En la mayoría de los servidores web los ***servlets*** son accedidos mediante un ***URL*** que comienza por ***/servlet/.*** El siguiente método ***HTTP GET*** solicita el servicio del ***servlet MiServlet*** al servidor ***miServidor.com,*** con lo cual petición ***GET*** tiene la siguiente forma (en **negrita** contenido de la petición):

GET/serv1et/MiServ1et**?nombre=Anton&Ape11ido=Perez%2Ode%2ORomera TTP/1 .1**

Connection: Keep-Alive
User-Agent: Mozilla/4.5 (
compatible;
MSIE 4.01;
Windows NT)
Host: miServidor.com
Accept: image/gif, image/x-bitmap, image/Jpeg, image/pjpeg

El ***URL*** de esta petición ***GET*** llama a un ***servlet*** llamado ***MiServlet,*** contiene dos parámetros, ***nombre*** y ***apellido.*** Cada parámetro es un par que sigue el formato ***clave=valor.*** Lo parámetros se especifican poniendo un ***signo de interrogación*** (?) tras el nombre del ***servlet.*** Los distintos parámetros están separados entre sí por el símbolo ***ampersand (&).***

Obsérvese que la secuencia de caracteres ***%20*** aparece dos veces en el apellido. Es una forma de decir que hay un ***espacio*** entre "Lopez y de", y otro entre "de" y "Romera".(20 en numeración hexadecimal o 32 en decimal es el valor de la codificación ASCII para el espacio en blanco)Esto ocurre por la forma en que se codifican los ***URL*** en el protocolo ***HTTP.*** Sucede lo mismo con otros símbolos como las tildes u otros caracteres especiales. Esta codificación sigue el esquema:

%+<valor hexadecimal del código ASCII correspondiente al carácter>

Por ejemplo, el carácter á se escribiría como ***%E1*** (código ASCII 225). También se puede cambiar la secuencia ***%20*** por el signo +, obteniendo el mismo efecto.

Los programadores de ***servlets*** no deben preocuparse en principio por este detalle, ya que la clase ***HttpServletRequest*** se encarga de la decodificación, de forma que los valores de los parámetros sean accesibles mediante el método ***getParameter(String parametro)*** de dicha clase. Sin embargo, hay que tener cuidado con algunos caracteres a la hora de incluirlos en u ***URL,*** en concreto con aquellos caracteres no pertenecientes al código ASCII y con aquellos que tienen un significado concreto para el protocolo ***HTTP***

Antes de la aparición de los servlets, ***Java*** ya proporcionaba la posibilidad de codificar un ***URL*** de forma que cumpliera con las anteriores restricciones. Para ello, se puede utilizar la clase ***URLEncoder,*** package ***java.net,*** que es un package estándar de ***Java.*** Dicha clase tiene un único método, ***String encode(String),*** que se encarga de codificar el ***String*** que recibe como argumento devolviendo otro ***String*** con el ***URL*** debidamente codificado. Así, considérese el siguiente ejemplo

import java.net

```
public class Codificar {
      public static void main(String argv[]) {
            String URLcodificada = URLRncoder.encode("/servlet/miServlet?nombre=Antonio"+
                                          +&Apellido=Lépez de Romera");
            System.out.println(URLcodificada);
      }
}
```

que cuando es ejecutado tiene como resultado la siguiente secuencia de caracteres:

%2Fservlet**%2F**MiServlet**%3F**nombre **%3D**Antonio **%26**Apellido **%3D**L **%A2**pez+de+Romera

Obsérvese además que, cuando sea necesario escribir una comilla dentro del ***String out.println(String),*** hay que precederla por el carácter ***escape*** (\). Ejemplo:

```
out.println( "<A HREF=\"http: //www.yahoo.com\">Yahoo</A>");
```

Las peticiones ***HTTP GET*** tienen una limitación importante (recuérdese que transmiten información a través de las variables de entorno del sistema operativo) y es un límite en la cantidad de caracteres que pueden aceptar en el ***URL.*** Si se envían los datos de un formulario muy extenso es necesario utilizar el método ***HTTP POST.***

6.4.2 Método HEAD: información de ficheros

Este método es similar al anterior. La petición del cliente tiene la misma forma que en el método ***GET,*** con la salvedad de que en lugar de ***GET*** se utiliza ***HEAD.*** En este caso el servidor responde dicha petición enviando únicamente ***información acerca del fichero,*** y no el fichero en si.

El método ***HEAD*** se suele utilizar frecuentemente para comprobar lo siguiente:

- ***La fecha de modificación*** de un documento presente en el servidor.
- El ***tamaño del documento*** antes de su descarga, de forma que el browser pueda presentar información acerca del progreso de descarga.
- El tipo de servidor.
- El ***tipo de documento*** solicitado, de forma que el cliente pueda saber si es capaz de soportarlo.

6.4.3 Método POST: el más utilizado

El método ***HTTP POST*** permite al cliente ***enviar información al servidor.*** Se debe utilizar en lugar de ***GET*** en aquellos casos que requieran transferir una cantidad importante de datos (formularios).

El método ***POST*** no tiene la limitación de ***GET*** en cuanto a volumen de informaci~ transferida, pues ésta no va incluida en el ***URL*** de la petición, sino que viaja encapsulada en un ***input stream*** que llega al ***servlet*** a través de la entrada estándar.

El encabezamiento y el contenido (en negrita) de una petición ***POST*** tienen la siguiente forma

```
POST /servlet/MiServlet HTTP/1.1
User-Agent:     Mozilla/4.5 compatib1e;
MSIE 4.01;
Windows NT)
Host:       www.MiServidor.com
Accept:         image/gif, image/x-bitmap, image/jpeg, image/jpeg
Content-type: app1ication/x-www~form-ur1encoded
Content-length:         39

nombre=Antonio&Ape11ido=Lopez%20de%20Romera
```

Nótese la existencia de una ***línea en blanco*** entre el encabezarniento *(header) y* el comienzo de la información extendida. Esta línea en blanco indica el final del ***header.***

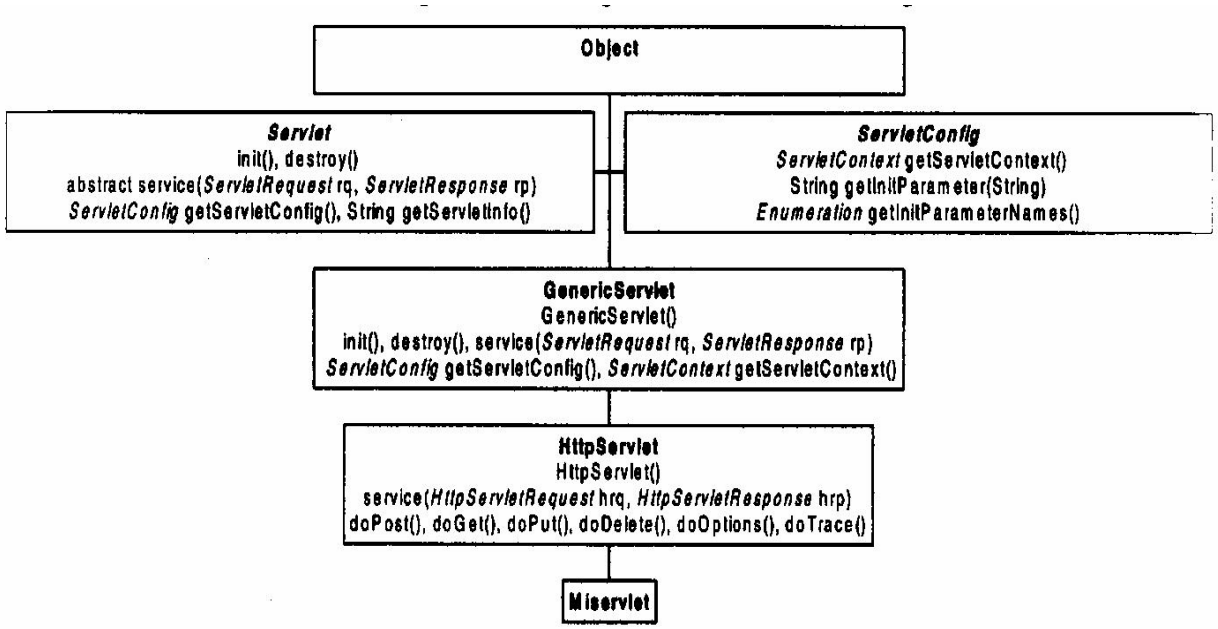

Figura 7. Jerarquía de clases en servlets.

6.4.4 Clases de soporte HTTP

Una vez que se han presentado unas ciertas nociones sobre el protocolo ***HTTP,*** resulta más sencillo entender las funciones del package ***javax.servlet.http,*** que facilitan de sobremanera la creación de ***servlets*** que empleen dicho protocolo.
La clase abstracta ***javax.servlet.http.HttpServlet*** implementa la interface ***javax.servlet.Servlet*** e incluye un numero de importante de funciones adicionales. La forma más sencilla de escribir un ***servlet HTTP*** es heredando de ***HttpServlet.***
La clase ***HttpServlet*** es también una clase ***abstract,*** de modo que es necesario definir una clase que derive de ella y redefinir en la clase derivada ***al menos uno*** de sus métodos, tales como ***doGet(), doPost(),*** etc.

Como ya se ha comentado, la clase ***HttpServlet*** proporciona una implementación del método ***service()*** en la que distingue qué método se ha utilizado en la petición ***(GET, POST,*** etc.), llamando seguidamente al método adecuado ***(DoGet(), doHead(), doDelete(), doOptions(), doPost(), dotarse()).*** Estos métodos e corresponden con los métodos ***HTTP*** anteriormente citados.

Así pues, la clase ***HttpServlet*** no define el método ***service()*** como ***abstract,*** sino como ***protected,*** al igual que los métodos ***init(), destroy(), Donet(), doPost(),*** etc., de forma que ya no es necesario escribir una implementación de ***service()*** en un ***servlet*** que herede de dicha clase Si por algún motivo es necesario redefinir el método ***service(),*** es muy conveniente llamar desde él aJ método ***service()*** de la super-clase ***(HttpServlet).***

La clase ***HttpServlet*** es bastante "inteligente", ya que es también capaz de saber qué método han sido redefinidos en una sub-clase, de forma que puede comunicar al cliente qué tipos de métodos soporta el ***servlet*** en cuestión. Así, si en la clase ***MiServlet*** sólo se ha redefinido el método ***doPost(),*** si el cliente realiza una petición de tipo ***HTTP GET*** el servidor lanzará automáticamente un mensaje de error similar al siguiente:

501 Method GET Not Supported

donde el número que aparece antes del mensaje es un código empleado por los ***servidores HTTP*** para indicar su estado actual. En este caso el código es el ***501.***

No siempre es necesario redefinir todos los métodos de la clase ***HttpServlet.*** Por ejemplo basta definir el método d***oGet()*** para que el ***servlet*** responda por sí mismo a peticiones del tipo ***HTTP HEAD*** o ***HTTP OPTIONS.***

6.4.5 Modo de empleo de la clase HttpServlet

Todos los métodos de clase ***HttpServlet*** que debe o puede redefinir el programador ***(doGet() doPost(), doPut(), doOptions(),*** etc.) reciben como argumentos un objeto ***HttpServletRequest*** y otro ***HttpServletResponse.***

Métodos de la interfase ***HttpServletRequest***:

getAuthType(), getDateHeader(String), getHeader(String), getHeaderNames(), getIntHeader(String), getMethod(), getQueryString()

Métodos de la interface ***HttpServletResponse.***

sendError(int), sendError(int, String), setStatus(int), setStatus(int, String), setHeader(String, String), setDateHeader(String, long)

Por otra parte, el objeto de la interface ***HttpServletResponse*** permite enviar desde el ***servlet*** al cliente información acerca del estado del servidor (métodos ***sendError()*** y ***setStatus()***), así como establecer los valores del ***header*** del mensaje saliente (métodos ***setHeader(), setDateHeader(),*** etc.)

Recuérdese que tanto ***HttpServletRequest*** como ***HttpServletResponse*** son interfaces que derivan de las interfaces ***ServletRequest*** y ***ServietResponse*** respectivamente, por lo que se

puede también utilizar todos los métodos declarados en estas últimas.

7 FORMAS DE SEGUIR LA TRAYECTORIA DE LOS USUARIOS (CLIENTES)

Los ***servlets*** permiten seguir la trayectoria de un cliente, es decir, obtener y mantener una determinada información acerca del cliente. De esta forma se puede ***tener identificado a un*** usuario que está utilizando un browser durante un determinado tiempo. Esto es muy importante se quiere disponer de aplicaciones que impliquen la ejecución de varios ***servlets*** o la ejecución repetida de un mismo ***servlet.*** Un claro ejemplo de aplicación de esta técnica es el de los ***comercios vía Internet*** que permiten llevar un ***carrito de compras*** en el que se van guardando aquellos productos solicitados por el cliente. El cliente puede ir navegando por las distintas secciones del comercio virtual, es decir realizando distintas conexiones ***HTTP*** y ejecutando diversos ***servlets*** y a pesar de ello no se pierde la información contenida en el carrito de la compra y se sabe en todo momento que es un mismo cliente quien está haciendo esas conexiones diferentes.

El mantener información sobre un cliente a lo largo de un proceso que implica múltiples conexiones se puede realizar de tres formas distintas:

- Mediante ***cookies***
- Mediante seguimiento de sesiones (Session Tracking)
- Mediante la ***reescritura*** de URLs

El detalle de cómo se implementa cada una de estas alternativas excede el alcance actual de nuestra asignatura. UD puede obtener mas información en el sistio http.\\labsys.frc.utn.edu.ar, Sitios de las cátedras/PPR 2004/Unidad III/"**Aprenda Servlets de Java.pdf**"