INTRODUCCIÓN

1.1 INTRODUCCIÓN A INTERNET/INTRANET

1.1.1 Introducción histórica

La red ***Internet*** es hoy día la red de ordenadores más extensa del planeta; enlaza centenares de miles de redes locales heterogéneas.

En 1990, ***Tim Berners-Lee,*** un joven estudiante del Laboratorio Europeo de Física dc Partículas ***(CERN)*** situado en Suiza, desarrolló un nuevo sistema de distribución de información en ***Internet*** basado en páginas ***hipertexto,*** al que denominó ***World Wide Web*** (telaraña mundial). La revolución de la ***Web*** había comenzado.

Realmente, el concepto de documento ***hipertexto*** no es nuevo: fue introducido por ***Ted Nelson*** en 1965 y básicamente se puede definir como *texto de recorrido no secuencial.* Clicando en las palabras con ***enlaces*** (links) se puede acceder al documento al que apuntan, que normalmente contiene una información más detallada sobre el concepto representado por las palabras del enlace De ordinario, las palabras del enlace aparecen subrayadas y de un color diferente al del resto dc documento, para que puedan diferenciarse fácilmente. Una vez que han sido clickeadas cambian de color, para indicar que el documento al que apuntan ya ha sido visitado. Lo realmente novedoso de la ***Web*** es la aplicación del concepto de ***hipertexto*** a la inmensa base de información accesible a través de ***Internet.***

Por otra parte, lo que inicialmente se había concebido como un sistema de páginas ***hipertexto*** se ha convertido posteriormente en un verdadero sistema ***hipermedia,*** en el que las páginas permiten acceder a imágenes, sonidos, videos, etc. Ello ha incrementado aún más el atractivo de la ***Web.***

1.1.2 Redes de ordenadores

Una ***red*** es una ***agrupación de computadores.*** Mediante una red, se posibilita el intercambio dc información entre ordenadores de un modo eficiente y transparente. Una red permite ver los discos de otros ordenadores como si fueran discos locales. Según sea la estructura de dicha agrupación, (según el número de ordenadores integrados en ella se pueden establecer diferentes clasificaciones:

- ***Red Local*** (LAN: Local Area Network). Una red dentro de un mismo edificio
- ***Red de campus*** (CAN: Campus Area Network). Une edificios dentro una zona geográfica.
- ***Red de ciudad*** (MAN: Metropolitan Area Network). Une edificios dentro de un área urbana
- **Red** de área extensa (WAN: Wide Area Network). Une centros dispersos en una zona geográfica muy amplia.

1.1.3 Protocolo TCP/IP

Lo que permite que ordenadores remotos con procesadores y sistemas operativos diferentes entiendan y en definitiva que ***Internet*** funcione como lo hace en la actualidad, es un conjunto instrucciones o reglas conocidas con el nombre de ***protocolo.*** La ***Internet*** utiliza varios protocolos(pero los que están en la base de todos los demás son el ***Transport Control Protocol*** **(TCP)** llamado ***Internet Protocol (IP),*** o en definitiva ***TCP/IP*** para abreviar. Se trata de una serie de reglas para mover de un ordenador a otro los datos electrónicos descompuestos en ***paquetes,*** asegurándose de que todos los paquetes llegan y son ensamblados correctamente en su destino. Todos los ordenadores en ***Internet*** utilizan el protocolo ***TCP/IP***

1.1.4 Servicios

Sobre la base de la infraestructura de transporte de datos que proporciona el protocolo ***TCP/IP*** se han construido otros protocolos más específicos que permiten por ejemplo enviar correo electrónico ***(SMTP),*** establecer conexiones y ejecutar comandos en máquinas remotas ***(TELNET),*** accede foros de discusión o *news* ***(NNTP),*** transmitir ficheros ***(FTP),*** conectarse con un servidor web ***(HTTP),*** etc. A estas capacidades de ***Internet*** se les llama ***servicios***

1.1.4.4 World Wide Web

La ***World Wide Web,*** o simplemente ***Web,*** es el sistema de información más completo y actual, que une tanto elementos multimedia como hipertexto. De hecho, tomando el todo por la parte, con mucha frecuencia la ***Web*** se utiliza como sinónimo de ***Internet.***

El ***World Wide Web (Www)*** es el resultado de cuatro ideas:

1. La idea de ***internet*** y los protocolos de transporte de información en que está basada.

2. La concepción de Ted Nelson de un sistema de ***hipertexto,*** extendida a la red.

3. La idea de programas ***cliente*** que interaccionan con programas ***servidores*** capaces de enviar la información en ellos almacenada. Para la ***Web,*** esto se hace mediante el protocolo ***http (HyperText Transfer Protocol).***

4. El concepto de *lenguaje anotado (Markup language)* y más en concreto del lenguaje ***HTML (HyperText Markup Language),*** del que ya hemos visto algo cuando estudiamos Applets.

HTML es una herramienta fundamental de ***Internet.*** Gracias al ***hipertexto,*** desde una página ***Web*** se puede acceder a cualquier otra página ***Web*** almacenada en un servidor ***HTTP*** situado en cualquier parte del mundo. Todo este tipo de operaciones se hacen mediante un programa llamado ***browser*** o ***navegador,*** que básicamente es un programa que reconoce el lenguaje HTML, lo procesa y lo representa en pantalla con el formato más adecuado posible..

1.2 PROTOCOLO HTTP Y LENGUAJE HTML

Hemos citado lo que son los protocolos de ***Internet*** y algunos de sus servicios: se pueden enviar/recibir ficheros de cualquier tipo, correo electrónico, conectarse a un servidor remoto y ejecutar comandos, etc. Sin embargo, ninguno de esos servicios permiten la posibilidad de colabora! en la creación de un entorno hipertexto e hipermedia, es decir, no se pueden pedir datos a un ordenador remoto para visualizarlos localmente utilizando **TCP/IP.** Es por ello que en 1991 se creó el protocolo llamado ***HTTP (HyperText Transport Protocol).***

Una de las características del protocolo ***HTTP*** es que ***no es permanente,*** es decir, una vez que el servidor ha respondido a la petición del cliente la conexión se pierde y el servidor queda en espera, al contrario de lo que ocurre con los servicios de ftp o ***telnet,*** en los cuales la conexión es permanente hasta que el usuario o el servidor transmite la orden de desconexión. La ***conexión no permanente*** tiene la ventaja de que es más difícil que el servidor se colapse o sature, y el inconveniente de que no permite saber que es un mismo usuario el que está realizando diversas.

Se llama ***mantener la sesión*** a la capacidad de un servidor ***HTTP*** y de sus programas asociados para reconocer que una determinada solicitud de un servicio pertenece a un usuario que ya había sido identificado y autorizado. Esta es una característica muy importante en todos los programas de comercio electrónico.

Uno de los lenguajes utilizados para la creación de las páginas Web en Internet es el HTML. Simple, su código se puede escribir con cualquier editor de texto como ***Notepad, Wordpad*** o ***Word.*** Se basa en comandos o tags reconocibles por el browser y que van entre los símbolos '<'y'>'.

El lenguaje ***HTML*** es tan importante que se han creado muchos editores especiales, por ejemplo ***Microsoft FrontPage 98.*** Además, las aplicaciones más habituales (tales como ***Word. Excel*** y ***PowerPoint)*** tienen posibilidad de exportar ficheros ***HTML.*** No es pues nada difícil aprender a crear páginas ***HTML.***

1.3 URL (UNIFORM RESOURCE LOCATOR)

Todo ordenador en ***Internet y*** toda persona que use ***Internet*** tiene su propia dirección electrónica ***(IP address).*** Todas estas direcciones siguen un mismo formato. Por ejemplo

pgcmez@frc.utn.edu.ar

donde ***pgcmez*** es el identificador ID o nombre de usuario que Pedro utiliza para conectarse a la red. Es así como el ordenador le conoce. La parte de la dirección que sigue al símbolo de *arroba (@)* identifica al ordenador en el que está el servidor de correo electrónico. Consta de dos partes: nombre del ordenador o ***host,*** y un identificador de la red local de la institución, llamado ***dominio.*** En este caso el ordenador se llama frc y el dominio es utn.edu.ar. Nunca hay espacios en blanco en una dirección de ***Internet.***

En realidad los ordenadores no se identifican mediante un nombre, sino mediante un número: el llamado número o ***dirección IP,*** que es lo que el ordenador realmente entiende. Los nombres son para facilitar la tarea a los usuarios, ya que son más fáciles de recordar y de relacionar con la institución. Es evidente que es más fácil recordar el nombre que la ***dirección IP.*** En ***Internet*** existen unos servidores especiales, llamados servidores de nombres o de dominios, que mantienen unas tablas mediante las que se puede determinar la ***dirección IP*** a partir del nombre.

Así pues, ***¿qué es exactamente un URL?*** Pues podría concebirse como la extensión de! concepto de nombre completo de un archivo (path). Mediante un ***URL*** no sólo puede apuntarse a un archivo en un directorio en un disco local, sino que además tal archivo y tal directorio pueden estai localizados de hecho en cualquier ordenador de la red, con el mismo o con distinto sistema operativo. Las ***URLs*** posibilitan el direccionamiento de personas, ficheros y de una gran variedad de información, disponible según los distintos protocolos o servicios de ***Internet.*** El protocolo más conocido es el ***HTTP,*** pero ***FTP*** y las direcciones de ***e-mail*** también pueden ser referidas con un ***URL.*** En definitiva, un ***URL*** es como la ***dirección completa*** de un determinado servicio: proporciona todos los datos necesarios para localizar el recurso o la información deseada.

En resumen, un ***URL*** es una manera conveniente y sucinta de referirse a un archivo o a cualquier otro recurso electrónico.

La sintaxis genérica de lOS ***URLs*** es la que se muestra a continuación:

método: // Servidor.dominio/ruta-completa-del-fichero

método es una de las palabras que describen el servicio: ***http, ftp, news, ...***

En ocasiones el URL empleado tiene una sintaxis como la mostrada, pero acabada con una barra (/). Esto quiere decir que no se apunta a un archivo, sino a un ***directorio.*** Según como esté configurado, el servidor devolverá el índice por defecto de ese directorio (un listado de archivos y subdirectorios de ese directorio para poder acceder al que se desee), un archivo por defecto que el servidor busca automáticamente en el directorio (de ordinario llamado *Index.htm o lndex.html)* o quizás impida el acceso si no se conoce exactamente el nombre del fichero al que se quiere acceder (como medida de seguridad).

¿Cómo presentar un ***URL*** a otros usuarios? Se suele recomendar hacerlo de la siguiente manera:

<URL: método: //ordenador.dominio/ruta-completa-del-fichero>

para distinguir así los ***URLs*** de los ***URIs (Uniform Resource Identification),*** que representan un concepto similar pero no idéntico.

A continuación se muestran las distintas formas de construir los ***URLs*** según los distintos servicios de ***Internet:***

1.3.1 URLs del protocolo HTTP

Como ya se ha dicho, ***HTTP*** es el protocolo específicamente diseñado para la ***World Wide Web.*** Su sintaxis es la siguiente:

http:/ /<host> : <puerto>/<ruta>

donde ***host*** es la dirección del servidor WWW, el ***puerto*** indica a través de que "entrada" el servidor atiende los requerimientos ***HTTP*** (puede ser omitido, en cuyo caso se utiliza el valor por defecto. 80), y la ***ruta*** indica al servidor el ***path*** del fichero que se desea cargar (el ***path*** es relativo a un directorio raíz indicado en el ***servidor HTTP).***

Así, por ejemplo, *http://www.msn.com/index/prev/welcome.htm* accede a la ***Web*** de ***Microsoft Network,*** en concreto al archivo ***welcome.htm*** (cuya ruta de acceso es ***index/prev).***

1.3.2 URLs del protocolo FTP

La sintaxis específica del protocoloftp es la siguiente:

ftp: / <Usuario>:<password>@host:<puerto>/<cwd1>/<cwd2>/…/<cwdN>/nombre

Los campos ***usuario y password*** sólo son necesarios si el servidor los requiere para autorizar el acceso; en otro caso pueden ser omitidos; ***host*** es la dirección del ordenador en el que se está ejecutando el servicio ftp; el ***puerto,*** como antes, es una información que puede ser omitida (por defecto suele ser el "21"); la serie de argumentos *<cwd]>/.../<cwdN>* son los comandos que el cliente debe ejecutar para moverse hasta el directorio en el que reside el documento; ***nombre*** es el nombre del documento que se desea obtener.

Así, por ejemplo, *ftp://www.msn.coni/index/prev/welcome.htm* traerá el fichero ***welcome.htm (***cuya ruta de acceso es ***index/prev)*** del servidor ftp de ***Microsoft Network.***

Otros servicios (mail, news, telnet) tienen protocolos específicos.

1.4 CLIENTES Y SERVIDORES

1.4.1 Clientes (clients)

Por su versatilidad y potencialidad, en la actualidad la mayoría de los usuarios de ***Internet*** utilizan en sus comunicaciones con los servidores de datos, los ***browsers*** o ***navegadores.*** Esto no significa que no puedan emplearse otro tipo de programas como clientes ***e-mail, news,*** etc. para aplicaciones más específicas. De hecho, los browsers más utilizados incorporan lectores de mail y de news.

En la actualidad los browsers más extendidos son Netscape Communicator y Microsoft Internet Explorer. A pesar de que ambos cumplen con la mayoría de los estándares aceptados en la ***Internet*** cada uno de ellos proporciona soluciones adicionales a problemas más específicos. Por este motivo, muchas veces será necesario tener en cuenta qué tipo de browser se va a comunicar con un servidor, pues el resultado puede ser distinto dependiendo del browser empleado.

Ambos browsers soportan ***Java,*** lo cual implica que disponen de una ***Java Virtual Machine*** en la que se ejecutan los ficheros ****.class*** de las ***Applets*** que traen a través de ***Internet.*** Netscape es más fiel al estándar de ***Java*** tal y como lo define ***Sun,*** pero ambos tienen la posibilidad de sustituir la ***Java Virtual Machine*** por medio de un mecanismo definido por ***Sun,*** que se llama ***Java Plug-in*** (los ***plug-ins*** son aplicaciones que se ejecutan controladas por los browsers y que perrniten extender sus capacidades, por ejemplo para soportar nuevos formatos de audio o video).

1.4.2 Servidores (servers)

Los ***servidores*** son programas que se encuentran permanentemente esperando a que algún otro ordenador realice una solicitud de conexión. En un mismo ordenador es posible tener simultáneamente servidores de los distintos servicios anteriormente mencionados ***(HTTP, FTP, TELNET,*** etc.). Cuando a dicho ordenador llega un requerimiento de servicio enviado por otro ordenador de la red, se interpreta el tipo de llamada, y se pasa el control de la conexión al ***servidor*** correspondiente a dicho requerimiento. En caso de no tener el ***servidor*** adecuado para responder a la comunicación, está será rechazada. Un ejemplo de rechazo ocurre cuando se quiere conectar a través de ***TELNET*** (típico de los sistemas ***UNIX)*** con un ordenador que utilice ***Windows 95/98.***

Como ya se ha apuntado, no todos los servicios actúan de igual manera. Algunos, como ***TELNET*** y ***FTP,*** una vez establecida la conexión, la mantienen hasta que el cliente o el servidor explícitamente la cortan. Por ejemplo, cuando se establece una conexión con un servidor de ***FTP,*** los dos ordenadores se mantienen en contacto hasta que el cliente cierre la conexión mediante el~ comando correspondiente ***(quit, exit,*** .... o pase un tiempo establecido en la configuración del servidor ***FTP*** o del propio cliente, sin ninguna actividad entre ambos.

La comunicación a través del protocolo ***HTTP*** es diferente, ya que es necesario establecer una comunicación o conexión distinta para cada elemento que se desea leer. Esto significa que en un documento ***HTML*** con 10 imágenes son necesarias 11 conexiones distintas con el servidor ***HTTP,*** esto es, una para el texto del documento ***HTML*** con las *tags* y las otras 10 para traer las imágenes referenciadas en el documento ***HTML.***

La mayoría de los usuarios de ***Internet*** son ***clientes*** que acceden mediante un ***browser*** a los distintos ***servidores*** WWW presentes en la red. El servidor no permite acceder indiscriminadamente a todos sus ficheros, sino únicamente a determinados directorios y documentos previamente establecidos por el *administrador* de dicho servidor.

1.5 TENDENCIAS ACTUALES PARA LAS APLICACIONES EN INTERNET

En la actualidad, la mayoría de aplicaciones que se utilizan en entornos empresariales están construidos en torno a una arquitectura *cliente-servidor,* en la cual uno o varios computadores son los ***servidores,*** que proporcionan servicios a un número mucho más grande de ***clientes*** conectados a través de la red. Los ***clientes*** suelen ser PCs de propósito general, de ordinario menos potentes y más orientados al usuario final. A veces los servidores son intermediarios entre los clientes y otros servidores más especializados (por ejemplo los grandes servidores de bases de datos corporativos basados en ***mainframes*** y/o sistemas ***Unix.*** En este caso se habla se *aplicaciones de varias capas.*

Con el auge de ***Internet,*** la arquitectura ***cliente-servidor*** ha adquirido una mayor relevancia, ya que la misma es el principio básico de funcionamiento de la ***World Wide Web:*** un usuario que mediante un ***browser (cliente)*** solicita un servicio (páginas ***HTML,*** etc.) a un computador que hace las veces de ***servidor.*** En su concepción más tradicional, los servidores ***HTTP*** se limitaban a enviar una página ***HTML*** cuando el usuario la requería directamente o clickeaba sobre un enlace. La interactividad de este proceso era mínima, ya que el usuario podía pedir ficheros, pero no enviar sus datos personales de modo que fueran almacenados en el servidor u obtuviera una respuesta personalizada. La siguiente figura representa gráficamente este concepto.

Arquitectura Cliente Servidor

Desde esa primera concepción del servidor ***HTTP*** como mero servidor de ficheros ***HTML*** el concepto ha ido evolucionando en dos direcciones complementarias:

1. Añadir más inteligencia en el ***servidor, y***

2. Añadir más inteligencia en el ***cliente.***

Las formas más extendidas de añadir inteligencia a los clientes (a las páginas ***HTML)*** ha sido ***Javascript*** y los ***applets de Java. Javascript*** es un lenguaje relativamente sencillo, interpretado, cuyo código fuente se introduce en la página ***HTML*** por medio de los tags <SCRJPT> </SCRIPT>; su nombre deriva de una cierta similitud sintáctica con ***Java.*** Los ***applets de Java*** tienen mucha más capacidad de añadir inteligencia a las páginas ***HTML*** que se visualizan en el

browser, ya que son verdaderas clases de ***Java*** (ficheros ****.class)*** que se cargan y se ejecutan en el cliente. (Ud ya lo ha visto en las unidades anteriores)

Los ***formularios HTML*** permiten de alguna manera invertir el sentido del flujo de información. Utilizando algunos campos con cajas de texto, botones de opción y de selección, el usuario puede definir sus preferencias o enviar sus datos al servidor.

¿Cómo recibe el servidor los datos de un formulario y qué hace con ellos? Éste es el problema que tradicionalmente han resuelto los ***programas CGI.*** Cada formulario lleva incluido un campo llamado ***Action*** con el que se asocia el nombre de programa en el servidor. El servidor arranca dicho programa y le pasa los datos que han llegado con el formulario. Existen dos formas principales de pasar los datos del formulario al ***programa CGI:***

1. Por medio de una variable de entorno del sistema operativo del servidor, de tipo String (método ***GET)***

2. Por medio de un flujo de caracteres que llega a través de la entrada estándar *(stdin o System.in),* que de ordinario está asociada al teclado (método ***POST).***

En ambos casos, la información introducida por el usuario en el formulario llega en la forma de una única cadena de caracteres en la que el nombre de cada campo del formulario se asocia con el valor asignado por el usuario, y en la que los blancos y ciertos caracteres especiales se han sustituido por secuencias de caracteres de acuerdo con una determinada codificación. Más adelante se verán con más detenimiento las reglas que gobiernan esta transmisión de información. En cualquier caso, lo primero que tiene que hacer el ***programa CGI*** es decodificar esta información y separar los valores de los distintos campos. Después ya puede realizar su tarea específica: escribir en un fichero o en una base de datos, realizar una búsqueda de la información solicitada, realizar comprobaciones, etc. De ordinario, el ***programa CGI*** termina enviando al cliente (el navegador desde el que se envió el formulario) una página ***HTML*** en la que le informa de las tareas realizadas, le avisa si se ha producido alguna dificultad, le reclama algún dato pendiente o mal cumplimentado, etc. La forma de enviar esta página ***HTML*** al cliente es a través de la salida estándar *(stduot* o *System.out),* que de ordinario suele estar asociada a la pantalla. La página ***HTML*** tiene que ser construida elemento a elemento, de acuerdo con las reglas de este lenguaje. No basta enviar el contenido: hay que enviar también todas y cada una de las *tags.* En un próximo apartado se verá un ejemplo completo.

En principio, los ***programas CGI*** pueden estar escritos en cualquier lenguaje de programación, aunque en la práctica se han utilizado principalmente los lenguajes ***Peri*** y *C/C++*. Un claro ejemplo de un ***programa CGI*** sería el de un formulario en el que el usuario introdujera sus datos personales para registrarse en un sitio web. El ***programa CGI*** recibiría los datos del usuario introduciéndolos en la base de datos correspondiente y devolviendo al usuario una página ***HTML*** donde se le informaría de que sus datos habían sido registrados. (Exactamente el práctico cuya codificación acabamos de ver).

Figura 2. Arquitectura cliente-servidor interactiva para la WEB.

Es importante resaltar que estos procesos tienen lugar en el servidor. Esto a su vez puede resultar

un problema, ya que al tener múltiples clientes conectados al servidor, el ***programa CGJ*** puede estar siendo llamado simultáneamente por varios clientes, con el riesgo de que el servidor se llegue a saturar. Téngase en cuenta que cada vez que se recibe un requerimiento se arranca una nueva copia del ***programa CGI.*** Existen otros riesgos adicionales que se estudiarán más adelante.

El objetivo de este capítulo es el estudio de la alternativa que ***Java*** ofrece a los ***programas CGI:*** los ***servlets,*** que son a los servidores lo que los ***applets*** a los browsers. Se podría definir un ***servlet*** como ***un programa escrito en Java que se ejecuta en el marco de un servicio de red, (un servidor HTTP, por ejemplo), y que recibe y responde a las peticiones de uno o más clientes.***

A nuestra asignatura le interesan los paradigmas**. "El mejor paradigma es el que tiene menor gap (brecha) con la realidad".** Y la realidad es que las aplicaciones que se desarrollan actualmente lo hacen en el entorno de la Web. A diferencia de las aplicaciones anteriores al esquema cliente/servidor, ahora tiene mucho mas interés añadir inteligencia en el servidor ***HTTP.***

La primera y más empleada tecnología ha sido la de programas **CGI (Common Gateway Interface),** unida a **los formularios HTML.** Sigue un par de ejemplos.

- **INVERSIÓN DE UNA CADENA DE CARACTERES**

En el primero de ellos, el gateway en cuestión, backwards, es un cgi-bin disponible en el sitio java.sun.com. Todo lo que hace es recibir del cliente una cadena de caracteres y devolvérsela revertida.

```
import java.io.*;
import java.net.*;
public class Reverse {
   public static void main(String[] args) throws Exception {
         String stringToReverse = URLEncoder.encode("Aqui me pongo a cantar ...");
         System.out.println("stringToReverse after encode "+stringToReverse);

         URL urlObj=new URL("http://java.sun.com/cgi-bin/backwards");
         URLConnection urlObjConnect = urlObj.openConnection();
         urlObjConnect.setDoOutput(true);
         PrintWriter out = new PrintWriter(connection.getOutputStream());
         /* Definimos el objeto flujo de salida (cliente => URL) out.
         out.println("string=" + stringToReverse);
         /* Aqui enviamos stringToReverse a cgi-bin/backwards */
         out.close(); // Nada mas diremos a backwards
         System.out.println("Ahora, un poquitin de paciencia ...");
         BufferedReader in = new BufferedReader(
                                 new InputStreamReader(
                                 connection.getInputStream()));
         /*      Definimos un objeto in, de tipo BufferedReader cuya mision es posibilitar la lectura
         de la    salida   textual generada por cgi-bin/backwards */
         String inputLine;
         System.out.println("vemos lo generado por cgi-bin/backwards");
         System.out.println("");
         while ((inputLine = in.readLine()) != null)
            System.out.println(inputLine);
         in.close();
   System.out.println("");
   }
}
```

// Y su ejecución:

```
...
stringToReverse after encode Aqui+me+pongo+a+cantar+...
Ahora, un poquitin de paciencia ...
vemos lo generado por cgi-bin/backwards
```

```
Aqui me pongo a cantar ...
 reversed is:
... ratnac a ognop em iuqA

Process Exit...
```

- **INVERSIÓN DE UNA CADENA DE CARACTERES, ahora usando unJAplet**

```
//  Autor: Tymoschuk, Jorge
import java.io.*;
import java.net.*;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextArea;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Graphics;
import java.security.*;
public class Reverse extends JFrame{
        String textTip, textDec;
        JLabel etiqueta;
        JTextArea frase;
        JButton invertir, salir;
        Graphics g;
        int x=10,y=130;         // Coordenadas de la etiqueta para report(...)
        int w=380,h=20;         // Tamaño de la idem

        public void iniciar() {
                try { // usando camiseta Windows
                        javax.swing.UIManager.setLookAndFeel(
                        "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
                } catch (Exception  e) {;}

                // elimina la distribución automática de componentes
                this.getContentPane().setLayout(null);
                this.setSize(420,520);

                // creación y configuración de la etiqueta
                etiqueta = new JLabel(" Frase a revertir, por favor ...");
                etiqueta.setSize(380, 20);
                etiqueta.setLocation(10, 10);
                this.getContentPane().add(etiqueta);

                // creación y configuración del area de texto
                frase = new JTextArea(2,40);
                frase.setSize(270, 50);
                frase.setLocation(10, 30);
                frase.setLineWrap(true);
                this.getContentPane().add(frase);

                ActionListener escuchaAccion = new Eventos(this);

                // creación y configuración del boton invertir
                nvertir = new JButton("Invertir");
                invertir.setSize(80, 27);
                invertir.setLocation(10,90);
                this.getContentPane().add(invertir);
                invertir.addActionListener(escuchaAccion);

                // creación y configuración del boton salir
                salir = new JButton("Salir");
                salir.setSize(80, 27);
```

```
        salir.setLocation(200,90);
        this.getContentPane().add(salir);
        salir.addActionListener(escuchaAccion);
        show();
}       // iniciar()

void leerFrase(){
        textTip=frase.getText();
        this.report(textTip);
}

void codiFrase(){
        textDec=URLEncoder.encode(textTip);
        this.report(textDec);
}

void procFrase() {
        this.report("En tramite de conexion ...");
        try {
                URL urlObj=new URL("http://java.sun.com/cgi-bin/backwards");
                URLConnection urlObjConnect = urlObj.openConnection();
                urlObjConnect.setDoOutput(true);
                this.report("Conexion establecida ...");
                PrintWriter out = new  PrintWriter(urlObjConnect.getOutputStream());
                this.report("Objeto out instanciado...");
                out.println("string=" + textDec);

                //  Aqui enviamos stringToReverse a cgi-bin/backwards
                this.report("Mensaje enviado, paciencia ...");
                out.close(); // Nada mas diremos a backwards
                BufferedReader in = new BufferedReader(new
                                InputStreamReader(urlObjConnect.getInputStream()));

                /* Definimos un objeto in, de tipo BufferedReader cuya
                   mision es posibilitar la lectura de la salida  textual
                   generada por cgi-bin/backwards  */
                String inputLine;
                this.report("Preparandonos a la recepcion ...");
                while ((inputLine = in.readLine()) != null)
                        this.report(inputLine);
                this.report("============================");
                in.close();
        } catch (MalformedURLException exce){
                this.report("Lamento, MalformedURLException");}
         catch (IOException exce){
                this.report("URLConnection, IOException");
                this.report("PrintWriter out, IOException");}
         catch (AccessControlException exce){
                this.report("Lamento, AccessControlException exce");
                this.report("Tema de socket, controlado  en java.sun.com ...");}
}       // void procFrase()

void report(String msge){
        etiqueta = new JLabel(msge);
        etiqueta.setSize(w,h);
        etiqueta.setLocation(x,y+=20);
        this.getContentPane().add(etiqueta);
        repaint();
}

class Eventos implements ActionListener{
        Reverse objRev;
        Eventos(Reverse objRev){
```

```
                this.objRev=objRev;}
        public void actionPerformed(ActionEvent e){
                if(e.getSource() == invertir){
                        objRev.report("Proc. solicitado en curso ...");
                        objRev.leerFrase();
                        objRev.codiFrase();
                        try {
                                objRev.procFrase();
                        } catch (Exception  exce) {
                                objRev.report("Lamento, some Exception ...");}
                }
                else System.exit(1);
        } // actionPerformed
    }   // class Eventos
}   // class Reverse

class prueRev{
   public static void main(String args[]) {
        Reverse objRev=new Reverse();
        objRev.iniciar();
    }   // main()
}
```

- **PROCESANDO UNA ENCUESTA**

A continuación el segundo ejemplo. Es una codificación ejemplo usada como práctico de trabajo en red en el año 2003. Los alumnos codificaban la petición cliente, el programa CGI registraba esta petición en un archivo. El profe, con otro juego cliente/gateway, verificaba quienes cumplimentaban el práctico.

A continuación una codificación tipo del cliente de los alumnos:

```
// Cliente del Gateway Prac001

// Autor: Tymoschuk, Jorge

import java.io.*;
import java.net.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Graphics;
import java.security.*;
public class Clie001 extends JFrame{
    String parametros;
    String textTip, textDec;
    JLabel etiqueta;
    JTextField curso, legajo, nombres;
    JTextArea frase;
    JButton procesar, salir;
    int x=10,y=240;          // Coordenadas de la etiqueta para report(...)
    int w=380,h=20;                  // Tamaño de la idem
    Graphics g;

    public void iniciar() {
    try { // usando camiseta Windows
       javax.swing.UIManager.setLookAndFeel(
       "com.sun.java.swing.plaf.windows.WindowsLookAndFeel");
    } catch (Exception  e) {;}

    // elimina la distribución automática de componentes
    this.getContentPane().setLayout(null);
```

```
this.setSize(420,520);

// creación y configuración de etiquetas y campos de texto
etiqueta = new JLabel("Primer Practico de TRABAJO en RED");
etiqueta.setSize(300, 20);
etiqueta.setLocation(100, 10);
this.getContentPane().add(etiqueta);
etiqueta = new JLabel("Curso");
etiqueta.setSize(50, 20);
etiqueta.setLocation(100, 50);
this.getContentPane().add(etiqueta);

curso = new JTextField(3);
curso.setSize(30, 20);
curso.setLocation(150, 50);
this.getContentPane().add(curso);

etiqueta = new JLabel("Legajo");
etiqueta.setSize(60, 20);
etiqueta.setLocation(220, 50);
this.getContentPane().add(etiqueta);

legajo = new JTextField(5);
legajo.setSize(45, 20);
legajo.setLocation(270, 50);
this.getContentPane().add(legajo);

etiqueta = new JLabel("Apellido, Nombres");
etiqueta.setSize(120, 20);
etiqueta.setLocation(50, 80);
this.getContentPane().add(etiqueta);

nombres = new JTextField(20);
nombres.setSize(270, 20);
nombres.setLocation(50,100);
this.getContentPane().add(nombres);

etiqueta = new JLabel("Algo que quiero decir");
etiqueta.setSize(120, 20);
etiqueta.setLocation(50, 130);
this.getContentPane().add(etiqueta);

frase = new JTextArea(3,40);
frase.setSize(270, 50);
frase.setLocation(50, 150);
frase.setLineWrap(true);
this.getContentPane().add(frase);

ActionListener escuchaAccion = new Eventos(this);

// creación y configuración del botón Procesar
procesar = new JButton("Procesar");
procesar.setSize(90, 27);
procesar.setLocation(50,220);
this.getContentPane().add(procesar);
procesar.addActionListener(escuchaAccion);

// creación y configuración del boton salir
salir = new JButton("Salir");
salir.setSize(60, 27);
salir.setLocation(260,220);
this.getContentPane().add(salir);
salir.addActionListener(escuchaAccion);
```

```
        show();
    }   // iniciar()

    void leerTodo(){
            parametros = " "+"tymos"+
                                " "+"regPra01"+
                                " "+ curso.getText()+
                                " "+ legajo.getText()+
                                " "+ nombres.getText();

            parametros = URLEncoder.encode(parametros);
            this.report(parametros);
            textTip = frase.getText();
            textDec = URLEncoder.encode(textTip);
            this.report(textDec);
    }

    void procFrase() {
            this.report("En tramite de conexion ...");
            try {
                    URL urlObj=new URL("http://labsys.frc.utn.edu.ar/cgi-
                                        bin/java.cgi?Prac001"+parametros);
                    URLConnection urlObjConnect = urlObj.openConnection();
                    urlObjConnect.setDoOutput(true);
                    this.report("Conexion establecida ...");
                    PrintWriter out = new PrintWriter(urlObjConnect.getOutputStream());
                    this.report("Objeto out instanciado...");
                    out.println("string=" + textDec);
                    //  Aqui enviamos stringToReverse a cgi-bin/backwards
                    this.report("Mensaje enviado, paciencia ...");
                    out.close(); // Nada mas diremos a backwards
                    BufferedReader in = new BufferedReader(
                            new InputStreamReader(
                            urlObjConnect.getInputStream()));

                    /*      Definimos un objeto in, de tipo BufferedReader cuya mision es
                    posibilitar la lectura de la       salida   textual generada por
                    cgi-bin/backwards  */
                    String inputLine;
                    this.report("Preparandonos a la recepcion ...");
                    while ((inputLine = in.readLine()) != null)
                            this.report(inputLine);
                    this.report("============================");
                    in.close();
            } catch (MalformedURLException exce){
                    this.report("Lamento, MalformedURLException");}
             catch (IOException exce){
                    this.report("URLConnection, IOException");
                    this.report("PrintWriter out, IOException");}
             catch (AccessControlException exce){
                    this.report("Lamento, AccessControlException exce");
                    this.report("Tema de socket, controlado  en java.sun.com ...");}
    }

    void report(String msge){
        etiqueta = new JLabel(msge);
        etiqueta.setSize(w,h);
        etiqueta.setLocation(x,y+=10);
        this.getContentPane().add(etiqueta);
        repaint();
    }

    class Eventos implements ActionListener{
```

```
        Clie001 clieServ;
        Eventos(Clie001 clieServ){
        this.clieServ=clieServ;}
        public void actionPerformed(ActionEvent e){
            if(e.getSource() == procesar){
                clieServ.leerTodo();
                try {
                    clieServ.procFrase();
                } catch (Exception  exce) {
                    clieServ.report("Lamento, some Exception ...");}
            }       // if (e.getSource()
            else System.exit(1);
        }       // actionPerformed
    }       // class Eventos
}           // class Clie001


class PrueClie001{
    public static void main(String args[]) {
        Clie001 clieServ=new Clie001();
        clieServ.iniciar();
    }  // main()
}   // class PrueClie001
```

El "gateway" o programa CGI-bin que atiende a los alumnos clientes, a continuación:

```
//  Gateway para el primer practico de Trabajo en Red

//  Author: Tymoschuk, Jorge

import java.io.*;
import java.net.*;
import java.util.StringTokenizer;

public class Prac001{  //
        private File      file;
        private RandomAccessFile iofile;
        private String legLei, curLei,  aNomLei;
        private String algoDecir="";
        private String direct, archivo, curso, legajo, apeNoms;
        public Prac001(String[] args){ // El constructor
                direct   = args[0];       // Directorio (tymos)
                archivo  = args[1];       // Archivo    (regPra01)
                curso    = args[2];       // Donde cursas ?
                legajo   = args[3];       // Legajo UTN
                apeNoms  = args[4];   // Apellido, Nombres
        }

        public static void main (String[] args) throws IOException{
                Prac001 gtw = new Prac001(args);
                if (gtw.errorParm(args))return;

                // Consistencia de argumentos: correcta
                if (gtw.openRAF())               // Consigo abrir Random Access File
                        if(!gtw.existeLeg()) // No tengo legajo: alta
                                gtw.grabRAF();
        }

        boolean errorParm(String[] args) throws IOException{
                if ( args.length != 5){   // Cantidad de parametros difiere
                        System.out.println("Content-Type: text/html");  // 1st line
                        System.out.println();                       // empty line
                        System.out.println("<html><H1> Error in parameters </H1>");
```

```
                System.out.println("<P>Quantity received differs from
                                                expected<P></html>");
                return true;
        }
        if ( !args[0].equals("tymos")){ // Directorio mal informado
                System.out.println("Content-Type: text/html");  // 1st line
                System.out.println();                       // empty line
                System.out.println("<html><H1> Error in parameters </H1>");
                System.out.println("<P>args[0] differs from tymos <P></html>");
                return true;
        }
        if ( !args[1].equals("regPra01")){ // Archivo mal informado
                System.out.println("Content-Type: text/html");  // 1st line
                System.out.println();                       // empty line
                System.out.println("<html><H1> Error in parameters </H1>");
                System.out.println("<P>args[1] differs from regPra01 <P></html>");
                return true;
        }
        return false; // No hubo error en los parámetros, sigamos adelante
}

boolean openRAF(){
        file = new File("./"+direct+"/"+archivo);
        try     {iofile = new RandomAccessFile(file,"rw");
                return true;
        }catch (IOException e){
                System.out.println("Content-Type: text/html");  // 1st line
                System.out.println();                       // empty line
                System.out.println("<html><H1> Error in new RAF </H1>");
                System.out.println("<P>openRAF(), IOException<P></html>");
                return false;
                }
}       //      openRAF()

boolean existeLeg(){
        if(file.length()==0) return false; // primera vez
        String legLei="00.000";
        try{
                while(true){
                        if(legLei.equals(legajo)){      // Encontre igual
                                System.out.println("Content-Type: text/html");  // 1st line
                                System.out.println();                       // empty line
                                System.out.println("<html><H1> Existes !!</H1>");
                                System.out.println("<P>"+apeNoms+"<P></html>");
                                return true;
                        } // if
                        legLei= iofile.readUTF();       //  legajo
                        curLei= iofile.readUTF();       //  curso
                        aNomLei=iofile.readUTF();       //  nombres
                        iofile.readUTF();               //  algo que quiero decir
                }       // while
        }       catch(EOFException e){return false;} // No tengo legajo
                catch(IOException e){
                        System.out.println("Content-Type: text/html");  // 1st line
                        System.out.println();                       // empty line
                        System.out.println("<html><H1> Error de lectura (?) </H1>");
                        System.out.println("<P>existeLeg(), IOException<P></html>");
                        System.exit(1);
                }
        return true;
}       //      existeLeg()

void grabRAF(){
```

```
                try{
                        iofile.writeUTF(legajo); //  Primero grabamos los parámetros
                        iofile.writeUTF(curso);
                        iofile.writeUTF(apeNoms);

                        // Ahora leemos el flujo enviado por el cliente.
                        int c;
                        while ((c = System.in.read() ) >  -1 )
                            algoDecir+=c;

                        iofile.writeUTF(algoDecir); //  Y lo grabamos …
                        iofile.close();                 //  Cerramos

                        // confirmacion del exito logrado
                        System.out.println("Content-Type: text/html");  // 1st line
                        System.out.println();                       // empty line
                        System.out.println("<html><H1> Practico exitoso !!! </H1>");
                        System.out.println("<P>"+apeNoms+"<P></html>");
                        System.exit(1);
                } catch (IOException e){
                        System.out.println("Content-Type: text/html");  // 1st line
                        System.out.println();                       // empty line
                        System.out.println("<html><H1> Error de grabacion </H1>");
                        System.out.println("<P>"+apeNoms+"<P></html>");
                        System.exit(1);
                        }
        }       //      grabRAF()

}       //  public class Prac001
```

2 DIFERENCIAS ENTRE LAS TECNOLOGÍAS CGI Y SERVLET

La tecnología ***Servlet*** proporciona las mismas ventajas del lenguaje ***Java*** en cuanto a ***portabilidad*** *("write once, run anywhere")* y ***seguridad,*** ya que un ***servlet*** es una ***clase*** de ***Java*** igual que cualquier otra, y por tanto tiene en ese sentido todas las características del lenguaje. Esto es algo de lo que carecen los ***programas CGI,*** ya que hay que compilarlos para el sistema operativo del servidor y no disponen en muchos casos de técnicas de comprobación dinámica de errores en tiempo de ejecución.

Otra de las principales ventajas de los ***servlets*** con respecto a los ***programas CGI,*** es la de rendimiento, y esto a pesar de que ***Java*** no es un lenguaje particularmente rápido. Mientras que es necesario cargar los ***programas CGI*** tantas veces como peticiones de servicio existan por parte de los clientes, los ***servlets,*** una vez que son llamados por primera vez, ***quedan activos en la nemoria del servidor hasta que el programa que controla el servidor los desactiva.*** De esta manera se minimiza en gran medida el tiempo de respuesta.

Además, los ***servlets*** se benefician de la gran capacidad de ***Java*** para ejecutar métodos en ordenadores remotos, para conectar con bases de datos, para la seguridad en la información, etc. Sc podría decir que las ***clases estándar de Java*** ofrecen resueltos muchos problemas que con otros lenguajes los tiene que resolver el programador.

3 CARACTERÍSTICAS DE LOS *SERVLETS*

Además de las características indicadas en el apartado anterior, los ***servlets*** tienen las siguientes características:

1. Son independientes del servidor utilizado y de su sistema operativo, lo que quiere decir que a' pesar de estar escritos en ***Java,*** el servidor puede estar escrito en cualquier lenguaje de programación, obteniéndose exactamente el mismo resultado que si lo estuviera en ***Java.***

2. Los ***servlets*** pueden llamar a otros ***servlets,*** e incluso a métodos concretos de otros ***servlets*** De esta forma se puede distribuir de forma más eficiente el trabajo a realizar. Por ejemplo, se podría tener un ***servlet*** encargado de la interacción con los clientes y que llamara a otro ***servlet*** para que a su vez se encargara de la comunicación con una base de datos. De igual forma, los ***servlets*** permiten *redireccionar* peticiones de servicios a otros ***servlets*** (en la misma máquina o en una máquina remota).

3. Los ***servlets*** pueden obtener fácilmente información acerca del ***cliente*** (la permitida por el protocolo ***HTTP),*** tal como su dirección ***IP,*** el ***puerto*** que se utiliza en la llamada, el método utilizado ***(GET, POST,*** ...), etc.

4. Permiten además la utilización de ***cookies y sesiones,*** de forma que se puede guardar información específica acerca de un usuario determinado, personalizando de esta forma la interacción cliente-servidor. Una clara aplicación es ***mantener la sesión*** con un cliente.

5. Los ***servlets*** pueden actuar como enlace entre el cliente y una o varias ***bases de datos*** en arquitecturas *cliente-servidor de 3 capas* (si la base de datos está en un servidor distinto).

6. Asimismo, pueden realizar tareas de ***proxy*** para un ***applet.*** Debido a las restricciones de seguridad, un ***applet*** no puede acceder directamente un servidor de datos localizado en cualquier máquina remota, pero el ***servlet*** sí puede hacerlo de su parte.

7. Al igual que los ***programas CGI,*** los ***servlets*** permiten la generación dinámica de código ***HTML*** dentro de una propia página ***HTML.*** Así, pueden emplearse ***servlets*** para la creación de contadores, banners, etc.

4 JSDK 2.0

El ***JSDK (Java Servlet Developer Kit),*** distribuido gratuitamente por ***Sun,*** proporciona el conjunto de herramientas necesarias para el desarrollo de ***servlets*** El ***JSDK*** se encuentra disponible en la dirección de ***Internet*** http://Java.sun.com para diversas plataformas (Windows, Linux, Solares). Consta de tres partes:

1. El ***API*** del ***JSDK,*** que se encuentra diseñada como una ***extensión*** del ***JDK*** propiamente dicho. Consta de dos *packages* que se encuentran contenidos en ***javax.servlet*** y ***javax.servlet.http.*** Este último es una particularización del primero para el caso del protocolo **HTTP,** el más extendido en la actualidad.

2. La ***documentación*** propiamente dicha del ***API*** y el código fuente de las clases

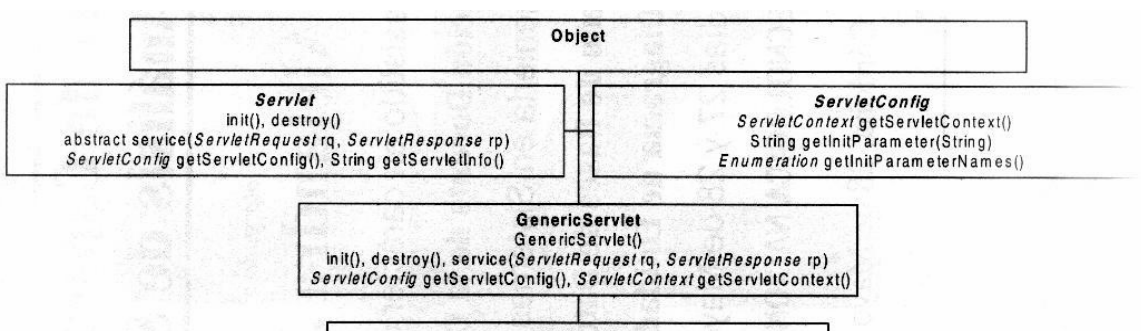
3. La aplicación ***servletrunner,*** que es una simple utilidad que permite probar los ***servlets*** creados sin necesidad de hacer complejas instalaciones de servidores ***HTTP.*** Es similar en concepción al ***appletviewer*** del ***JDK.*** Su utilización será descrita mas adelante.

4.1 VISIÓN GENERAL DEL API DE JSDK 2.0

El *JSDK 2.0* condene dos packages: *javax.servlet y javax.servlet.http.* Todas las clases e interfaces que hay que utilizar en la programación de *servlets* están en estos dos packages.

La relación entre las clases e interfaces de *Java,* muy determinada por el concepto de *herencia,* se entiende mucho mejor mediante una representación gráfica tal como la que puede verse en la Figura siguiente. En dicha figura se representan las *clases* con letra normal y las *interfaces con cursiva.*

La clase ***Genericservlet*** es una clase *abstract* puesto que su método *service()* es *abstract.* Esta clase implementa dos interfaces, de las cuales la más importante es la interface *Servlet.*



La interface *Servlet* declara los métodos más importantes de cara a la vida de un servlet: INIT*()* que se ejecuta sólo al arrancar el *servlet; destroy()* que se ejecuta cuando va a ser destruido v *service()* que se ejecutará cada vez que el *servlet* deba atender una solicitud de servicio.
Cualquier clase que derive de *GenericServ¡et* deberá definir el método *service().* Es muy interesante observar los dos argumentos que recibe este método, correspondientes a las interfaces *ServletRequest y ServletResponse.* La primera de ellas referencia a un objeto que describe por completo la solicitud de servicio que se le envía al servlet. Si la solicitud de servicio viene de un formulario HTML, por medio de ese objeto se puede acceder a los nombres de los campos y a los valores introducidos por el usuario; puede también obtenerse cierta información sobre el cliente (ordenador y browser). El segundo argumento es un objeto con una referencia de la interface *SerletResponse,* que constituye el camino mediante el cual el método *service()* se conecta de nuevo con el cliente y le comunica el resultado de su solicitud. Además, dicho método deberá realizar cuantas operaciones sean necesarias para desempeñar su cometido: escribir y/**O** leer datos de un fichero, comunicarse con una base de datos, etc. El método *service()* es realmente el corazón del servlet.

En la práctica, salvo para desarrollos muy especializados, todos los servlets deberán construirse a partir de la clase *HttpServlett,* sub-clase de *GenericServlet.*

La clase *HttpServlet* ya no es *abstn¡ct* y dispone de una implementación o definición del método *service().* Dicba implementación detecta el tipo de servicio o método *HTTP* que le ha sido solicitado desde el browser y llama al método adecuado de esa misma clase *(doPost(), Donet(),* etc.). Cuando el programador crea una sub-clase de *HttpServlet,* por lo general no tiene que redefinir el método *service(),* sino uno de los métodos más especializados (norrnalmente *doPost()),* que tienen los mismos argumentos que *service():* dos objetos referenciados por las interfaces *ServletRequest* y *ServletResponse.*

En la Figura 3 aparecen otras ***interfaces,*** cuyo papel se resume a continuación.

1. La interface *ServletContext* permite a los *servlets* acceder a información sobre el entorno en que se están ejecutando.

2. La interface *ServletConfig* define métodos que permiten pasar al *servlet* información sobre sus parámetros de inicialización.

3. La interface *ServletRequest* permite al método *service()* de *GenericServIet* obtener información sobre una petición de servicio recibida de un cliente. Algunos de los datos proporcionados por *GenericServlet* son los nombres y valores de los parámetros enviados por el formulario HTML y una input stream.

4. La interface *ServietResponse* permite al método *service()* de *GenericServlet* enviar su respuesta al cliente que ha solicitado el servicio. Esta interfase dispone de métodos para obtener un ***output stream*** o un ***writer*** con los que enviar al cliente datos binarios o caracteres respectivamente.

5. La interfase ***HttpServletRequest*** deriva de ***ServletRequest.*** Esta interfase permite a los métodos ***service(), doPost(), Donet(),*** etc. de la clase ***HttpServlet*** recibir una petición dL servicio ***HTTP.*** Esta interface permite obtener información del header de la petición de servicio ***HTTP.***

6. La interface ***HttpServletResponse*** extiende ***ServíetResponse.*** A través de esta interfase lo métodos de ***HttpServlet*** envían información a los clientes que les han pedido algún servicio.

El ***API*** del ***JSDK 2. O*** dispone de clases e interfaces adicionales, no citadas en este apartado

4.2 LA APLICACIÓN SERVLETRUNNER

Servietrunner es la utilidad que proporciona ***Sun*** conjuntamente con el ***JSDK.*** Es a los ***servlets lo*** que el ***appletviewer*** a los ***applets.*** Sin embargo, es mucho más útil que ***appletviewer,*** porque mientras es muy fácil disponer de un ***browser*** en el que comprobar las ***applets,*** no es tan sencillo instalar y disponer de un ***servidor HTTP*** en el que comprobar los ***servlets.*** Por esta razón al aplicación ***servletrunner***, es una herramienta muy útil para el desarrollo de ***servlets,*** pues se ejecuta desde la línea de comandos del ***MS-DOS.*** Como es natural, una vez que se haya probado debidamente el funcionamiento de los ***servlets,*** para una aplicación real sería preciso emplear ***servidores HTTP*** profesionales.

Además, ***servletrunner*** es ***multithread,*** lo que le permite gestionar múltiples peticiones a la vez. Gracias a ello es posible ejecutar distintos ***servlets*** simultáneamente o probar ***servlets*** que llaman a su vez a otros ***servlets.***

Una advertencia: ***servíetrunner*** no carga de nuevo de modo automático los ***servlets*** que hayan sido actualizados externamente; es decir, si se cambia algo en el código de un ***servlet y*** se vuelve compilar, al hacer una nueva llamada al mismo ***servletrunner*** utiliza la copia de la anterior versión del ***servlet*** que tiene cargada. Para que cargue la nueva es necesario cerrar el ***servletrunner (Ctrl+C)*** y reiniciarlo otra vez. Esta operación habrá que realizarla cada vez que se modifique el ***servlet.***

Para asegurarse de que ***servietrunner*** tiene acceso a los packages del ***Servlet API,*** será necesario comprobar que la variable de entorno ***CLASSPATH*** contiene la ruta de acceso del fichero ***jsdk.jar*** en el directorio ***lib.*** En la plataforma ***Java 2*** es más sencillo simplemente copiar el ***JAR*** al directorio ***ext*** que se encuentr~ en ***\Jre\1ib.*** Esto hace que los ***packages*** sean tratados como extensiones estándar de ***Java.*** También es necesario cambiar la variable ***PATH*** para que se encuentre la aplicación ***servletrunner.exe.*** Otra posibilidad es copiar esta aplicación al directorio donde están los demás ejecutables de ***Java*** (El directorio bin)

4.3 FICHEROS DE PROPIEDADES

Servletrunner permite la utilización de ficheros que contienen las propiedades ***properties*** utilizadas en la configuración, creación e inicialización de los ***servlets.*** Las propiedades son pares del tipo ***clave/valor.*** Por ejemplo, ***servlet.catalogo.codigo=servletcatalogo*** es una propiedad cuya "clave" es ***servlet.catalogo.codigo*** y cuyo "valor" es ***ServetCatalogo..***

Existen ***dos propiedades*** muy importantes para los ***servlets:***

1. servlet.*nombre*.code
2. servlet.*nombre*.initars

La propiedad ***servlet.nombre.code*** debe contener el nombre completo de la clase del ***servlet*** incluyendo *su* ***package.*** Por ejemplo, la propiedad,
servlet. **libros .** code **= basededatos . ServletLibros**

asocia el nombre ***libros*** con la clase ***basededatos.ServletLibros.***

La propiedad ***initargs*** contiene los parámetros de inicialización del ***servlet.*** El valor de un único parámetro se establece en la forma N***ambreDelParametro=valorDeParametro.*** Es posible establecer el valor de varios parámetros a la vez, pero el conjunto de la propiedad debe ser una única línea lógica. Por tanto, para una mayor legibilidad será preciso emplear el carácter ***barra invertida*** (\) para emplear varias líneas del fichero. Así, por ejemplo:

```
serv~et. librodb. initArgs=\
fichero=servlets/Datos, \
usuario =administrador, \...
```

Obsérvese que los distintos parámetros se encuentran separados por ***comas*** (,). El último éL los parámetros no necesitará ninguna coma al final.

Todas estas propiedades estarán almacenadas en un fichero que por defecto tiene el nombre

servlet.properties (se puede especificar otro nombre en la línea de comandos de ***servietrunner***). Se pueden incluir ***líneas de comentario,*** que deberán comenzar por el carácter (#). Por defecto, este fichero debe estar en el mismo directorio que el ***servlet,*** pero al ejecutar ***servletrunner*** puede especificarse un nombre de fichero de propiedades con un ***path*** diferente.

4.4 EJECUCIÓN DE LA APLICACIÓN SERVLETRUNNER

La aplicación ***servletrunner*** se ejecuta desde la línea de comandos de ***MS-DOS*** y admite los siguientes parámetros (aparecen tecleando en la consola ***"servletrunner?"):***

-p puerto al que escuchar
-m número máximo de conexiones
-t tiempo de desconexión en milisegundos
-d directorio en el que están los servlets
-s nombre del fichero de propiedades

Así por ejemplo, si se tuviera un ***servlet*** en el directorio ***c:\programas,*** el fichero de propiedades se llamara ***ServletEjemplo.prop*** y se quisiera que el ***servíetrunner*** estuviera escuchando el ***puerto*** 8000, habría que escribir lo siguiente en la línea de comandos:
C:\servletrunner -p 8000 -d c:\programas -s ServletEjernplo.prop

5 EJEMPLO INTRODUCTORIO

Para poder hacerse una idea del funcionamiento de un ***servlet*** y del aspecto que tienen los mismos lo mejor es estudiar un ejemplo sencillo. Imaginase que en una página web se desea recabar la opinión de un visitante así como algunos de sus datos personales, con el fin de realizar un estudio estadístico. Dicha información podría ser almacenada en una base de datos para su posterior estudio

La primera tarea sería diseñar un formulario en el que el visitante pudiera introducir los datos. Este paso es idéntico a lo que se haría al escribir un ***programa CGI,*** ya que bastará con utilizar los *tags* que proporciona el lenguaje ***HTML*** **(<FORM>, <ACTION>, <TYPE>,** etc.). Algo de esto se ha visto al comiezo de la Unidad III, cuando estudiamos Applets.

5.1 INSTALACIÓN DEL JAVA SERVLET DEVELOPMENT KIT (JSDK 2.0)

Para poder ejecutar este ejemplo es necesario que el ***JSDK 2.0*** esté correctamente instalado, O que Ud utilice una herramienta que lo soporte (Web Sphere ?) Para realizar esta instalación en un ordenador propio se pueden seguir los siguientes pasos:

1. En primer lugar se debe conseguir el fichero de instalación, llamado **j*sdk2O-win32.exe.*** Este fichero se puede obtener de ***Sun*** *(httpl/wwwjavasoft.com/products/servlet/index.htrnl).*

2. Se copia el fichero citado al directorio ***C:\Temp*** del propio ordenador. Doble clic sobre dicho fichero y comienza el proceso de instalación.

3. Se determina el directorio en el que se realizará la instalación. El programa de instalación propone el directorio ***C:\jsdk2.0,*** que es perfectamente adecuado.
4. En el directorio ***C:\jsdk2.0\win*** aparece la aplicación ***servletrunner.exe.*** Para que esta aplicación sea encontrada al tipear su nombre en la ventana de ***MS-DOS*** es necesario que el nombre de dicho directorio sea declarado en la variable de entorno ***PATH.*** Una posibilidad es modificar de modo acorde dicha variable y otra copiar el fichero ***servletrunner.exe*** al directorio donde están los demás ejecutables de Java***;*** como ese directorio ya está en el ***PATH,*** la aplicación ***servletrunner.exe*** será encontrada sin dificultad. Ésta es la solución más sencilla.

5. Además de encontrar ***servletrunner.exe,*** tanto para compilar los servlets como ejecutarlos con ***servíetrunner*** es necesario encontrar las clases e interfaces del ***API*** de ***JSDA 2.0.*** Estas clases pueden estar por ejemplo en el archivo ***C:\¡sdk2.0\lib\jsdk.jar.*** Para que este archivo pueda ser localizado, es necesario modificar la variable de entorno ***CL4SSPATH*** Esto se puede hacer:

```
set CLASSPATH=C: \Jsdk2.O\lib\jsdk.jar;%CLASSPATH%
```