVIDOR.

* Representa al cliente que hace pedidos al servidor.
* Tiene la responsabilidad de comprender las respuestas provenientes del Servidor, interpretarlas y visualizarlas en pantalla de modo grafico.

**Responsabilidades de un browser o Navegador**

Es interpretar lo que me envia el Servidor. Por ejemplo creamos un HTML sencillo con un párrafo “Hola Mundo”.

Opciones:

1. Descargar Sublime. Opcion elegida.
2. Realizarlo via txt.

Pasos:

1. Crear un archivo html (sublime) o txt.
2. Colocar el siguiente codigo:

<!DOCTYPE html>

<html>

<head>

<title>Hola Mundo</title>

</head>

<body>

<p>Hola Mundo</p>

</body>

</html>

1. Lo abrimos con un navegador (recomendación Google Chrome).
2. Y vemos como el navegador interpreta el html a traves del protocolo HTTP lo que envia el Servidor (en este caso nuestra PC).

**Responsabilidades del Servidor**

Responderme un HTML de acuerdo a las solicitudes que realiza el browser.

BROWSER (usuario) > SERVIDOR (servlets) > BACKEND (bases de datos, otros servidores)

**Request:**

Solicitud o pedido que hace un cliente a un servidor. En ese pedido el cliente le envia datos al servidor.

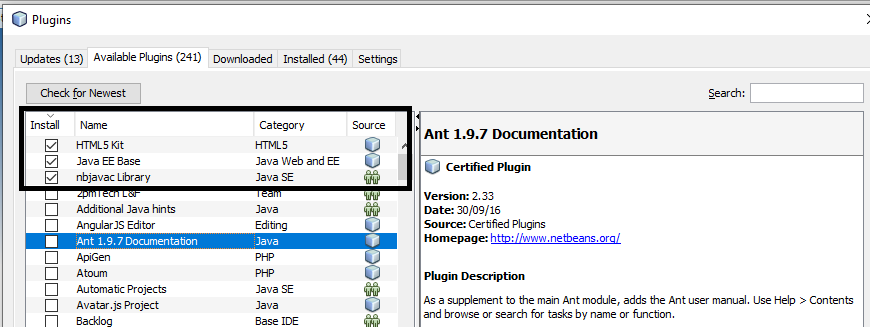
A partir de este pedido el servidor recibe los datos, los procesa y genera una respuesta.

**Response:**

En general retorna un codigo HTML. Aunque puede retornar diverso tipo de información como por ejemplo un XML, imágenes. O alguna otra función especifica. Por ej: enviar un mail, etc.

**Instalamos los plugins para crear un Proyecto Web (Neatbeans):**

1. Tools=> Plugins => Available Plugins.
2. Ordenamos por “nombre”.
3. Seleccionamos “HTML 5 kit”, “Java EE Base”, “nbjavac Library”.

****

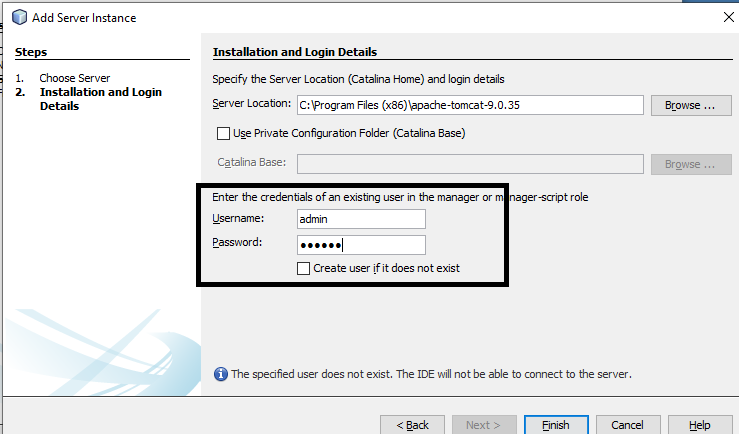
1. Install.
2. Restart IDE.

**Crear un proyecto Web (Netbeans)**

1. File.
2. New Project.
3. Java Web.
4. Web Application.
5. Colocamos un “Project\_Name” => Next
6. Server => Add=> Apache Tomcat or TomEE=> Next.
7. Server Location => Browse => colocamos la ruta de nuestro Tomcat. En mi caso es:

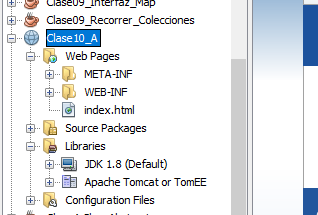
C:\Program Files (x86)\apache-tomcat-9.0.35

1. En Credenciales ingresen el usuario y password de Tomcat (user admin)



1. Finish.

**Proyecto Creado**



**Posibles errores**

**Desde Netbeans**

**Setear el Proxy:**

Setear en Tools => Option => No proxy

**Setear las credenciales de tomcat en tomcat**

Setear en el archivo tomcat-users.xml de conf dentro de Tomcat.

**Setear las credenciales de tomcat en Netbeans**

Ir a Windows => Services => Servers => seleccionar apache tomcat e ingresarle el usuario y contraseña desde “connection”.

**Cambiar el browser por defecto**

=> Tools > Options > General> Browser: Google Chrome

**SERVLETS**

Que es un Servlet?

* Es parte de la tecnología Java junto con JSP, y pertenece a la Edicion Empresarial (JEE).
* Un servlet es un objeto java que se ejecuta en un servidor (contenedor J2EE), especialmente diseñado para ofrecer contenido dinamico desde un servidor Web generalmente HTML.
* Es una unidad de funcionalidad que se ejecuta del lado del servidor, y genera resultados que son enviados al cliente.

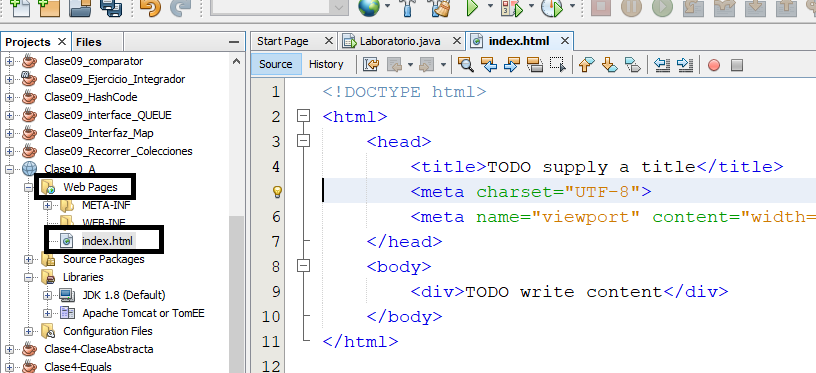
Hace de **intermediario** entre la **capa web** y **la capa de acceso a la BDD**.

* Un Servlet debe ser desplegado dentro de un Servlet Container para su correcto funcionamiento. No puedo colocarlo en cualquier servidor, sino en un servidor que pueda procesar codigo Java.

Por ejemplo paginas como Amazon.com esta llena de links y formularios que lo que hacen son pedidos al servidor. Tanto los links como los formularios utilizan peticiones que van a través de distintos métodos. Ej: los **links** utilizan el método **GET** y los **formularios** pueden utilizar tanto **GET** como **POST** para enviarse.

Lo que vamos a hacer ahora es crear una pagina HTML pero dentro de Netbeans para poder llamar un SERVLET que vamos a crear. Es decir vamos a conectar el mundo de HTML con el mundo de Java WEB.

Para comenzar vamos a crear primero una pagina de HTML:

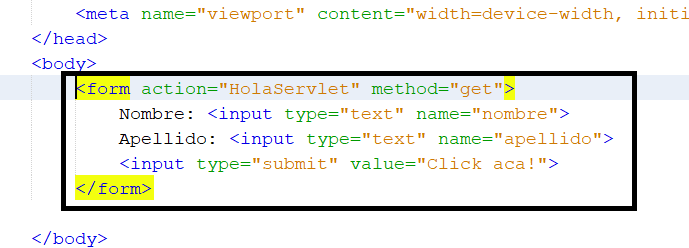


Luego vamos a insertar un formulario desde nuestro HTML para poder enviar datos a nuestro SERVLET de JAVA.

**Ejercicio A: Enviamos la info de un formulario a un SERVLET.**

**Codigo:** Clase10\_A

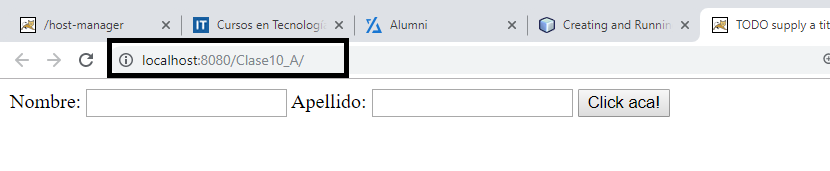
1. Nos vamos a Web pages.
2. Index.html
3. Abrimos el Index.html.
4. Creamos un formulario:



El formulario va a tener checkbox, radiobuttons, botones, listas, campos, es decir todo lo que el HTML puede contener y ser enviado a un SERVLET.

Los dos métodos principales son get y post. En este caso usamos **get** y la información va a viajar en la barra que tienen los browser arriba. En cambio si usamos el método **post** el envio se realizara via headers (cabeceras) en el cuerpo del request. En **action** del htmldebo colocar el **serlvet** de java que va a recibir toda esta información.

1. Ejecutamos el proyecto. Vemos que se levanta en el servidor de Tomcat.

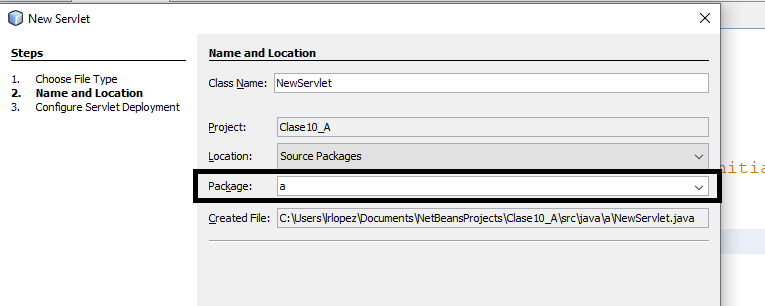


Esto se puede abrir desde cualquier browser copiando la url.

Esta pagina esta generando un pedido (request) a un servidor.

**EJEMPLO DE CREACION DE SERVLET.**

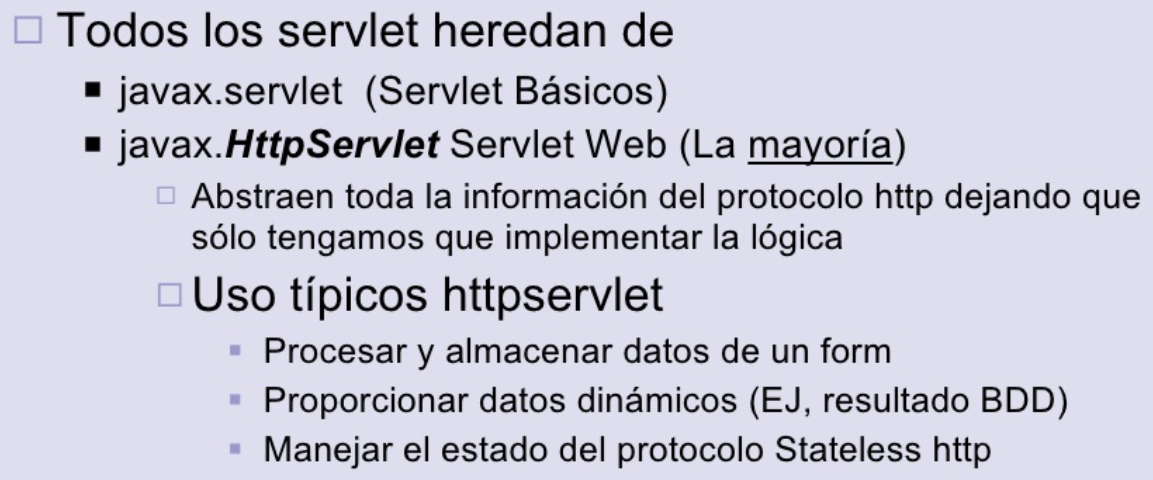
1. Source Packages => New => Servlet
2. Importante: Los servlet tienen que tener un paquete. Le ponemos cualquier nombre.



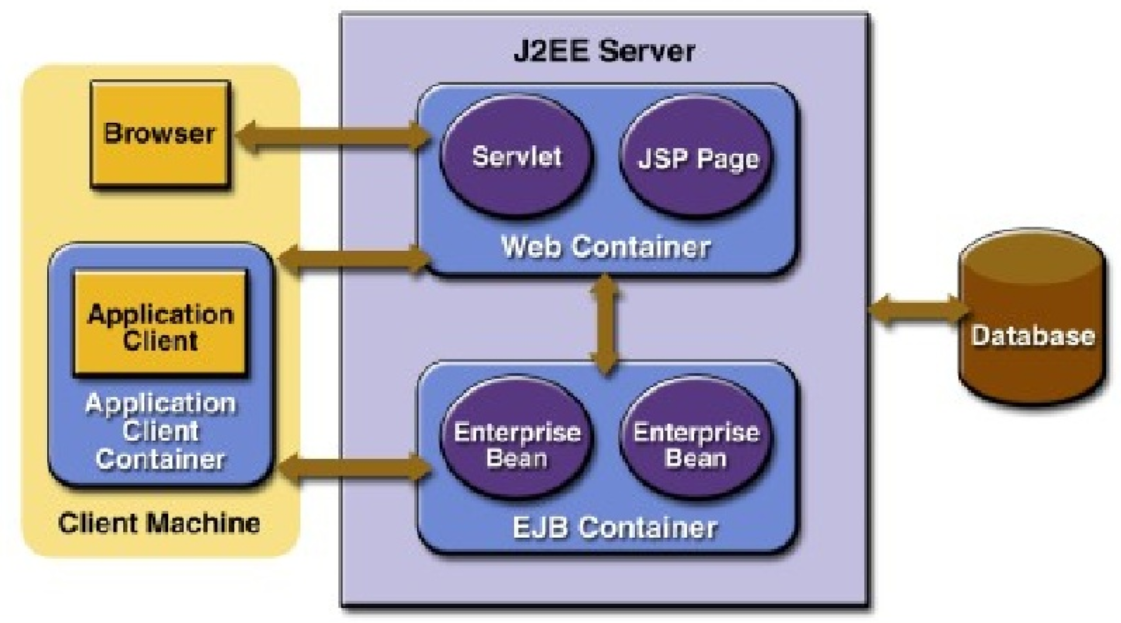
1. ClassName: HolaServlet. Igual a como lo habíamos llamado.
2. FINISH.

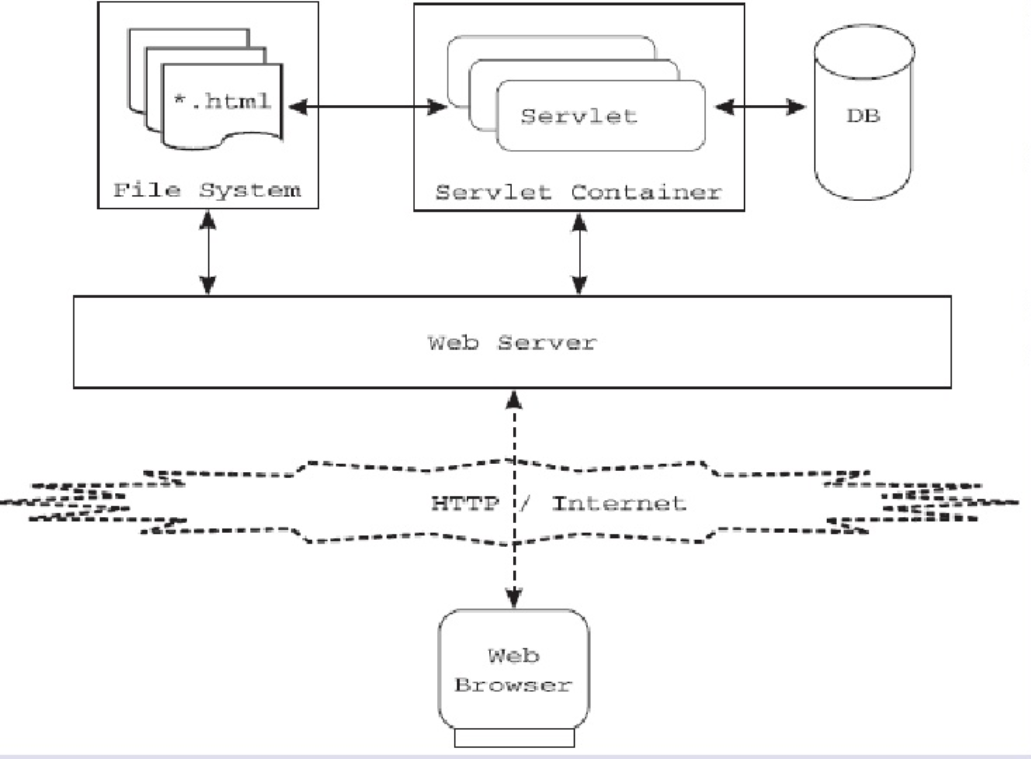
#### 1.1.2. Arquitectura del paquete servlet

Dentro del paquete **javax.servlet** tenemos toda la infraestructura para poder trabajar con servlets. El elemento central es la interfaz **Servlet**, que define los métodos para cualquier servlet. La clase **GenericServlet**es una clase abstracta que implementa dicha interfaz para un servlet genérico, independiente del protocolo. Para definir un servlet que se utilice vía web, se tiene la clase **HttpServlet** dentro del subpaquete **javax.servlet.http**. Esta clase hereda de *GenericServlet*, y también es una clase abstracta, de la que heredaremos para construir los servlets para nuestras aplicaciones web.



SERVLET Y CONTENEDORES





1. En el método doGet (ya que estamos usando este método) colocamos el siguiente codigo:

//Ejercicio 1

response.getWriter().append("Hola Mundo"); //append: concatena el string al printWriter

//getWriter: retorna un printwriter el

//cual podra enviar una cadena de caracteres al cliente

Esto lo que hace es que cuando oprimimos el botón desde nuestra pagina html, el servlet devuelve un “Hola Mundo”.

**Ejercicio 2**: **Objetivo**: Que el nombre y apellido que ingresa el usuario desde el html sea lo que salga por pantalla, es decir lo que devuelva el Servlet.

**Codigo**: Mismo codigo. Clase10\_A



Para hacer esto debemos prestar atención a los objetos que recibe como parámetro el método doGet=> Esto es el **request** que va a tener todos los datos del usuario y el **response** el cual me va a permitir responder desde el servidor. Son objetos de tipo interfaz.

1. Hago una modificación al doGet por lo siguiente:

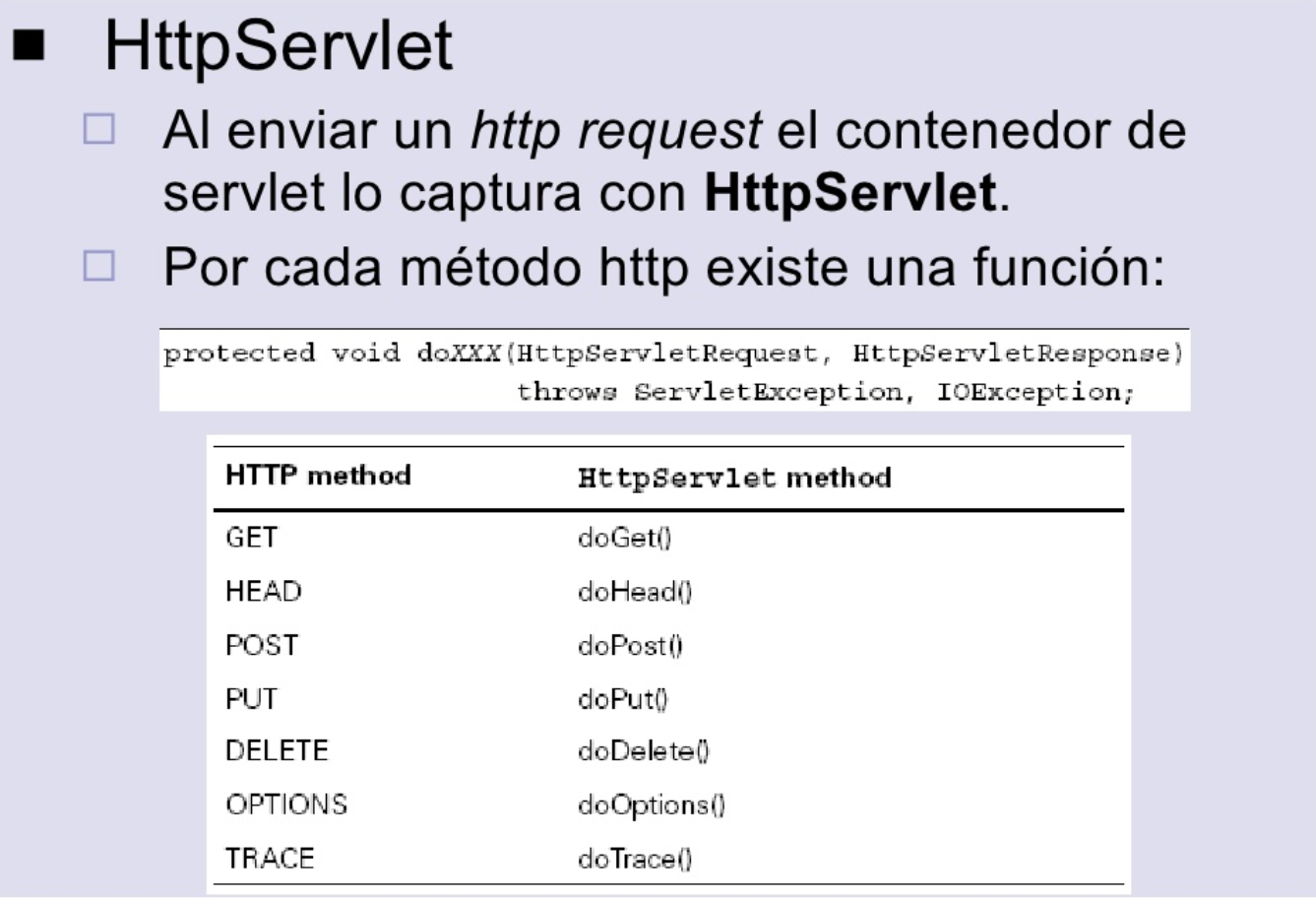
//Ejercicio 2

String nombre = request.getParameter("nombre"); //obtiene el dato del nombre en html

String apellido = request.getParameter("apellido"); //idem apellido

response.getWriter().append(nombre + " " +apellido); //el servlet devuelve el nombre y apellido

El getWriter() es un método de la interface ServletResponse y devuelve un objeto del tipo PrintWriter en donde contiene toda la text-data (character data). Entonces básicamente responde con el texto del nombre y apellido.



**Ejercicio 3: Ejercicio de la clase 10**

Consignas:



Codigo: **Clase10\_B**



1. Construimos el Form en index.html.

Incluimos el siguiente codigo en el Body:

<h1>Formulario</h1>

<form action="PersonaServlet" method="post">

<label>Nombre:<br> <input type="text" name="nombre"></label> <br>

<label>Apellido: <br> <input type="text" name="apellido"></label><br>

<label>Domicilio:<br> <input type="text" name="domicilio"></label><br>

<label>Pais: <br> <input type="text" name="pais"></label><br><br>

<input type="submit" name="boton" value="Registrarme!">

</form>

1. Agrego en el método **doGet** lo siguiente:

String respuesta = request.getParameter("nombre") + "\n" + request.getParameter("apellido") + "\n"

+ request.getParameter("domicilio") + "\n" + request.getParameter("pais");

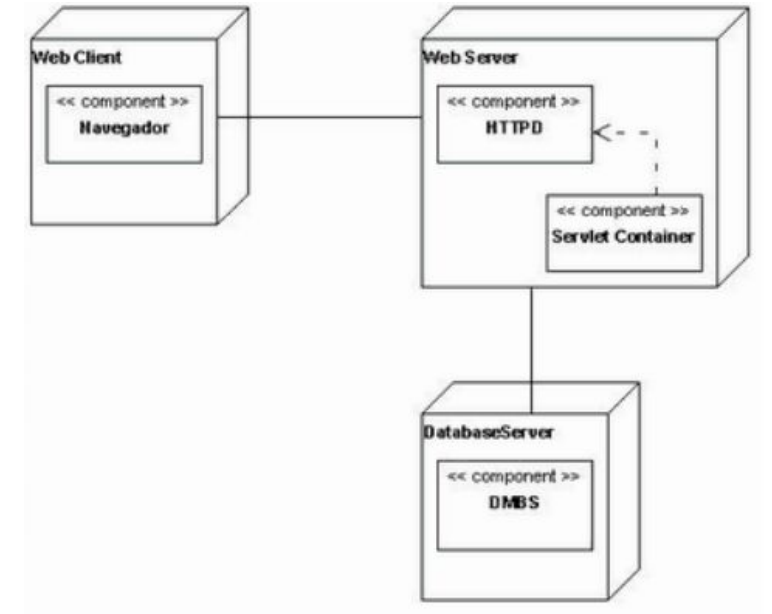
response.getWriter().append(respuesta);

1. Cerciorarse que en el metodo **doPost** tengamos lo siguiente (el metodo doPost esta llamando a doGet):

doGet(request, response);

**Observaciones**:

Cabe destacar que si quiero registrar los datos en la BDD puedo hacer referencia a **PersonaDAO** creando un objeto de tipo **Persona**. Eso lo vamos a ver en la próxima clase.



El método **doPost** procesa request de tipo **post**. Solo que en el caso previo la inteligencia la derivo al método **get**.

**Ejercicio 4:** El objetivo es cuando al presionar el botón el servidor nos devuelva el siguiente mensaje de salida.

# Bienvenido a Java EE.

# Hoy es (agrego fecha y hora de hoy)

# y ha creado su primer servlet

**Codigo: Clase10\_C**

****

1. Ingreso el siguiente botón en mi HTML (index).

<body>

<form action="Servlet" method="get">

<input type="submit">

</form>

</body>

1. Crear un servlet llamado Servlet y en el método **processRequest** ingresar el siguiente codigo a la altura del BODY:

out.println("<h1>Bienvenido a Java EE. Hoy es "+Calendar.getInstance().getTime()+ "y ha "

+ "creado su primer servlet</h1>");

**Ejercicio 5: Objetivo:** Ver cada uno de los componentes HTML para conocer los elementos mas importantes en dicho lenguaje. Ver el siguiente codigo.

Codigo: **Clase10\_D**.



1. Plantear el index.html de acuerdo al codigo previo.

**Nota:** <input type=”hidden”>

El elemento input, teniendo el valor "hidden" en su atributo type, representa cualquier cadena de texto arbitraria que no está pensada para ser vista o editada por el usuario.

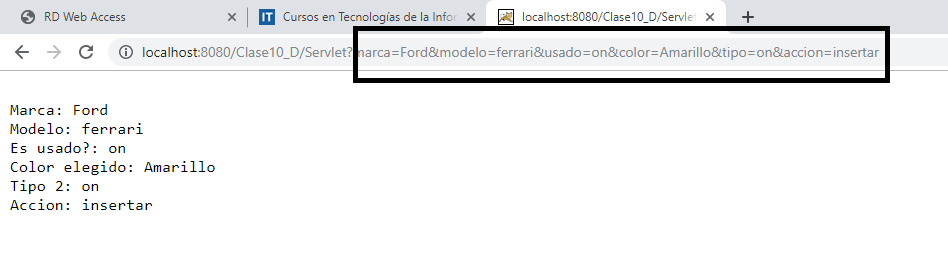
**Metodo GET:** Envia la información del usuario adjuntada al request. La URL y la información codificada es separada con el carácter”**?”**.

Esta información es enviada utilizando el mencionado **QUERY\_STRING header** y será accesible a través del QUERY STRING variable de entorno a través de los métodos **getQueryString**() y **getParameter**() del **objeto** **request**.

**Metodo POST:** Mas confiable que el GET para el envio de información. La información es enviada a través de un **mensaje.**

**JSP** maneja este tipo de **requests** a través del **getParameter()** para leer parámetros simples y **getInputStream()** para leer data binary stream que viene del cliente.

**Querystring**: En el ejercicio vemos que todo lo que esta después del ? es lo que enviamos via método Get con el formulario. Lo enviamos desde el cliente al servidor. Está compuesta de pares de nombre/valor separados por un "&".



Mismo ejercicio usando **POST**

Modificamos el index.html (method=POST) y en el Servlet en el método doPost() borramos la llamada al método **processRequest()** y llamamos al método **doGet().** Para que la inteligencia de respuesta continue estando allí.

La diferencia que vemos ahora es que la información va inmersa en el cuerpo del **request. No figura mas en el Address bar.**

**Web.xml: Descriptor de Despliegue.**

Esta localizado en la ruta YOURPROJECT\web\WEB-INF.

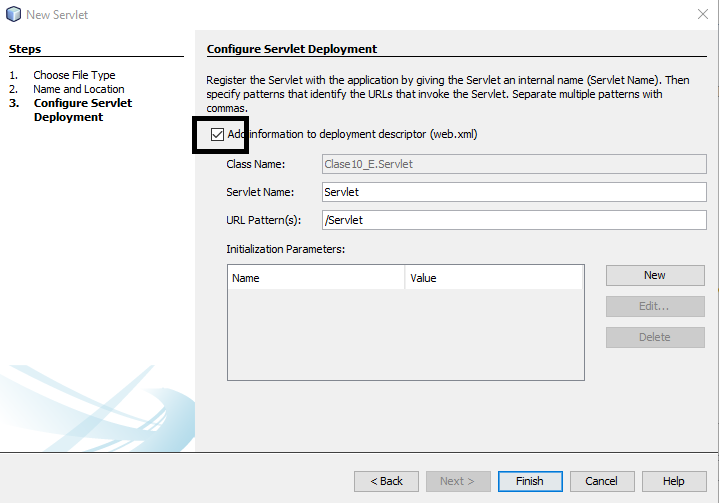
Es opcional a partir de Java EE 6. Por default no es cargado en Netbeans. Hay que crearlo en caso de necesitarlo.

Creacion:

New -> Other-> Web -> Standard Deployment Descriptor (web.xml) -> Next -> Finish

El archivo web.xml se denomina descriptor de despliegue y es opcional cuando se utilizan servlet. Con él se define su nombre y se identifica al paquete y clase que lo contiene, entre otras cuestiones. Tambien tiene otros usos, tales como los filtros, los parametros de contexto, los parametros iniciales, etc…

También se puede crear cuando se crea un Servlet:

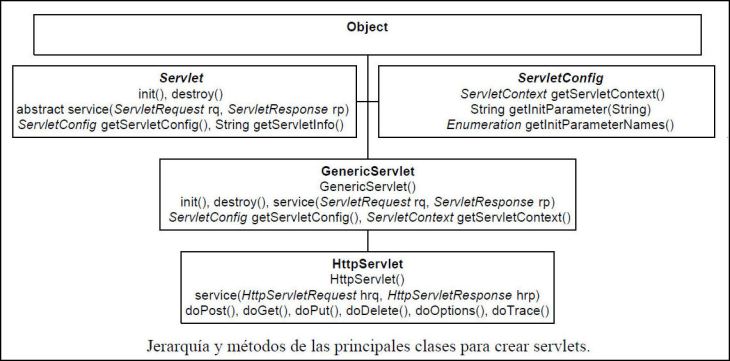


**Ciclo de Vida del Servlet:**

**El API Servlets y el ciclo de vida.**

Los servlets usan clases e interfaces de dos paquetes:

1. javax.servlet que contiene clases para servlets genéricos (independientes del protocolo que usen) y,
2. **javax.servlet.http** (que añade funcionalidad particular de http). El nombre javax indica que los servlets son una extensión.

[](https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/image1-53/)

Los servlets no tienen el método main() como los programas Java, sino que se invocan unos métodos cuando se reciben peticiones. A esta metodología se le llama ciclo de vida de un servlet y viene dado por tres métodos: **init**, **service**, **destroy**:

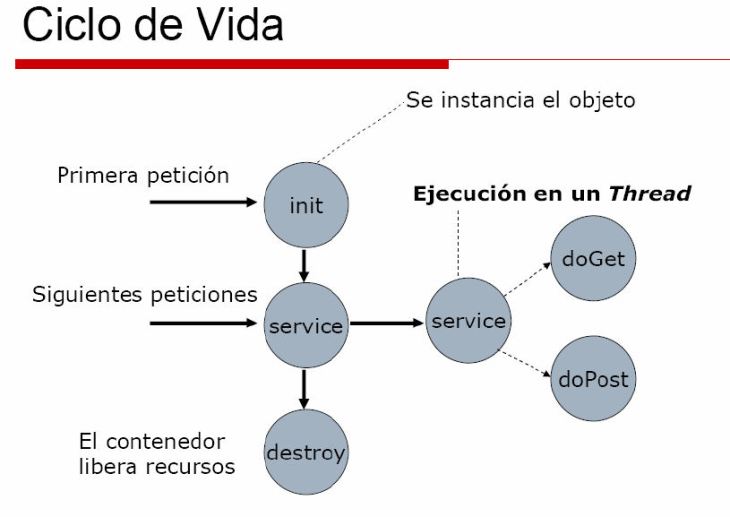
**INICIALIZACIÓN**: Una única llamada al método “init” por parte del servlet. Incluso se pueden recoger unos parámetros concretos con “getInitParameter” de “ServletConfig” iniciales y que operarán a lo largo de toda la vida del servlet.

**init(ServletConfig config)**: Es el método utilizado para crear una nueva instancia del servlet (análogo al constructor). [Ver el ciclo de vida](https://users.dcc.uchile.cl/~jbarrios/servlets/vida.html). Este método puede ser sobreescrito para realizar tareas como crear una conexión a una BD que se mantendrá mientras el servlet se mantenga cargado y puede ser utilizada por cada petición. **ServletConfig** contiene los parámetros de inicialización que entrega el servidor al servlet.

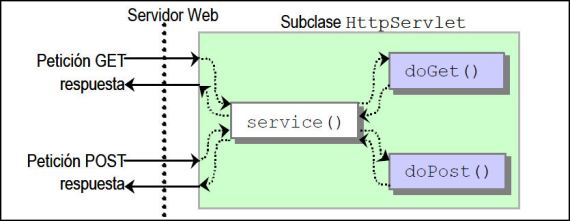
**SERVICIO**: una llamada a service() por cada invocación al servlet para procesar las peticiones de los clientes web.

**service(ServletRequest req, ServletResponse res)**: Este método es el que se llama cuando se recibe una petición de un cliente y en su implementación normal para HTTP verifica el tipo de solicitud GET, POST, etc. y la redirige a los métodos respectivos. En general no es necesario reimplementar este método.

**DESTRUCCIÓN**: Cuando todas las llamadas desde el cliente cesen o un temporizador del servidor así lo indique o el propio administrador así lo decida se destruye el servlet. Se usa el método “destroy” para eliminar al servlet y para “recoger sus restos” (garbage collection).

[](https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/image2-45/)

Cada vez que el servidor pasa una petición (distinta a la primera) a un servlet se invoca el método service(), este método habrá que sobreescribirlo (override). Este método acepta dos parámetros: un objeto petición (request) y un objeto respuesta. Los servlets http, que son los que vamos a usar, tienen ya definido un método service() que llama a doXxx(), con Xxx el nombre de la orden que viene en la petición al servidor web. Estos dos métodos son doGet() y doPost() y nos sirven para atender las peticiones específicamente provinientes de métodos GET o POST respectivamente,

[](https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/image1-51/)

De esta manera, una vez se carga el servlet, es muy eficiente, pues sólo hay una copia cargada en memoria (se ejecutan uno o varios hilos), no hay que crear nuevos objetos (un solo objeto servlet), y tiene persistencia: puede guardar información entre peticiones, como contadores o conexiones a una base de datos. Esto último puede ser muchísimo más eficiente hacerlo en el método init que abrir y cerrar la conexión con la base de datos en cada petición.

**Ejercicio 6:**

Usamos el mismo formulario del ejercicio anterior y utilizamos el método **processRequest** para ver como se arma un html de respuesta. Incluimos los datos de respuesta y la lógica en el mismo.



**Fuentes de Informacion (No sustituyen sino complementan al Alumni):**

<https://flanagan.ugr.es/docencia/2005-2006/2/servlets/ejemplos/Alumno.java>

<https://jtagua.wordpress.com/2010/10/31/tutorial-de-servlet-1-introduccion-ciclo-de-vida-y-ejemplo-basico/>

<https://users.dcc.uchile.cl/~jbarrios/servlets/clases.html>