INTRODUCCION A LA PROGRAMACION DISTRIBUIDA

Solución a necesidades:

- Repartir el volumen de información.
- Compartir recursos, software o hardware.

Dos tipos principales:
- sistemas computacionales distribuidos (Esto vemos)
- sistemas de procesamiento paralelo

No es fácil:
- múltiples modelos
- muchos aspectos de seguridad.
- diversas tecnologías.
- Dificultades en prueba y depuración

Cómo se comunican cliente y servidor?
- **Java:** herramienta ideal
  - portabilidad,
  - seguridad y
  - amplio abanico de componentes.

Sistema distribuido consiste
- en una colección de ordenadores autónomos
- unidos por una red y
- con un sistema que les permite compartir recursos hardware, software, datos.

Cómo comunicarse?

- **SOCKETS** (esto vemos)
  - puntos finales de comunicacion entre procesos.
  - se pueden intercambiar datos con otros procesos
  - Se identifican por:
    - dirección de Internet
    - un puerto.
  - **tipos de sockets:**
    - Stream: orientado a la conexión, (esto vemos)
    - Datagram: transporte sin conexión, (no vemos)
    - Raw: protocolos de más bajo nivel, (no Java)
    - Multicast: Datagram con capacidades adicionales (no)

- **RMI (Remote Method Invocation)**
  - un objeto en una máquina puede invocar métodos de un objeto que se encuentra en otra máquina
  - RMI permite instanciar objetos en máquinas locales y remotas
  - RMI permite abstraer las interfaces de comunicación
  - RMI permite trabajar con abstracción del protocolo.

  - **Arquitectura de RMI:**
  - **stub/skeleton:** transmitir objetos a <> espacios de direcciones.
  - **Remote Reference Layer** crea independencia de protocolos.
  - **Transpor Layer** crea conexiones a <> espacios de direcciones.
  - Es más simple que sockets.

- **CORBA** (Common Object Request Broker Architecture) (No vemos)
  - ESPECIFICACIÓN PARA CREAR Y USAR OBJETOS DISTRIBUIDOS
  - Desarrollar en CORBA es similar a hacerlo con RMI
    (la interfaz debe ser definida primero, lenguaje IDL)
  - Objetos CORBA son diferentes porque:
    - Pueden ejecutarse en cualquier plataforma.
    - Pueden situarse en cualquier punto de la red (ventaja)
    - El uso de CORBA es muy complejo (desventaja)

- **AGENTES MOVILES**
  - entidades que circulan libremente por la red
  - esperando llamadas del cliente.
  - Representan un nuevo paradigma en la programación distribuida.
  - ventajas principales:
    - Eficiencia: consumen menos recursos de red
    - Utilizan menos ancho de banda
    - Son robustos y tolerantes a fallos.
    - Soportan entornos heterogéneos.
    - Buenos para aplicaciones de comercio electrónico.
    - Fácil desarrollo.
  - **Voyager**: plataforma Java para crear sistemas distribuidos

- **SOAP** (*Simple Object Access Protocol*)
  - protocolo basado en XML
  - comunica componentes y aplicaciones mediante http
  - para comunicación SOAP segura usar el protocolo HTTPS
  - Propiedades de SOAP:
    - Es un protocolo ligero
    - Es simple y extensible
    - Se usa para comunicación entre aplicaciones
    - Está diseñado para comunicarse vía http
    - No está ligado a ninguna tecnología de componentes
    - No está ligado a ningún lenguaje de programación
    - Está basado en XML
    - Está coordinado por el W3C
    - SOAP es pieza clave de la tecnología **.NET de Microsoft** (protocolo abierto)
- **JINI**

  - construye redes para sistemas dinámicos,
  - arquitectura apta para sistema complejo con cambios.
  - ideal adaptación automática a modificaciones en la red.
  - fácil de usar y aprender.
  - puede adecuarse todas las tecnologías y servicios.
  - Todas las representaciones en Jini son objetos java.
  - Gracias a esta tecnología podemos:
    - Buscar servicios "multicast".
    - Buscar servicios para autentificar código.
    - Buscar servicios que mandan alertas.

    encontrar un servicio: un cliente Jini lo localiza por tipo en "Lookup Service" (Interfaz de Java), después mueve el servicio al cliente y el código que el servicio necesita usar es dinámicamente cargado en demanda.

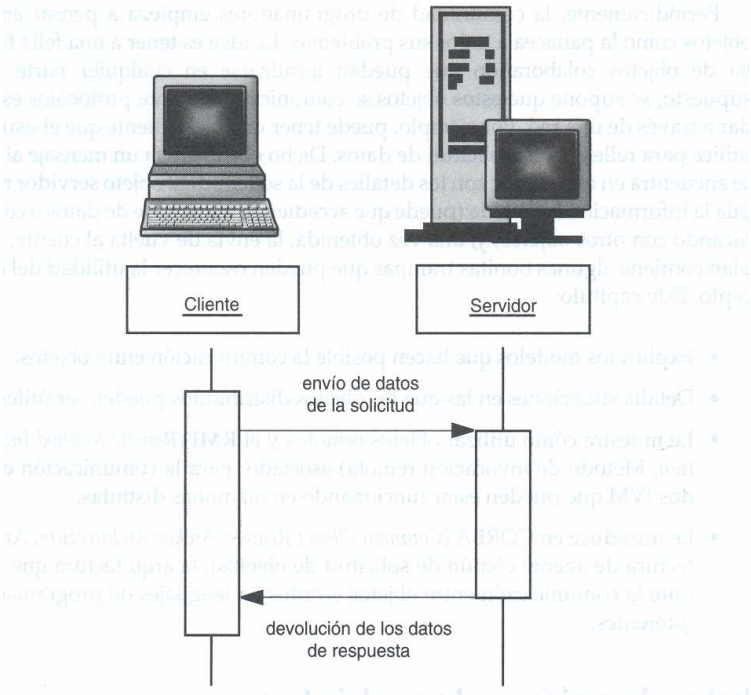
Jini utiliza RMI:
- pasa los objetos (código incluido) programados en Java.
- Se trabaja mejor con la Java Virtual Machine.
- Obtenemos serialización en el proceso.
- Hay robustez y las configuraciones de seguridad son diversas.

Entrando en tema:

Roles: cliente y servidor

Cliente: un navegador, por ejemplo
- Usuario rellena un formulario HTML, solicita info.
- (solicitud ➔ parejas nombre/valor o datos XML).
- ➔ servidor.

Servidor: programa Java
- procesar la solicitud y
- ➔ información al usuario.



nivel de interacción entre cliente y servidor
- Cliente y servidor están bien separados. **HTML, Java.**
- Cliente y servidor son programas Java. **Apto para RMI**
- El cliente es un programa Java, el objeto servidor no lo es. **Corba**

- **En esta unidad**
  - trataremos conceptos y código relativos al primer caso
  - introducción al segundo.

Introducción a INTERNET.
- conjunto de redes informáticas distribuidas por todo el mundo
- intercambian información vía protocolos TCP/IP.
- Origen: programa de investigación (DARPA, EEUU)
- Primer resultado: ARPANet (1969).
- Crece, se conecta con CSNet y MILNet: nace Internet
- muchos años es usada académicamente, intercambio de info.
- Recientemente, negocios y nuestra vida cotidiana.
- Origina replanteo de los sistemas de comunicación/información.
- mayoría soluciones vía Internet.

- Conectividad: intranets, extranets.

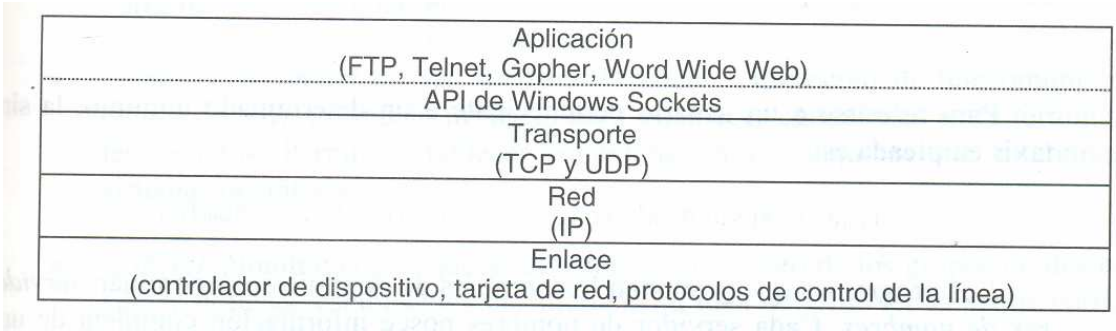
- **Intranet**, red local, usa los mismos protocolos que Internet
- Puede estar o no conectada a Internet.
- Ventajas:
  - independencia de los proveedores habituales
  - forma unificada de trabajar
  - velocidad bastante mayor que la habitual en Internet

Terminología Internet

- Internet son miles de redes informáticas
- trabajan conjuntamente bajo **los protocolos TCP/IP**
  (Transmision Control Protocol/Internet Protocol)
  (protocolo: conjunto de normas que regulan la comunicación)

- Internet está compuesto por **muchos protocolos.**
- relacionados con las **capas de servicios** de la red;
  (funciones de una red se agrupan en capas de servicios).
  capas: estaciones por las que pasa un paquete de información

TCP/IP así visto



Capa ➔ API ➔ capa
(como la API de Windows Sockets)
libera al programador de tratar con detalles

capa de enlace: empaquetar la información
capa de red: llevar los paquetes de un lugar a otro de la red.

¿Cómo se identifican estos lugares?
direcciones de Internet, identifican
- ordenador
- usuario,

direcciones especificadas según un convenio
Domain Name System (DNS).

formato DNS [subdominioJ.[subdominioJ.[...].dominio

Dominios concidos: ➔

Dominio	*Cobertura*
com	organizaciones comerciales
edu	instituciones educativas
net	suministradores de servicios de red
us	Estados Unidos
de	Alemania
es	España

Cada nombre de dominio ➔ una única dirección dirección IP.
Una dirección IP consiste en cuatro campos de 8 bits.

Por ejemplo: 200.16.20.12

- los nombres de dominio son gestionados por servidores DNS.
- Cada servidor DNS posee información:
  - **zona determinada** (subconjunto de un dominio)
  - **otros servidores** de nombres responsables de otras zonas.

servidores DNS : base de datos distribuida de nombres de Internet.

Direcciones Internet y dominios

Obtener direccion IP

```
InetAddress address =
            InetAddress.getByName("time-A.timefreq.bldrdoc.gov");
            (address = 132.163.4.102)
```

Obtener varias direcciones IP (servidores con mucho tráfico)

```
InetAddress[] addresses = InetAddress.getAIIByName(host);
```

Obtener mi dirección IP local.

```
InetAddress address = InetAddress.getLocalHost();
```

```
import java.net.*; import java.io.*;
public class InetAddressTest{
   public void direccion() throws IOException{
      BufferedReader in = new BufferedReader
            (new InputStreamReader(System.in));
      String dominio = "xxx" , auxText;
      InetAddress[] addresses;
      while(!dominio.startsWith("/")){
         try{
            System.out.println("de que dominio necesita su(s)
                        direccion(es) IP? ");
            dominio = in.readLine();
            if(dominio.length() == 0)
                auxText = "La mia, la de mi PC ";
            else{
                auxText = "Me interesan las de ";
            }
            System.out.println(auxText+dominio+", por favor");
            if (dominio.length() > 0){
               if(dominio.startsWith("/")) break;
               addresses = InetAddress.getAllByName(dominio);
               for (int i = 0; i < addresses.length; i++)
                   System.out.println("Va...("+i+1+") "+ addresses[i]);
            }
            else{
               InetAddress localHostAddress
                  = InetAddress.getLocalHost();
               System.out.println(localHostAddress);
            }
         }
         catch (Exception e){e.printStackTrace();}
      } //  while(...)
   }    //  direccion
```

```
      public static void main(String[] args){
       InetAddressTest iAT = new InetAddressTest();
       try{
            iAT.direccion();
            System.out.println("Demo finished!");
       }catch(IOException e){System.out.println("problema en lectura");}
   }
}
```

```
run:
de que dominio necesita su(s) direccion(es) IP?
Me interesan las de www.java.sun.com, por favor
Va...(0) www.java.sun.com/72.5.124.93
de que dominio necesita su(s) direccion(es) IP?
Me interesan las de www.LaNacion.com.ar, por favor
Va...(0) www.LaNacion.com.ar/200.59.146.34
de que dominio necesita su(s) direccion(es) IP?
Me interesan las de www.frc.utn.edu.ar, por favor
Va...(0) www.frc.utn.edu.ar/200.69.137.165
Va...(1) www.frc.utn.edu.ar/200.69.137.166
Va...(2) www.frc.utn.edu.ar/170.210.46.20
de que dominio necesita su(s) direccion(es) IP?
La mia, la de mi PC , por favor
tymoschuk/200.82.18.39
de que dominio necesita su(s) direccion(es) IP?
Me interesan las de /, por favor
Demo finished!
BUILD SUCCESSFUL (total time: 1 minute 13 seconds)
```

Protocolos en TCP/IP (hay muchos, los más conocidos)

- **FTP** - File Transfer Protocol
- **Gopher -** buscar y recuperar documentos.
- **POP 3** - Post Office Protocol (ejemplo mas adelante)
- **SMTP** - Simple Mail Transfer Protocol.
- **Telnet -** (Telecomunications NetWork Protocol) Terminales remotas.
- **USENet -** conjunto de los grupos de discusión y noticias.
- **HTTP** (HyperText Transfer Protocol)
  - usado por WWW (World Wide Web - La telaraña mundial).
  - sistema avanzado para la búsqueda de información en Internet basado en hipertexto y multimedia.
  - Incluye browsers(navegadores), con interfaz gráfica.

SERVICIOS EN INTERNET

- e-mail, es uno de los servicios más usados.
- telnet, sesiones con un ordenador remoto, como si fuera local
- ftp, transferir ficheros ASCII y binarios entre ordenadores conectados
  Las órdenes ftp utilizadas con mayor frecuencia son las siguientes:

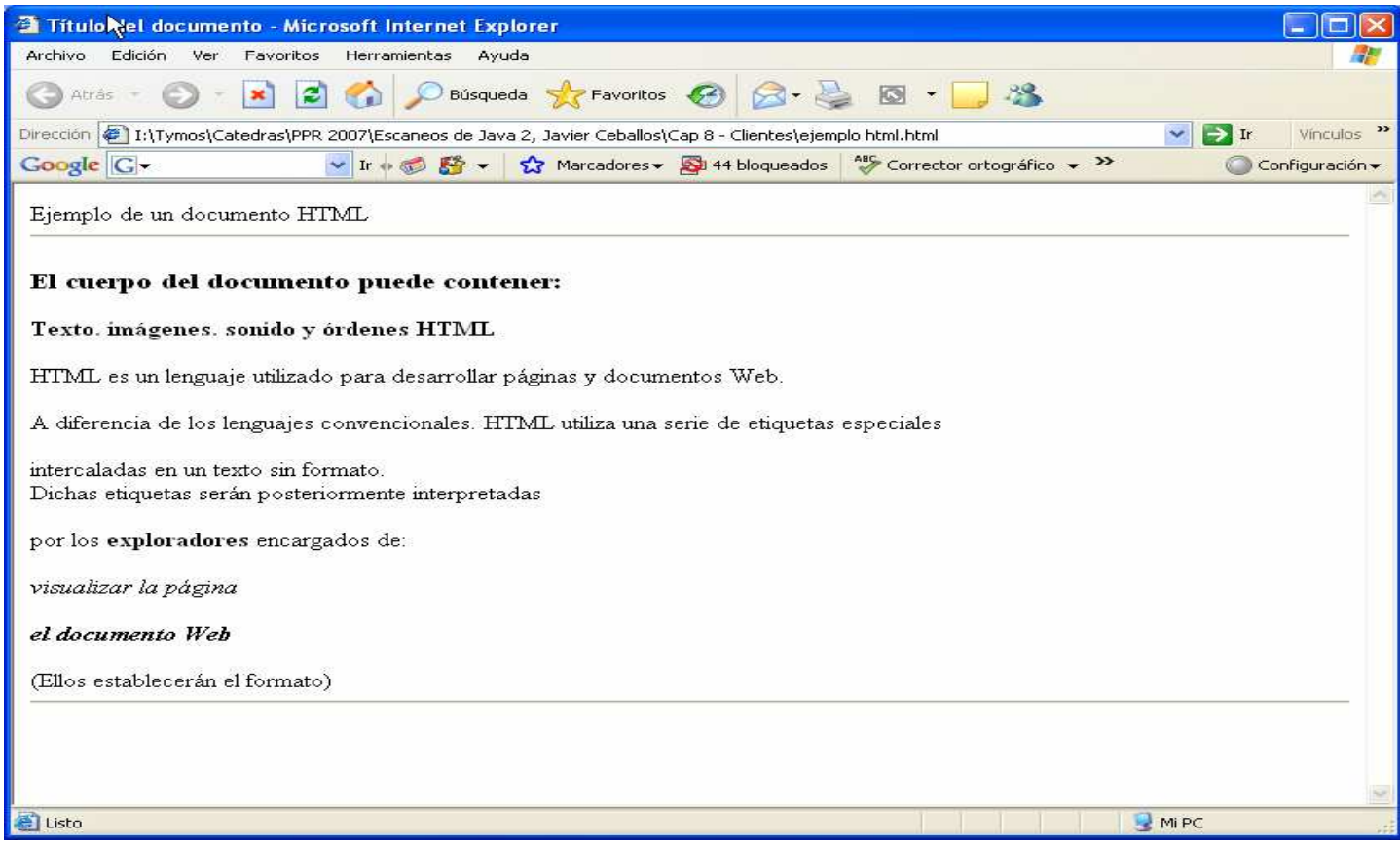
Orden ftp	*Significado*
ascii	Establece el modo ASCII para la transferencia de ficheros.
binary	Establece el modo binario para la transferencia de ficheros.
bye	Finaliza la sesión *ftp* y sale.
cd	Cambia de directorio de trabajo en el ordenador remoto.
close	Finaliza la sesión *ftp*.
ftp	Inicia una sesión *ftp*.
get	Obtiene un fichero del ordenador remoto.
help	Proporciona las órdenes *ftp* disponibles o información relativa a la orden especificada.
lcd	Cambia al directorio local de trabajo. Suele utilizarse para seleccionar los directorios al que irán a parar los ficheros transferidos.
ls	Lista el contenido de un directorio remoto.
mget	Obtiene un grupo de ficheros que pueden haberse especificado utilizando algún comodín.
mput	Envía un grupo de ficheros que pueden haberse especificado utilizando algún comodín.
open	Inicia una conexión *ftp* con el ordenador remoto especificado.
put	Envía un fichero al ordenador remoto.

- **La world wide web,** (www, Web) es el servicio mas usado.
- La Web es un sistema hipermedia interactivo
- compuesto por páginas que contienen:
  - texto,
  - imágenes,
  - sonido,
  - vídeo y
  - enlaces a otras páginas

PÁGINAS WEB,
necesitamos de HTML (HyperText Markup Language)

- HTML es utilizado para desarrollar páginas y documentos Web.
- etiquetas especiales y texto sin formato.
- etiquetas interpretadas por exploradores Web
- editar una página HTML → editor de texto sin formato
- visualizarla, → explorador Web. (no necesita conexión)
- otras herramientas: FrontPage facilita creación páginas HTML. <html>

```
<head>
   <title>Título del documento</title>
</head>
<body> <hl>Ejemplo de un documento HTML</hl>
   <hr> <h3>El cuerpo del documento puede contener:</h3>
   <h4>Texto. imágenes. sonido y órdenes HTML</h4>
   <p> HTML es un lenguaje utilizado para desarrollar
         páginas y documentos Web.
   <p> A diferencia de los lenguajes convencionales. HTML
         utiliza una serie de etiquetas especiales
   <p> intercaladas en un texto sin formato.
   <br> Dichas etiquetas serán posteriormente interpretadas
   <p>  por los <b>exploradores</b> encargados de:
   <p><i>    visualizar la página</i>
   <p><b><i>    el documento Web</i></b>
   <p>  (Ellos establecerán el formato)
   <hr>
</body>
</html>
```



URL's (Universal Resource Locutor)

- dirección descriptiva de recursos de la Web
- son una extensión de la ruta que define a un fichero (path).
- Añade un prefijo que identifica el servicio

http://www.sun.com/software/download/app_dev.html

Enlaces entre páginas

- world wide web es un servicio de información basado en **hipertexto.** (**hipertexto:** Palabra o imagen ➔(enlace, link) otra página.
- Enlace: texto subrayado

```
<a href="Huy, que dificil.html"> a ver...</a>
```

Un poco de turismo.html

```
<html>
  <head>
    <title>
      Turismo en alto nivel
    </title>
  </head>
  <body>
    <h1>Turismo arriba las nubes, mas cerca de las estrellas ...</h1>
    <hr>
    <p> Si Ud se dispone a unas vacaciones de turismo de aventuras,
    <p> Así sean ellas lo último de su vida, entonces ... <h1>El Champaqui</h1>
    <p> Deberá vadear torrentes infranqueables, llenos de rocas movedizas <a
          href="Huy, que dificil.html"> a ver...</a>
    <p> Deberá, sin previo aviso enfrentar tribus de ambrientos trogloditas<a
          href="trogloditas.html"> animo...</a>
    <p> Claro que habrá recompensas, en la alta cima, en la niebla...<a
          href="cimaJorge.html"> la gloria ...</a>
```

Turismo en alto nivel - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Atrás Búsqueda Favoritos

Dirección I:\Tymos\Catedras\PPR 2007\Unidad IV - Programación distribuida\Enlaces entre páginas\Un poco de turismo.html

Google Ir Marcadores 46 bloqueados Corrector ortográfico Enviar a

Turismo arriba las nubes, mas cerca de las estrellas ...

Si Ud se dispone a unas vacaciones de turismo de aventuras,

Así sean ellas lo último de su vida, entonces ...

El Champaqui

Deberá vadear torrentes infranqueables, llenos de rocas movedizas a ver...

Deberá, sin previo aviso enfrentar tribus de ambrientos trogloditas animo...

Claro que habrá recompensas, en la alta cima, en la niebla... la gloria ...

Y disfrutará de una camaradería sin par, aire puro y mucho ... hermanos ...

Y esa noche brindaremos, dormiremos bien ... mañana el retorno The End

Listo

Vadeando indómitos torrentes - Microsoft Internet Explorer

Archivo Edición Ver Favoritos Herramientas Ayuda

Atrás Búsqueda Favoritos

Dirección I:\Tymos\Catedras\PPR 2007\Unidad IV - Programación distribuida\Enlaces entre páginas\Huy, que dificil.html

Google Ir Marcadores 46 bloqueados Corrector ortográfico Enviar a Configuración

Todos debemos colaborar, aportar nuestro esfuerzo...

file:///I:/Tymos/Catedras/PPR%202007/Unidad%20IV%20-%20Programaci%F3n%20distribuida/Enlaces%20entre%20p%E1ginas/Un poco de turi

Mi PC

```
Huy, que difícil.html      // primer página referenciada
<html>
  <head> <title>Vadeando indómitos torrentes </title> </head>
  <body>
    <h1>Todos debemos colaborar, aportar nuestro esfuerzo...</h1>
    <hr>
       <center>
       <a href="Un poco de turismo.html">
       <img src="Huy, que dificil.jpg" align=bottom
       alt="Cruzando un arroyito">
       </a>
       </center>
    <hr>
  </body>
</html>
```

Formularios

- crear interfaces gráficas de usuario dentro de una página Web.
- Sus componentes(controles):
  - cajas de texto,
  - cajas para clave de acceso,
  - botones de pulsación,
  - botones de opción,
  - casillas de verificación,
  - menús,
  - tablas,
  - listas desplegables,

```
<form method={get|post} action="fichero para procesar los datos">
     Componentes del formulario
</form>
```

- **method** es opcional y su valor puede ser:
  get(default): comunicación por **pares clave-valor** unicamente
  post: pares clave-valor + **flujo** de un fichero
  pares clave-valor: Van inmediatamente después de action
- **action** (URL donde se procesa formulario → **<enviar>**).

Componentes del formulario

```
<input type {text|password|checkbox|radio|hidden|image|submit|reset}
     name "Variable que toma el valor">
```

Caja de texto (JTextField)

```
     El alumno:<br><input type="text" name="alumno" size="60">
```

Caja de clave de acceso (caracteres →*)

```
     Pass: <input type="password" name="clave" size="25" maxlength="20">
```

Casilla de verificación (JCheckBox) (se pueden seleccionar **varias**)

```
     <input type="checkbox" name="cvl" value="l" checked> Opción 1 <br>
     <input type="checkbox" name="cv2" value="2"> Opción 2 <br>
     <input type="checkbox" name="cv3" value="3"> Opción 3 <br>
```

Botón de opción (JRadioButtom) (se selecciona unicamente **una**)

```
     <input type="radio" name="opcion" value="l"> Opción 1 <br>
     <input type="radio" name="opcion" value="2" checked> Opción 2 <br>
     <input type="radio" name="opcion" value="3"> Opción 3 <br>
```

Parámetros ocultos

```
<input type="hidden" name="variable" value="valor">
```

Enviar datos

```
<br><br><input type="submit" value="Enviar consulta">
(Este boton puede reemplazarse por imagen)
```

Borrar los datos de un formulario

```
  <input type="reset" value="Borrar, otra...">
```

Caja de texto multilínea (JCheckBox)

```
<br><br>Mi problema:<br><textarea name="miDuda"
          rows="5" cols="40" wrap></textarea>
```

Listas desplegables (selección simple)

```
<br><br>Tenga la deferencia de atenderlo el día:<br>
<select name="día">
  <option>lunes <option>miércoles <option>jueves
</select>
```

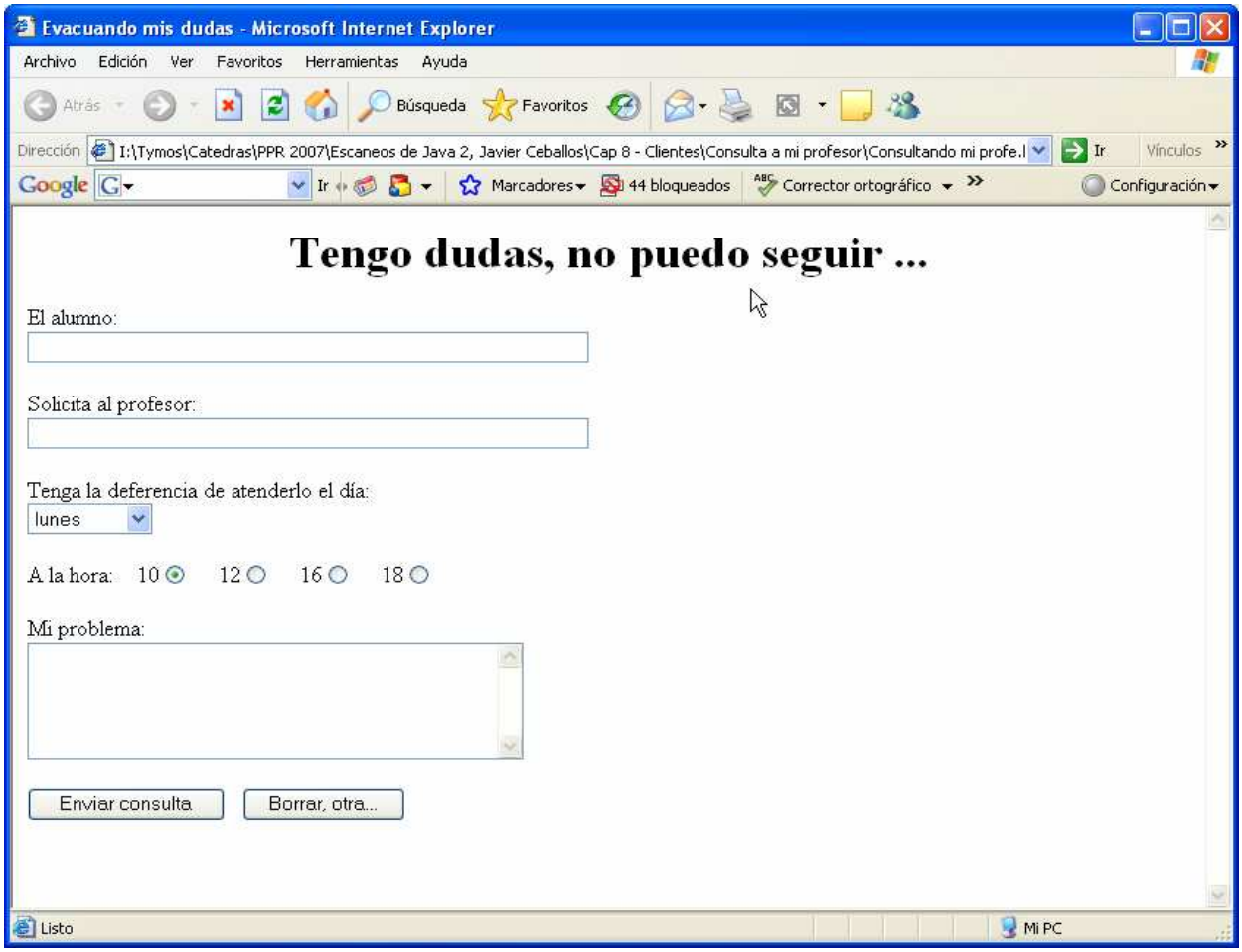
Listas desplegadas (selecciones múltiples)

```
<select name="opcion" multiple)
     <option value="1") Opción 1
     <option selected value="2") Opción 2
     <option value="3") Opción 3
</select)
```

Una consulta a mi profesor

```
<html>
  <head><title>Evacuando mis dudas</title></head>
  <body>
    <h1 align="center">Tengo dudas, no puedo seguir ...</h1>
    <form action="http:\\localhost:8080/servlet/ConsultaProfe"
          method=post>
      El alumno:<br><input type="text" name="alumno" size="60">
      <br><br>Solicita al profesor:
      <br><input type="text" name="profesor" size="60">
      <br><br>Tenga la deferencia de atenderlo el día:<br>
      <select name="día">
        <option>lunes <option>miércoles <option>jueves
      </select>
      <br><br>A la hora:   
      10<input type="radio" name="hora" value="10" checked>
         
      12<input type="radio" name="hora" value="12">
         
      16<input type="radio" name="hora" value="16">
         
      18<input type="radio" name="hora" value="18">
      <br><br>Mi problema:<br><textarea name="miDuda"
                          rows="5" cols="40" wrap></textarea>
      <br><br><input type="submit" value="Enviar consulta">
        <input type="reset" value="Borrar, otra...">
    </form>
  </body>
</html>
```

Y esto lo vemos:



HTML no es la unica forma de diseñar formularios.

XML - (Extensible Markup Language)

- convirtiéndose en el estándar para intercambio de datos en la Web.
- las etiquetas indican lo que los datos significan

fecha de nacimiento

```
<p>01/01/2004</p>          // fmto HTML
<FechaNacimiento>01/01/2004</FechaNacimiento>    // fmto XML
```

- XML crea esquema describiendo los componentes de un documento.
  - Crear un esquema XML con especificaciones
  - crear el documento utilizando el esquema.

- El esquema es un diccionario de etiquetas
- Las etiquetas describiben los elementos del documento.

```
<foto origen="Girafales.jpg">Profesor Girafales</foto>
```

- un navegador no **puede leer y mostrar un documento XML** directamente.
- **Solución:** XSTL *(Extensible Stylesheet Language Transformation)*.
- **Hoja de estilo** ➔ etiquetas de la plantilla ➔ componente XML.
- **Hoja de estilo** ➔ XSTL ➔ documento HTML

XHTML (Extensible HyperText Markup Language)

- HTML mejorado, incorpora características de XML.
- Utiliza todos los elementos de HTML + caracteristicas nuevas,

PÁGINAS WEB DINÁMICAS

- HTML ➔ página web estática. (sin remedio).

Alternativas:

- **CGI** (Common Gateway Interface): ejecutables grandes, (historia)
  - Cada petición requiere una copia, proceso independiente.

- **ISAPI** (Internet Server Application Programming Interface).
  - ISAPI trabaja sobre bibliotecas DLL.
  - Una biblioteca DLL se comparte por muchos procesos.
  - Se minimizan los requerimientos de memoria.

- **Scripts**
  - secuencias de órdenes ejecutadas dentro de la página HTML.
  - puede procesarse en el servidor o en el cliente.
  - lenguajes más comunes: JavaScript y VBScript.

- **ASP** (Active Server Page, Microsoft) (Historia)
  - contiene: texto, HTML, Scripts, y componentes *ActiveX.*
  - Obtenemos páginas dinámicas y aplicaciones Web muy potentes.
  - **Inconveniente**: propiedad de Microsoft, solamente disponible para el servidor IIS, unicamente plataformas Windows.

- **ASP.NET**
  - Es más que una nueva versión de ASP;
  - plataforma de programación Web unificada
  - proporciona todo lo necesario para aplicaciones Web.

- **Applets**, Sun Microsystems
  - pequeños programas interactivos ejecutados en un navegador.
  - vinculan código estándar Java con las páginas HTML

- **servlets**, Sun Microsystems
  - funcionalidad interactiva a los servidores.
  - Sustituyen a la programación CGI.
  - Ambas tecnologías proporcionan la misma funcionalidad básica (petición cliente ➔ programa servidor ➔ respuesta ➔ cliente)
  - petición atendida por un hilo, no por un proceso (mejor uso de memoria y tiempo de respuesta)
  - se codifican en Java, ➔ strings fmto HTML ➔ cliente
  - desarrolladores expertos en Java y HTML. (limitación)
  - mejor separar la programación de la presentación

- **JSP (Java Server Pages**, Sun Microsystems**)**
  - Tecnología para crear páginas Web con Java
  - Parecido a ASP o PHP
  - Ficheros .jsp, etiquetas HTML
  - Scriplets: etiquetas HTML con sentencias java

Conectandonos con un servidor

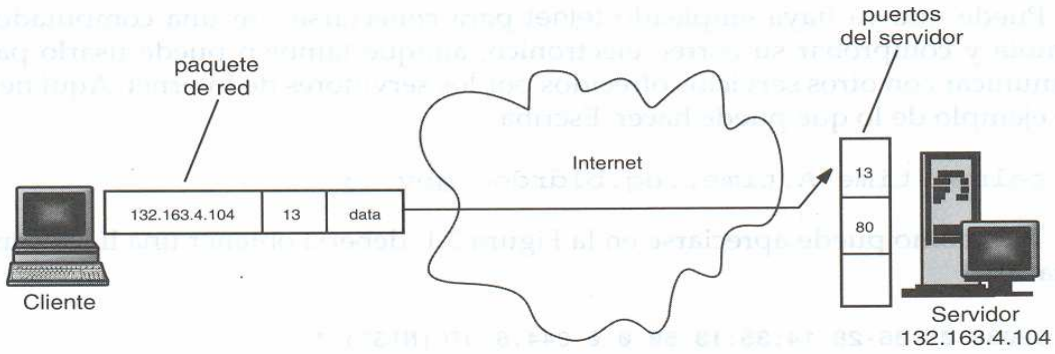
- Usando un servicio http: **Telnet**
- Como cliente, solo un comando

telnet tirne-A.tirnefreq.bldrdoc.gov 13

53923 06-07-07 10:25:24 50 0 0 0.0 UTC (NIST) *

Se ha perdido la conexión con el host

- National Institute of Standards and Technology en Boulder, Colorado, responde, con 3 hs de defasaje



- Programa servidor ejecutándose continuamente, espera cualquier tráfico de red que entre por pto 13.
- Cliente → telnet: time-A.timefreq.bldrdoc.gov pto 13, soft red →(convierte) 132.163.4.102,
- envía una petición de conexión, entrando al pto 13.
- respuesta y cierre de la conexión.

Conectandonos con un servidor, cliente Java (mismo tema)

```
import java.io.*; import java.net.*;
public class SocketTest{
   public static void main(String[] args){
      try{
         // Instanciamos objeto Socket
         Socket s = new Socket("time-A.timefreq.bldrdoc.gov",13);
         BufferedReader in = new BufferedReader // flujo de datos
            (new InputStreamReader(s.getInputStream()));
         boolean more = true;
         while (more){    //  Ciclo lectura flujo in
            String line = in.readLine();
            if (line == null)
               more = false;
            else
               System.out.println(line);
         }
      }catch (IOException e){e.printStackTrace();}
   }
}
```

53923 06-07-07 11:03:31 50 0 0 431.6 UTC(NIST) *

Implementación de servidores

- desarrollar sencillo servidor que devuelve información.
- Usamos pto 8189, no se utiliza por servicios estándar.
  ServerSocket s = new ServerSocket(8189);

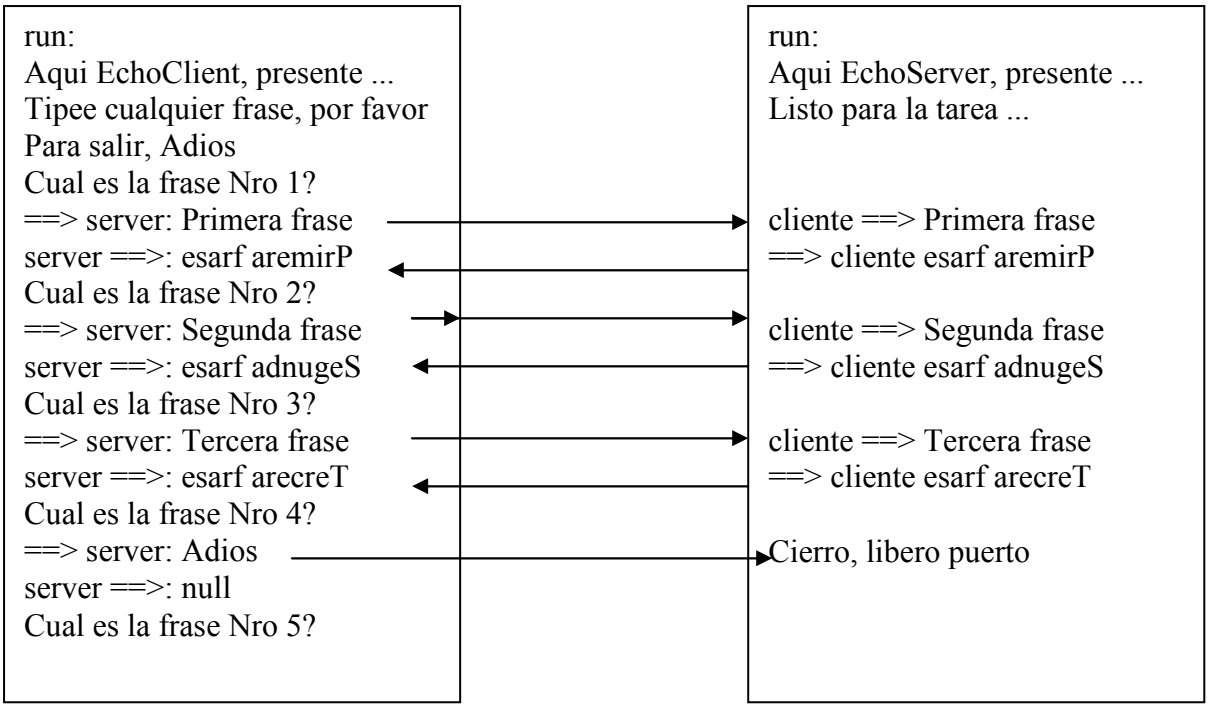
- establecemos ciclo de espera indefinido
  Socket incoming = s.accept(); //

- usamos objeto incoming para leer → cliente
  BufferedReader in = new BufferedReader (new
  InputStreamReader(incoming.getInputStream()));

- usamos objeto incoming para escribir → cliente
  PrintWriter out = new PrintWriter
  (incoming.getOutputStream(), true /* autoFlush */);

```
package javaapplication10; import java.io.*; import java.net.*;
public class EchoClient{
    //  Comunicacion con el humano
    String entrada;  // entrada teclado
    String salida;   // salida pantalla
    BufferedReader teclado = new BufferedReader(
          new InputStreamReader(System.in));//recibo del teclado
    //  Referencias - Cliente comunicandose con el servidor
    BufferedReader in;      // EchoServer → cliente
    PrintWriter   out;      // cliente → EchoServer
    Socket cliente;         // pto EchoServer

    public void tarea(){
      try{
          cliente = new Socket("Localhost",8189);
          in = new BufferedReader(new InputStreamReader
               (cliente.getInputStream()));
          out = new PrintWriter(
              cliente.getOutputStream(), true);
        int contFrases = 0;
        System.out.println("Aqui EchoClient, presente ...");
        System.out.println("Tipee cualquier frase, por favor");
        System.out.println("Para salir, Adios");
        while (true){ //  Ciclo de lectura y proceso en el servidor
           contFrases++;
           System.out.println("Cual es la frase Nro "+contFrases+"?");
           entrada = teclado.readLine();
           if (entrada == "Adios") break;  // Terminamos
           else{                   // Tengo una frase
               System.out.println("→ server: "+entrada);
               out.println(entrada);
               salida = in.readLine();
               System.out.println("server →: "+salida);
               if (entrada == "Adios")  // Terminamos
                   break;
           }   //  else
        }  //    while
        cliente.close();
      }catch (IOException e){e.printStackTrace();}
       finally{System.out.println("Conexion cerrada, nos vamos ...");}
    }//   void tarea
```

```
    public static void main(String[] args){
        EchoClient client = new EchoClient();
        client.tarea();
    }
}     // EchoClient

package javaapplication9;
import java.io.*; import java.net.*;
public class EchoServer{
    ServerSocket server; Socket incoming; String revLine;
    public String reverse(String linea){
        revLine = "";
        for(int i = linea.length()-1;i>=0;i--)
            revLine+=linea.charAt(i);
        return revLine;
    }
    public void tarea(){
        try{
            server = new ServerSocket(8189);
            incoming = server.accept();
            // cliente ➔ servidor
            BufferedReader in = new BufferedReader(new
                InputStreamReader(incoming.getInputStream()));

            // servidor ➔ cliente
            PrintWriter out = new PrintWriter
                (incoming.getOutputStream(), true);
            int contFrases = 0;
            System.out.println("Aqui EchoServer, presente ...");
            System.out.println("Listo para la tarea ...");
            boolean done = false;
            while (true){  // Ciclo de entrada / salida
                String entrada = in.readLine();
                if (entrada.trim().equals("Adios")){
                    break;
                }
                else{
                      System.out.println("cliente ==> "+entrada);
                      System.out.println("==>cliente "+reverse(entrada));
                      out.println(revLine);
                }
            }   //   while
            System.out.println("Cierro, libero puerto");
            incoming.close();//  Liberar el puerto !!!
        }
        catch (Exception e){
             e.printStackTrace();}
    }   // tarea

    public static void main(String[] args )throws IOException{
        EchoServer server = new EchoServer();
        server.tarea();
   } //  public static void main
}   //  public class EchoServer
```

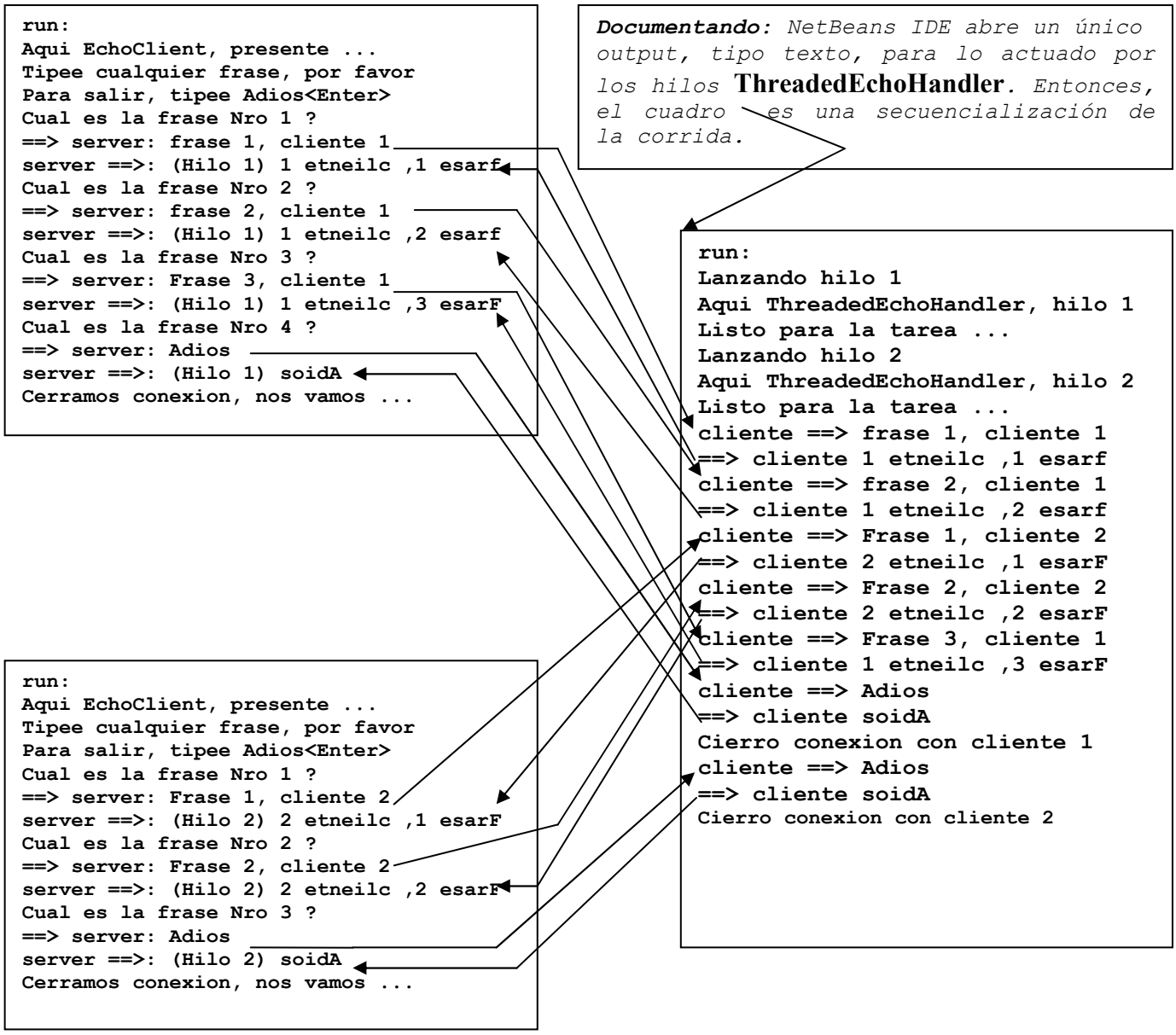


Servicio a varios clientes

- necesitamos un EchoServer que atienda n clientes simultáneamente.
- Solución: usar Threads para atender los clientes que se conectan;
- Servidor arranca un thread cada nuevo cliente peticionando:
  - **ThreadedEchoServer**, detecta nuevos clientes peticionando
  - **ThreadedEchoHandler**, atención al cliente

```
package javaapplication11;
import java.io.*;import java.net.*;
public class ThreadedEchoServer{
    ServerSocket server; Socket incoming; String revLine;
    public String reverse(String linea){
        revLine = "";
        for(int i = linea.length()-1;i>=0;i--)
            revLine+=linea.charAt(i);
        return revLine;
    }
    public void tarea(){
        int hilo = 1;   // Control de hilos
        try{
            server = new ServerSocket(8189);
            for(;;){
                incoming = server.accept( );// espera indefinidamente
                System.out.println("Lanzando hilo "+hilo);
                Thread t = new ThreadedEchoHandler(incoming, hilo,this);
                t.start(); hilo++;
            }
        }catch (Exception e){e.printStackTrace();}
    }   // tarea
    public static void main(String[] args )throws IOException{
        ThreadedEchoServer server = new ThreadedEchoServer();
        server.tarea();
   } //  public static void main
}   //  public class ThreadedEchoServer
```

```
class ThreadedEchoHandler extends Thread{
   private Socket incoming;
   private int hilo;
   private ThreadedEchoServer server;

   public ThreadedEchoHandler(Socket i, int hilo, ThreadedEchoServer
server){
      incoming = i; this.hilo = hilo; this.server = server;
   }

   public void run(){
      try{
          // Objeto flujo de entrada al servidor
          BufferedReader in = new BufferedReader
             (new InputStreamReader(incoming.getInputStream()));

          // Objeto flujo de salida del servidor
          PrintWriter out = new PrintWriter
              (incoming.getOutputStream(), true /* autoFlush */);
          int contFrases = 0;
          System.out.println("Aqui ThreadedEchoHandler, hilo "+hilo);
          System.out.println("Listo para la tarea ...");

          // Ciclo de entrada / salida
          boolean done = false;
           while (!done){

              // recibo del ciente
              String entrada = in.readLine(), revLine;
              System.out.println("cliente ==> "+entrada);
              if (entrada.trim().equals("Adios")) done = true;
              // envio al cliente
              System.out.println("==> cliente "+(revLine =
                                          server.reverse(entrada)));
              out.println("(Hilo "+hilo+") "+revLine);
              if (done){
                  System.out.println("Cierro conexion con cliente "
                                          +hilo);
                  incoming.close();//  Liberar el puerto !!!
                  break;
              }
          }   //  while
      }
      catch (Exception e){e.printStackTrace();}
   }    //  void run
}       //  class ThreadedEchoHandler
```



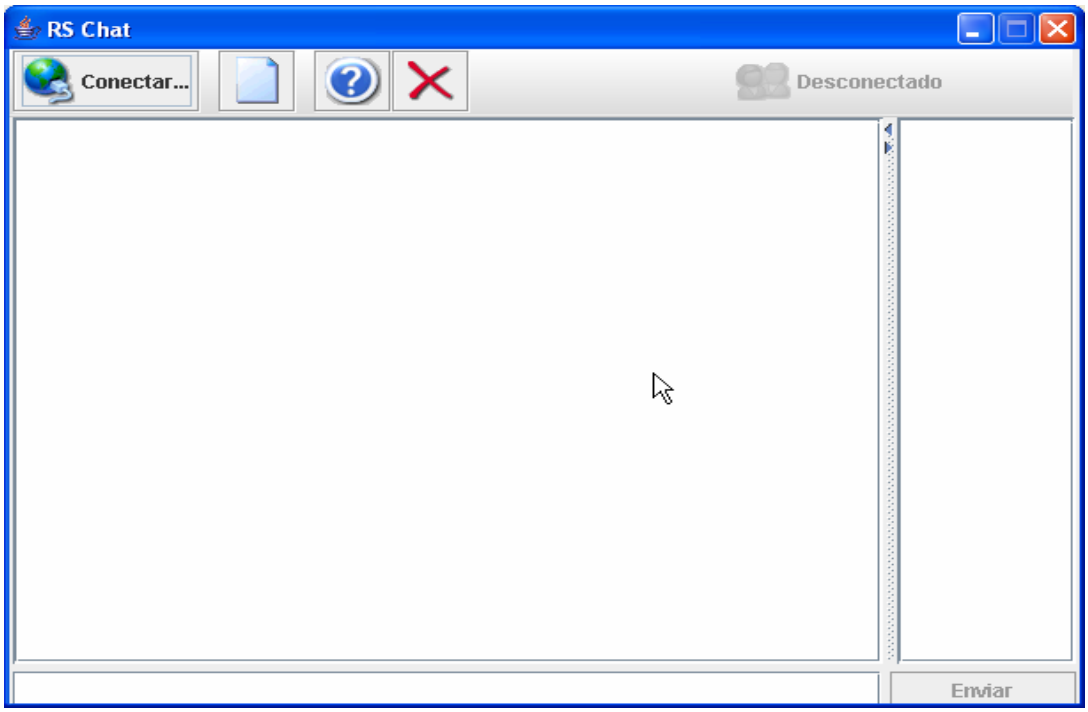
Un programa Chat. (Autor: Rodrigo Bogues)

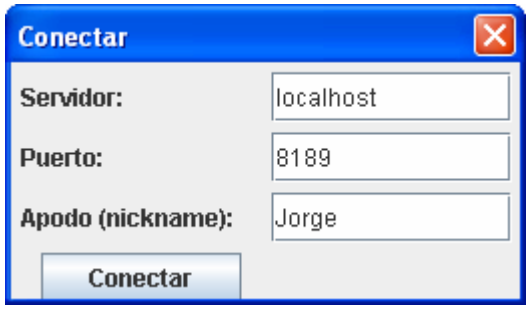
- Usamos Socket, ServerSocket, interface gráfica con eventos
- Incluye:
  - trabajar en red
  - sofisticadas interfaces gráficas
  - manejo de concurrencia
- RS Chat consta de tres clases fundamentales
  - **Servidor.class**
    - Funcionalidad de **ThreadedEchoServer**
  - **ServidorHilo.class**
    - Funcionalidad de **ThreadedEchoHandler**, (y algo mas)
  - **Cliente.class**
    - Entorno grafico para uso del Cliente

Usando el Chat

- Nos conectamos a Internet.
- Arrancamos servidor: IDE, run, run File, run "Servidor.java"
- Arrancamos cliente: IDE, run, run File, run "cliente.java".

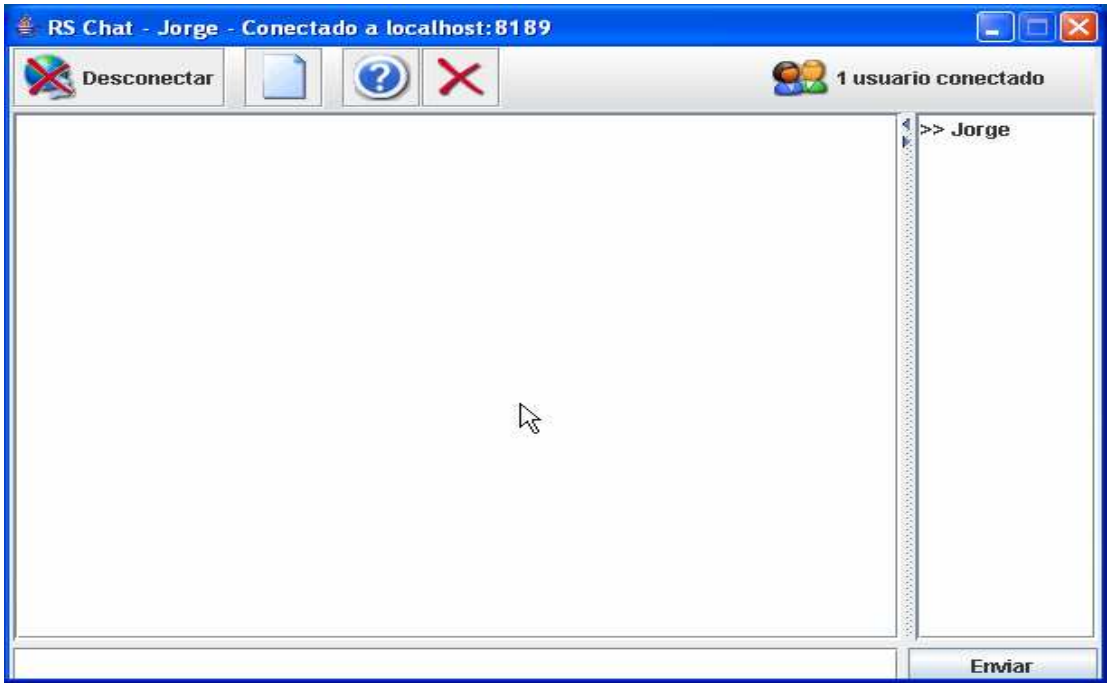
vemos





Clickeamos ícono conectar ➔

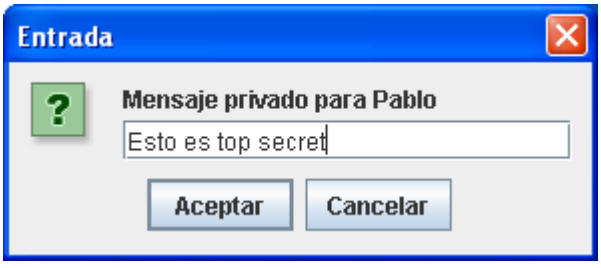
ventana cliente se ve ahora



cliente.java, a continuación.

```
import java.net.*;
import java.io.*;
import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.awt.event.*;
import java.awt.Dimension;

public class Cliente extends JFrame implements ActionListener, Runnable{

// Componentes interfase usuario
    private JTextArea txtCharla;     // Area de texto con todo el chat
    private JScrollPane scrCharla;   // Panel para area texto c/scrollBar
    private DefaultListModel lm;     // lista sobre vector dinámico que
                                     //  notifica cambios en la lista
    private JList lstConectados;     // Lista de usuarios conectados
    private JScrollPane scrConect;   // Lista desplegable
    private JTextField txtEnvio;     // Texto a enviar
    private JButton btnEnviar;       // Boton para evento enviar
    private JSplitPane splitPane;    // Panel dividido. p/2 componentes,
    private JToolBar tbBarra;        // barra de herramientas
    private JButton btnConectar;     // Boton p/pedir ventana conexion
    private JButton btnCerrar;       // Boton para cerrar conexión
    private JLabel lblUsuarios;      // Etiquetas de usuarios

// Objetos
    private String mensaje; // Texto a enviar a todos los clientes
    private String nick;    // Apodo, alias del usuario cliente
    private Socket servidor = null; // Puerto de escucha en el servidor
    private DataInputStream entrada = null; // servidor → cliente
    private DataOutputStream salida = null; // cliente → servidor
    private Thread receptor;    // Objeto receptor
    private boolean conectado = false;


    public Cliente(){   // Constructor
     btnConectar = new JButton("Conectar...",new
                           ImageIcon("conectar.gif"));
     btnConectar.addActionListener(this);
     lblUsuarios = new JLabel("Desconectado", new
               ImageIcon("usuarios.gif"),JLabel.LEFT);
     lblUsuarios.setEnabled(false);

     // Barra de tareas
     tbBarra = new JToolBar();        // Instanciamos
     tbBarra.setBounds(0,0,600,45);  //  Dimensionamos
     tbBarra.setFloatable(false);     //  No podrá ser arrastrada, movida
     tbBarra.add(btnConectar);        // Agregamos boton conectar
     tbBarra.addSeparator();        // Separamos
     JButton botonTB = new JButton(new ImageIcon("limpiar.gif"));

     botonTB.setToolTipText("Limpiar área de conversación");
     botonTB.addActionListener(new ActionListener(){
          // Respuesta al evento
          public void actionPerformed(ActionEvent e){
               txtCharla.setText("");}});
```

```
tbBarra.add(botonTB); // Lo incorporamos a la barra
tbBarra.addSeparator();    //  Separacion

botonTB = new JButton(new ImageIcon("acercaDe.gif"));
botonTB.setToolTipText("Acerca de");
botonTB.addActionListener(new ActionListener(){
     //  Respuesta al evento
     public void actionPerformed(ActionEvent e){acercaDe();}});
tbBarra.add(botonTB);      // Incorporamos a la barra

btnCerrar = new JButton(new ImageIcon("cerrar.gif"));
btnCerrar.setToolTipText("Salir");
btnCerrar.addActionListener(new ActionListener(){
     //  Respuesta al evento
     public void actionPerformed(ActionEvent e){
     if(conectado){conectado=!desconectar();}
          dispose();      //  libera recursos de este componente
          System.exit(0);
     }
});
tbBarra.add(btnCerrar);
tbBarra.addSeparator(new Dimension(150,10));
tbBarra.add(lblUsuarios);   //  Etiqueta para mensajes a usuarios

//  Area de conversación
txtCharla = new JTextArea();     //  Instanciamos
txtCharla.setEditable(false);    //  No editable (Solo salida)
txtCharla.setLineWrap(true);     //  Desplazamiento vertical
txtCharla.setWrapStyleWord(true);// Envoltura p/lineas desplazables
txtCharla.setToolTipText("Conversación en curso");
scrCharla = new JScrollPane(txtCharla);// Incorporamos a contenedor
scrCharla.setMinimumSize(new Dimension(100,50));// def. tamaño mín.

//  Lista de usuarios (Conectados)
lm = new DefaultListModel(); // Definimos modelo de lista, con ella
lstConectados = new JList(lm);  // instanciamos lista de usuarios
lstConectados.setSelectionMode(0);  // Seleccion simple
lstConectados.setToolTipText("Usuarios conectados");

 //  Mecanica envio mensajes privados entre usuarios conectados
```




```
lstConectados.addListSelectionListener(new ListSelectionListener(){
/*  Incorporamos y desarrollamos el escucha. El metodo valueChanged
 *  responde al evento cambio de selección en JList              */
    public void valueChanged(ListSelectionEvent e){
      String nickdest=lstConectados.getSelectedValue().toString();
     // A quien de los usuarios seleccionamos (p/ msge privado)
      if(nickdest!=null && !nickdest.startsWith(">> ")){
       //  Si no me seleccione (a mi mismo), ventana para dialogo
```

```
        String msg = JOptionPane.showInputDialog("Mensaje privado
                                    para "+nickdest);
        if(msg!=null && !msg.matches("")){ //si algo escribo
            //  preparo el JTextField y lo envio a todos sus escuchas
             txtEnvio.setText("/p"+lstConectados.getSelectedValue() +
                                                        "\t"+msg);
             txtEnvio.postActionEvent();
         }   //   if interno
      }      //   if externo
    }  //   valueChanged
 });   //   addListSelectionListener

 scrConect = new JScrollPane(lstConectados); // incorp. a contenedor
 scrConect.setMinimumSize(new Dimension(50,50));

  /* Usaremos un obj JSplitPane, que permite situar dos componentes:
   * izquierdo: el area de conversación y el
   * derecho: la lista de usuarios conectados.                    */

 splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
 splitPane.setLeftComponent(scrCharla);  //  comp. izquierdo
 splitPane.setRightComponent(scrConect); //  Idem derecho
 splitPane.setOneTouchExpandable(true);  //  redimensionable
 splitPane.setDividerLocation(490);      //  Posicion divisoria
 splitPane.setDividerSize(10);           //  Ancho de la divisoria
 splitPane.setBounds(2,45,600,360);      //  Dimensiones

 //  El campo de texto p/mensaje y boton enviar

 txtEnvio = new JTextField();
 txtEnvio.setBounds(2,410,490,25);
 txtEnvio.addActionListener(this);
 txtEnvio.setToolTipText("Escriba aquí su mensaje");
 btnEnviar = new JButton("Enviar");
 btnEnviar.setEnabled(false);
 btnEnviar.setBounds(497,410,105,25);
 btnEnviar.setToolTipText("Presione aquí para enviar su mensaje");
 btnEnviar.addActionListener(this);

  // Incorporamos todos los componentes a la ventana principal

 getContentPane().setLayout(null);
 getContentPane().add(splitPane);
 getContentPane().add(txtEnvio);
 getContentPane().add(btnEnviar);
 getContentPane().add(tbBarra);
 this.addWindowListener(new CierroVentana()); // Escucha ventana
 txtEnvio.requestFocus();
}   //  fin del constructor !!! (public Cliente)

public void acercaDe(){
 JOptionPane.showMessageDialog(this,
    "RS Chat v1.0\n\nDesarrollado por:\n<<
    Rodrigo Nogués >>\npara el final de PPR.",
    "Acerca de RS  Chat",
    JOptionPane.INFORMATION_MESSAGE,
    new ImageIcon("usuarios.gif"));
}
```

```
/* A seguir definimos la respuesta a eventos a nivel ventana cliente
 * (JFrame), o sea todos aquellos que no tienen su escucha definida
 * junto al componente botonTB, btnCerrar, por ejemplo)          */

public void actionPerformed(ActionEvent evt){
 if(evt.getActionCommand().equals("Desconectar")){
     conectado=!desconectar();
     return;
 }

 if(conectado){
     String texto = txtEnvio.getText();
     texto.trim();
     if (texto.length()>0){
        try{
           salida.writeUTF(texto); //  Enviamos al servidor un flujo
                    // tipo string UTF-8 (Independiente maquina)
           salida.flush(); //  Forzamos el envio de los datos por el
                           //  flujo de salida
        }catch(IOException e){;}
        txtEnvio.setText("");   //  Limpiamos textField
     }
 }   //  if(conectado)

 if(evt.getActionCommand().equals("Conectar..."))
       new ConectarDlg(this);  // ventana de dialogo para establecer
                               //   conexion con nuevo usuario
}   //  public void actionPerformed

public void run(){  // Arranca el cliente, objeto ya construido
 try{
     while(true){
        if(mensaje.startsWith("/")){// Inicia con '/', (control)
           char tipo = mensaje.charAt(1); // obtengo tipo
           mensaje=mensaje.substring(2); // recorto 2 primeras pos.
           if(mensaje.matches(nick)){ //  si mensaje contiene apodo
                mensaje=">> "+mensaje;  // antepongo ">> "
           }
           if(tipo=='c'){              // 'c' por conectar
               lm.addElement(mensaje); // usuario ➔  lista
           }
           if(tipo=='d'){      // 'd' por desconectar
               lm.remove(lm.lastIndexOf(mensaje));// usuario ➔
           }
           if(lm.getSize()==1)
                lblUsuarios.setText("1 usuario conectado");
           else
                lblUsuarios.setText(lm.getSize() + " usuarios
                                                   conectados");
           }
        else {   //  charla propiamente dicha (!control)
           txtCharla.append(mensaje + "\n");
        }
        mensaje = new String(entrada.readUTF());// leemos siguiente
     }   //  while(true)
 }catch (SocketException e){
       conectado=!desconectar();}
```

```
      catch (IOException e) {;}
    }   //  public void run()

    public boolean conectar(String ipserv, int puerto, String nick){
      try{
          lblUsuarios.setText("Conectando...");
          servidor = new Socket(ipserv, puerto);
          entrada = new DataInputStream(new
            BufferedInputStream(servidor.getInputStream()));
          salida = new DataOutputStream(new
            BufferedOutputStream(servidor.getOutputStream()));
          lblUsuarios.setText("Registrandose...");
          do{
             nick.trim();
             salida.writeUTF(nick);
             salida.flush();
             mensaje = new String(entrada.readUTF());
             if(mensaje.matches("/n")){
                nick=JOptionPane.showInputDialog(this,
                "El apodo (nickname) elegido esta siendo usado por otro
                  participante.\nElija otro por favor:","Apodo usado",
                 JOptionPane.INFORMATION_MESSAGE);
             }
          }while(mensaje.matches("/n"));
      }catch(UnknownHostException e){
             JOptionPane.showMessageDialog(this,
                 "No se pudo encontrar el servidor " +
                 ipserv + ":" + puerto, "Error de conexión",
                 JOptionPane.ERROR_MESSAGE);
             desconectar();
             return false;
      }catch(IOException e){
             JOptionPane.showMessageDialog(this,
                 "No se pudo obtener E/S de la
                  conexión", "Error de E/S", JOptionPane.ERROR_MESSAGE);
             desconectar();
             return false;
       }
      this.nick=nick;
      lblUsuarios.setText("Conectado.");
      receptor = new Thread(this); //  Le abrimos un ServidorHilo
      receptor.start();            //  Lo arrancamos
      this.setTitle("RS Chat - " +nick+
                     " - Conectado a "+ipserv+":"+puerto);
      btnConectar.setIcon(new ImageIcon("desconectar.gif"));
      btnConectar.setLabel("Desconectar");
      lblUsuarios.setEnabled(true);
      btnEnviar.setEnabled(true);
      return true;
    }   //  public boolean conectar


/* La clase interna ConectarDlg (Dialogo de conexion) que vemos a
 * continuacion se ocupa de pedir los datos(localhost, puerto, apodo)
 * de nuevo usuario que va a usar la ventana cliente que acabamos de
 * arrancar. Esa ventana la podemos arrancar desde el IDE. Por ejemplo,
 * si estamos usando NetBeans IDE, clickear Run, Run file,
 * Run "Cliente.java" y ya en la ventana, <"Conectar">.
```

```
 * Esta clase tiene tambien comportamiento para desconectar y cerrar
 * ventana                                                          */

class ConectarDlg extends JDialog implements ActionListener{
    private JLabel lblServ;
    private JLabel lblPto;
    private JLabel lblNick;
    private JTextField txtServ;
    private JTextField txtPto;
    private JTextField txtNick;
    private JButton btnConec;
    private JButton btnCancelar;
    private JDialog dlgConectar;

    public ConectarDlg(JFrame origen){
      super(origen,"Conectar", true);
      lblServ = new JLabel("Servidor:");
      lblPto = new JLabel("Puerto:");
      lblNick = new JLabel("Apodo (nickname):");
      txtServ = new JTextField();
      txtPto = new JTextField();
      txtNick = new JTextField();
      btnConec = new JButton("Conectar");
      btnCancelar = new JButton("Cancelar");
      lblServ.setBounds(5,5,120,25);
      lblPto.setBounds(5,35,120,25);
      lblNick.setBounds(5,65,120,25);
      txtServ.setBounds(130,5,120,25);
      txtPto.setBounds(130,35,120,25);
      txtNick.setBounds(130,65,120,25);
      btnConec.setBounds(15,95,100,25);
      btnCancelar.setBounds(15,95,100,25);
      btnConec.addActionListener(this);
      btnCancelar.addActionListener(this);
      txtServ.addActionListener(this);
      txtPto.addActionListener(this);
      txtNick.addActionListener(this);
      setSize(260,150);
      setResizable(false);
      setLocationRelativeTo(origen);
      getContentPane().setLayout(null);
      getContentPane().add(lblServ);
      getContentPane().add(lblPto);
      getContentPane().add(lblNick);
      getContentPane().add(txtServ);
      getContentPane().add(txtPto);
      getContentPane().add(txtNick);
      getContentPane().add(btnConec);
      getContentPane().add(btnCancelar);
      show();
    }   //  public ConectarDlg

    public void actionPerformed(ActionEvent evt){
      if(evt.getActionCommand().equals("Conectar")){
         if(txtServ.getText().length()==0
          || txtPto.getText().length()==0
          || txtNick.getText().length()==0){
               JOptionPane.showMessageDialog(this,
```

```
                "Debe ingresar todos los datos.");
        }
        else{
           int puerto;
           try{
              puerto = Integer.parseInt(txtPto.getText());
           }
           catch(NumberFormatException e){
              JOptionPane.showMessageDialog(this,
                      "El puerto debe ser un número entero.");
              return;
           }

           conectado=conectar(txtServ.getText(),puerto,txtNick.getText())
           ;
           dispose();
        }   //  if(txtServ.getText().length()==0
      }
      else if(evt.getActionCommand().equals("Cancelar"))dispose();
           else{
             if(txtServ.isFocusOwner())          txtPto.requestFocus();
             else if(txtPto.isFocusOwner())      txtNick.requestFocus();
                  else                       btnConec.requestFocus();
           }
       }   //  if(evt.getActionCommand()
    }   //  public void actionPerformed

    public boolean desconectar(){
      try{
          servidor.close();
          entrada=null;
          salida=null;
          servidor=null;
          receptor=null;
          lm.clear();
          this.setTitle("RS Chat");
          btnConectar.setIcon(new ImageIcon("conectar.gif"));
          btnConectar.setLabel("Conectar...");
          lblUsuarios.setEnabled(false);
          lblUsuarios.setText("Desconectado");
          btnEnviar.setEnabled(false);
          return true;
       }
      catch (Exception ex){return false;}
    }   //  public boolean desconectar()
    class CierroVentana extends WindowAdapter{
           public void windowClosing(WindowEvent e){
                btnCerrar.doClick();
           }
    }   //   class CierroVentana
    public static void main(String []args){
      Cliente ventana = new Cliente();
     ventana.setSize(610,463);
      ventana.setResizable(false);
      ventana.setTitle("RS Chat");
      ventana.setVisible(true);
    }
}   // public class Cliente
```

Ahora, la clase servidor. Es muy simple, por suerte…

```
import java.net.*;
import java.io.*;
import java.util.*;
public class Servidor{
    public static void main (String[] args) throws IOException{
      ServerSocket servidor=null;
      boolean escuchando = true;
      try{
            servidor = new ServerSocket(8189);
      }catch (IOException e){
            System.out.println("No se puede escuchar en el puerto " +
                                                                  8189);
            System.exit(1);
      }
      while (escuchando){
            Socket cliente=null;
            try{
                cliente = servidor.accept();
            }catch (IOException e){
                System.out.println("Fallo la conexion: " + args[0]
                                                        + ", " +
                e.getMessage());
                continue;
            }
            System.out.println("Se conecto " +
                                cliente.getInetAddress());
            new ServidorHilo(cliente).start();
      }
      try{
           servidor.close();
           }catch (IOException e){
                System.err.println("No se pudo cerrar el servidor. " +
                e.getMessage());
      }
    }   //  public static void main
}   //  public class Servidor
```

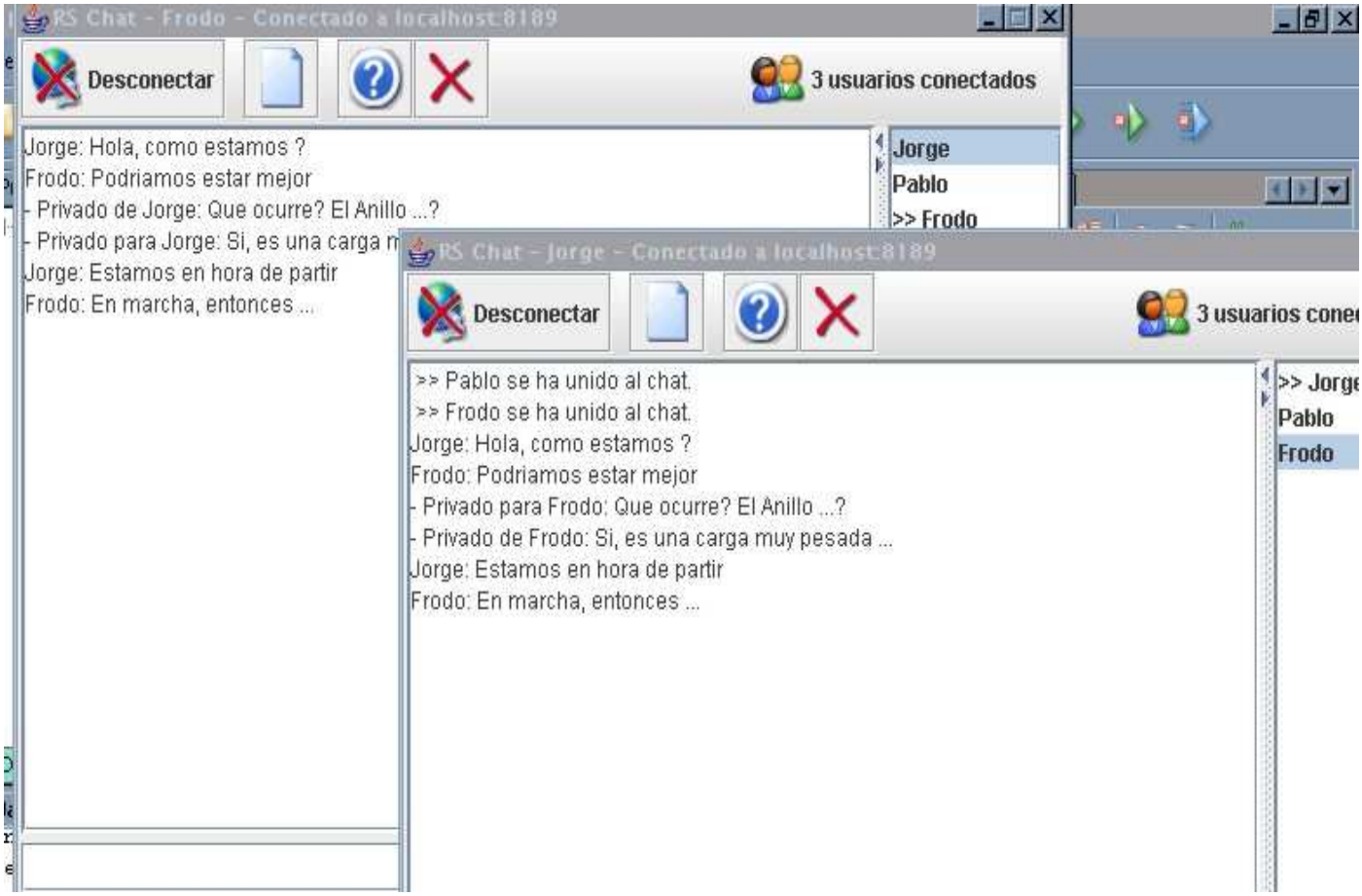
La clase servidor starta ServidorHilo, tambien bastante sencilla.

```
import java.net.*;
import java.io.*;
import java.util.Vector;
import java.util.Enumeration;

public class ServidorHilo extends Thread{
    private Socket cliente = null;
    private DataInputStream entrada = null;
    private DataOutputStream salida = null;
    private static Vector conectados = new Vector();
    private String nick;

    //  Se establecen los flujos de entrada/salida con el cliente

    public ServidorHilo(Socket cliente) throws IOException{
           super(new String(cliente.getInetAddress().toString()));
           this.cliente=cliente;
           entrada = new DataInputStream(new
                     BufferedInputStream(cliente.getInputStream()));
```

```
        salida = new DataOutputStream(new
                    BufferedOutputStream(cliente.getOutputStream()));
    }

    public void run(){
     try{
         boolean n=true;
         do{
            nick = new String(entrada.readUTF()); // quiere conectarse?
            ServidorHilo ni = buscarnick(nick); // apodo existe?

            /* buscarnick retorna el objeto ServidorHilo en el cual ya
             * tenemos al apodo (nick) ya conectado o null
             * caso contrario                                        */

            if(ni!=null) transmitir("/n",this); // ya te tenemos ...

            /* Ya tenemos alguien así apodado, los parámetros a trans-
             * mitir son el mensaje "/n" y objeto ServidorHilo null.
             * Con ello transmitir activa comportamiento del cliente que
             * hace la correspondiente advertencia...                  */

          else n=false;
         }while(n);  // Salimos si el apodo (nick) es nuevo
         transmitir(" >> " + nick + " se ha unido al chat.",null);
         transmitir("/c" + nick, null);  // Indicamos la nueva conexion
         conectados.addElement(this);    // Incorporamos en la lista
         transmitir("", this);
         while(true){    //  Ciclo de atencion al cliente
           String ent = new String(entrada.readUTF()); // Que nos envia ?
           if(ent.startsWith("/p")){   //  Control, (msge privado)
                /* Buscamos el destinatario del mensaje, está
                *inmediatamente despues de /p, o sea la posición 2,
                * termina con \t                                     */

              String nickdest=ent.substring(2,ent.indexOf("\t"));

              /* El mensaje está a continuación del destinatario, o sea
               * posicion de "\t" + 1, hasta el fin de la cadena      */

               String msg=ent.substring(ent.indexOf("\t")+1);
               // Busquemos al destinatario de ese mensaje privado
               ServidorHilo dest=buscarnick(nickdest);
               if(dest==null)  // O sorpresa, no existe (Esto no ocurre)
                 transmitir(">> No se pudo entregar el mensaje privado a
                                        "+nickdest,this);
               else{           //  Lo tenemos, comuniquemonos con el
                 transmitir("- Privado de "+ nick + ": " + msg, dest);
                 transmitir("- Privado para "+ nickdest + ": " + msg,
                                                          this);
               }
               ent="";
           }    //   Fin del tratamiento mensaje privado
           else    // No es un mensaje privado, comunicaremos c/todos
                transmitir(nick + ": "+ ent,null);
         }   // while(true)
     }       // try
     catch (IOException e) {;}
```

```
      finally{
             conectados.removeElement(this);
             transmitir("/d" + nick,null);
             transmitir(" >> " + nick + " ha dejado el chat.",null);
             System.out.println("Se desconecto "+
                                          cliente.getInetAddress());
             try{
                 entrada.close();
                 salida.close();
                 cliente.close();
             }catch (IOException e) {;}
      }
    } // public void run

    public ServidorHilo buscarnick(String n){

     /* A continuación vemos un ejemplo de uso de comportamiento de la
      * clase Enumeration. Ya la hemos usado en la U I, la conocemos.
      * Aquí la estamos usando para recorrer la lista de usuarios ya
      * conectados al chat                                          */

       Enumeration con = conectados.elements();
       while(con.hasMoreElements()){
           ServidorHilo cl = (ServidorHilo) con.nextElement();
           if(n.matches(cl.nick)){return cl;} //  Ya tenemos ese (nick)
       }
       return null;    //  El apodo, alias (nick) es nuevo
    }

    private static synchronized void transmitir(String mensaje,
      ServidorHilo sh){
      Enumeration conect = conectados.elements();
         while(conect.hasMoreElements()){
             ServidorHilo ctes = (ServidorHilo) conect.nextElement();
             if(sh==null){   // este alias ya existía
                 try{
                     ctes.salida.writeUTF(mensaje);
                     ctes.salida.flush();
                 }
                 catch (IOException e){ctes.stop();}
             }
             else{
                 try{
                     if(mensaje==""){
                         sh.salida.writeUTF("/c"+ ctes.nick);
                         sh.salida.flush();
                     }
                     else{
                         sh.salida.writeUTF(mensaje);
                         sh.salida.flush();
                         break;
                     }
                 }   // try
                 catch(IOException e){;}
             }   // if((sh==null)
      }   // while(conect.hasMoreElements())
    }   // private static synchronized void transmitir
}   // public class ServidorHilo
```

- *A continuación, un mini chat con tres usuarios.*
- *proyecto ya abierto en el NetBeans IDE*
- *soft de red previamente cargado (Fullzero, Tutopia,etc),*
- *Abrimos los tres fuentes (cliente, Servidor, ServidorHilo*
- *Editamos Servidor y <run>, <run file>, run Servidor.java*
- *Editamos Cliente y <run>, <run file>, run Cliente.java"*
- *<Conectar>,localhost,8189,Jorge, <Conectar>*
- *<run>, <run file>, run Cliente.java"*
- *<Conectar>,localhost,8189,Pablo, <Conectar>*
- *<run>, <run file>, run Cliente.java"*
- *<Conectar>,localhost,8189,Frodo, <Conectar>*
- *nos vamos posicionando en las distintas ventanas cliente*
- *y enviamos mensajes.*



Conexiones URL

```
import java.net.*;
import java.io.*;
public class ParseURL{     // Decodificar URL's
    public static void main(String[] args){
        String urlAux;
        try{
            System.out.println("Que URL desea Ud investigar?");
            urlAux = In.readLine();
            System.out.println("Investigamos "+urlAux);
            URL aURL = new URL(urlAux);
            System.out.println("protocol = " + aURL.getProtocol());
            System.out.println("host = " + aURL.getHost());
            System.out.println("filename = " + aURL.getFile());
            System.out.println("port = " + aURL.getPort());
            System.out.println("ref = " + aURL.getRef());
            System.out.println("--------------");
        }
        catch (IOException exception){exception.printStackTrace();}
    }
}
```

Un ejemplo a investigar

```
"http://java.sun.com:80/docs/books/"
             + "tutorial/index.html#DOWNLOADING"

Que URL desea Ud investigar?
Investigamos
http://java.sun.com:80/docs/books/tutorial/index.html#DOWNLOADING·
protocol = http
host = java.sun.com
filename = /docs/books/tutorial/index.html
port = 80
ref = DOWNLOADING·
--------------
```

otro ejemplo

```
http://docs.mycompany.com/api/java/lang/String.html

Que URL desea Ud investigar?
Investigamos
http://docs.mycompany.com/api/java/lang/String.html
protocol = http
host = docs.mycompany.com
filename = /api/java/lang/String.html
port = -1
ref = null
--------------
```

URLConnection, recuperar información

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

/**
 *   Este programa conecta con el URL solicitado por el operador,
 *  o con java.sun.com si no se informa nada. Exhibe datos de
 *  cabecera y diez primeras líneas archivo definión URL        */

public class URLConnectionTest{
   public static void main(String[] args){
      String urlName  = null;
      try{
         System.out.println("Que URL desea Ud investigar?");
         urlName = In.readLine();
         if (urlName == null)urlName  = "http://www.java.sun.com";
         System.out.println("Investigaremos ==> "+urlName);
         URL url = new URL(urlName);
         URLConnection connection = url.openConnection();

         int n = 1; String key;
         System.out.println("Van campos de cabecera");
         while ((key = connection.getHeaderFieldKey(n)) != null){
            String value = connection.getHeaderField(n);
            System.out.println(key + ": " + value);
            n++;
         }
         System.out.println("Fin campos de cabecera");
         System.out.println("");
         System.out.println("Van 10 lineas archivo definicion del URL");
         BufferedReader in = new BufferedReader(new
            InputStreamReader(connection.getInputStream()));
         String line;
         n = 1;
         while ((line = in.readLine()) != null && n <= 10){
            System.out.println(line);
            n++;
         }
         if (line != null)
             System.out.println("Fin 10 lineas archivo de definicion");
      }catch (IOException exception){exception.printStackTrace();      }
   }
}
```

Una primera ejecución

```
run:
Que URL desea Ud investigar?
Investigaremos ==> http://www.fcm.unc.edu.ar
Van campos de cabecera
Date: Sat, 10 Mar 2007 01:49:40 GMT
Server:   Apache/2.0.55   (Ubuntu)   PHP/4.4.2-1build1   mod_ssl/2.0.55
OpenSSL/0.9.8a
Last-Modified: Thu, 28 Dec 2006 15:12:46 GMT
ETag: "aab8-1a9d-425ab94631780"
Accept-Ranges: bytes
Content-Length: 6813
Keep-Alive: timeout=15, max=500
Connection: Keep-Alive
Content-Type: text/html; charset=ISO-8859-1
Fin campos de cabecera
```

```
Van 10 lineas archivo de definicion del URL
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Facultad de Ciencias Médicas</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

<body bgcolor="#9FBDB1" link="#000000" vlink="#000000" alink="#0000CC">
<table width="817" height="1032" border="0">
  <tr>
Fin 10 lineas archivo de definicion
```

Una segunda

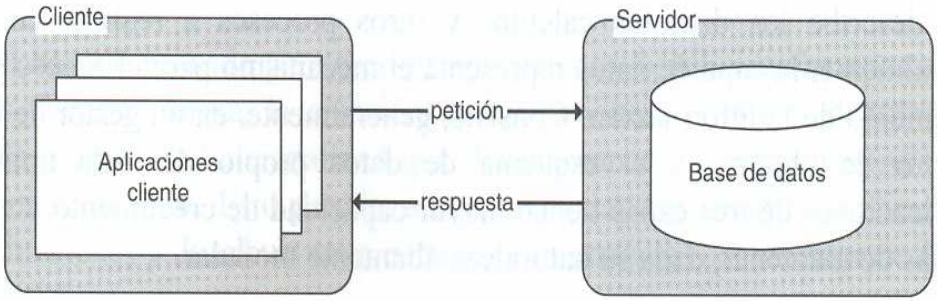
```
run:
Que URL desea Ud investigar?
Investigaremos ==> http://www.frc.utn.edu.ar
Van campos de cabecera
Date: Sat, 10 Mar 2007 01:52:21 GMT
Server: Microsoft-IIS/6.0
pragma: no-cache
cache-control: private
Content-Length: 46042
Content-Type: text/html
Expires: Fri, 09 Mar 2007 01:52:20 GMT
Set-Cookie: ASPSESSIONIDCSBDTBAS=GONBIDABKAJJJFKLPNMOKPIM; path=/
Cache-control: no-cache
Fin campos de cabecera

Van 10 lineas archivo de definicion del URL

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html><!-- InstanceBegin template="/Templates/Principal.dwt.asp"
codeOutsideHTMLIsLocked="false" -->
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"
/>
<meta name="keywords" content="UTN-FRC, UTN, ARGENTINA, FRC, Universidad
Tecnológica Nacional, Universidad Tecnológica Nacional - Facultad
Regional Córdoba, Facultad Regional Córdoba, U.T.N., F.R.C, UTN -
Facultad Regional Córdoba, U.T.N. - Facultad Regional Córdoba,
ARGENTINE, CIUDAD DE CORDOBA, CORDOBA, CORDOBA, CIUDAD DE CORDOBA" />
<meta name="description" content="UTN FRC, casa de altos estudios de la
ciudad de Córdoba. P&aacute;gina Web de la Universidad Tecnológica
Nacional - Facultad Regional Córdoba" />
<meta                                                    name="verify-v1"
content="+iYX1Z0WEddeqeNwGIgtu68PXg0qXJcPRedckxgD6Xk=" />
<!-- InstanceBeginEditable name="doctitle" -->
<title>Universidad Tecnol&oacute;gica Nacional - Facultad Regional
C&oacute;rdoba</title>
Fin 10 lineas del archivo de definición
```

J2EE – (Java 2 Enterprise Edition)

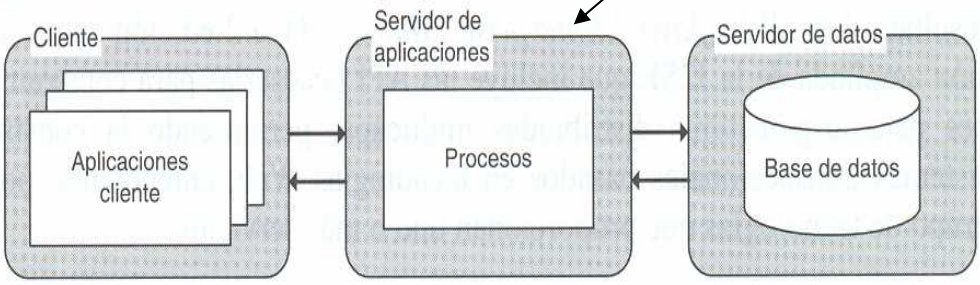
- Java J2SE permite construir aplicaciones Java robustas,
- no resuelve **eficientemente** aplicaciones distribuidas.
- Entonces Sun Microsystems ➔ **J2EE (Java 2 Enterprise Edition)**, (versión ampliada de la J2SE)
  - API's para aplicaciones arquitectura distribuidas multicapa,
  - sistemas transaccionales basados en tecnologías Web, (capa: grupo de tecnologías que proporcionan servicios)

INTRODUCCIÓN

- Tradicionalmente se utilizó **modelo cliente/servidor.**(**3** componentes)
  - interfaz de usuario, (cliente)
  - lógica de negocio (cliente) o (cliente y servidor)
  - y gestión de datos. (servidor)
  - resultado: **arquitectura de dos capas de software.**



- la lógica de negocio ➔ servicio ➔ servidor,
- **arquitectura de tres capas:**



- **Cliente** (**capa de presentación)** interfaz con el usuario.
- **Servidor de aplicaciones:**
  - **capa de lógica de negocio**
  - se modela el comportamiento del sistema,
  - usa los datos provistos por la base de datos,
  - actualizándolos según sea necesario.
- **Servidor de datos:**
  - **capa de datos**
  - mecanismo para el acceso
  - y el almacenamiento de la información.

- mayor capacidad de crecimiento
- son más sencillas de mantener, (por ser altamente modulares)

Un ejemplo: (tres componentes)

- Una persona **(cliente)** solicita a un
- agente de viajes que le organice **(aplicación)** sus vacaciones.
- el agente ➔ contacta hoteles, ➔ líneas aéreas, ➔ restaurantes, ➔ alquiler de coches, **(datos)**

J2EE ➔ **aplicaciones distribuidas** basadas en componentes.

- Componente modulariza, encapsula la funcionalidad.
- desarrolladores dividen la aplicación según sus funciones
- funciones ➔ componentes del servidor de J2EE.

Qué es una aplicación empresarial? Brevemente:

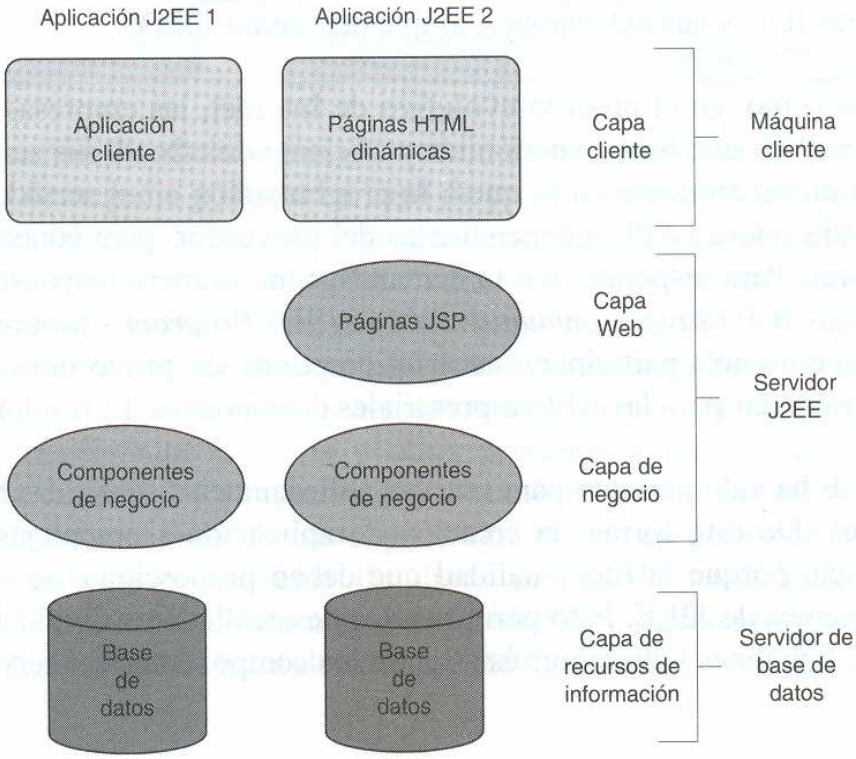
- Utilizada concurrentemente por muchos usuarios.
- Utiliza recursos distribuidos (distintas bases de datos).
- Las tareas las realizan distintos objetos distribuidos.
- Utiliza tecnología J2EE para enlazar componentes distribuidos.

Middleware.

- software que comunica componentes de un sistema distribuido
- software gestiona interacciones a nivel de aplicación
- localizado entre aplicación y un sistema de menor nivel (sistema operativo, DBMS, servicios de red, etc.).

ARQUITECTURA J2EE MULTICAPA

- un modelo de aplicación distribuida multicapa
- La lógica de negocio ➔ componentes según su función,
- componentes ➔ instalados en diferentes máquinas



Capa cliente. (presentación)
- interfaz con el usuario
- peticiones ➔ componentes de la capa Web.

Capa Web.
- componentes se ejecutan en el servidor J2EE
- utilizan HTTP para peticiones y respuestas.

Capa de negocio.
- componentes se ejecutan en el servidor J2EE.
- Contienen la lógica de negocio
- interactúa con la capa de recursos de información.

Capa de recursos de información (middelware).
- bases de datos,
- sistemas de terceros)
- archivos

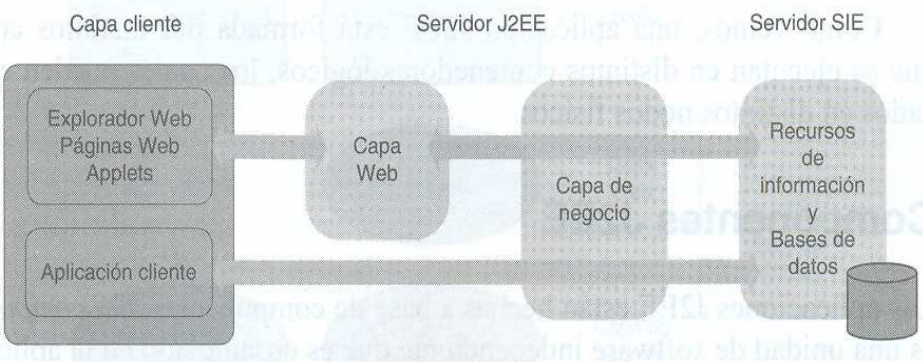
Concepto de componente en J2EE
- Las aplicaciones J2EE están armadas con componentes.
- Un componente J2EE es una unidad de software independiente
- se comunica con otros componentes a través de canales definidos.

Estos **componentes** son, según La especificación J2EE:
  - Aplicaciones cliente y applets.
  - Servlets y Java Server Pages (JSP).
  - Enterprise JavaBeans (EJB), contienen las reglas de negocio.

- Un **cliente J2EE** (o capa cliente) puede ser:
  - una aplicación cliente (JFrame)
  - un cliente Web, compuesto por
    - páginas Web dinámicas generadas por componentes capa Web (Applets, o mejor JSP)
    - explorador Web que presenta las páginas recibidas desde el servidor.

Comunicación capa cliente ➔ restantes capas (figura abajo)
- **Cliente ➔ capa de negocios**
  - directamente, (Ej. Aplicación Java JFrame)
  - a través de la capa Web (Applet o HTML sobre explorador)
- **Capa de negocios ➔ Capa de recursos de información**
  - directamente

(Luego tenemos comunicación inversa, es simétrica)



Componentes de la capa Web

- **servlets**: clases escritas en Java
  - procesan peticiones cliente obteniendo datos
  - que envían a una página JSP
  - para que se los transmita al cliente.
- **Páginas JSP**: documentos de texto
  - se ejecutan como los servlets,
  - forma más natural de crear contenido dinámico.

Componentes de la capa de negocio.

- elemento vital de la arquitectura J2EE, proporcionan:
  - concurrencia,
  - escalabilidad,
  - gestión de ciclo de vida y
  - tolerancia a fallos.
- son los **enterprise JavaBeans** (EJB).
  - **beans de sesión** (session beans), conversación transitoria, sin guardar datos.
  - **beans de entidad** (entity beans); **transacción en firme**, datos almacenados.
  - **beans orientados a mensajes** (message-driven beans); combina características de un bean de sesión y un JMS.

Componentes de la capa de recursos de información

- conexión con aquellos recursos que no están dentro de J2EE
  - DBMS
  - servidores ya existentes.

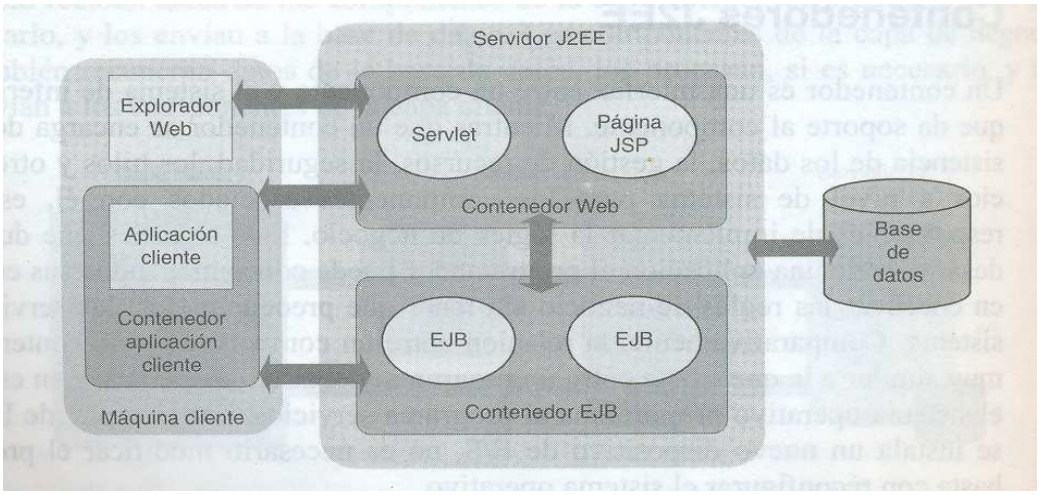
Contenedores J2EE

- interfaz entre un componente y el sistema de nivel inferior.
  - persistencia de los datos,
  - la gestión de recursos,
  - la seguridad,
  - multithreading,
  - otros servicios a nivel de sistema
  - todo componente debe ser
    - ensamblado en una aplicación J2EE
    - y desplegado en su contenedor.

Tipos de Contenedores J2EE

- Servidor J2EE. Es el motor de ejecución.
- Proporciona los contenedores Web y EJB.
  - Contenedor Web. Gestiona JSP y servlets.
  - Contenedor EJB. Gestiona los Enterprise Java Beans
- Contenedor aplicación cliente. Gestiona apl. cliente.
- Contenedor de applets. (Navegador Web)

Gráficamente:



Distribución

- aplicación java: archivo .jar (Java Archive),
- aplicación J2EE lo hace en fichero:
  - .war (Web ARchive),
  - .ear (Entreprise ARchive) o
  - .jar (Java ARchive).
  - .XML (**Descriptor de despliegue**, ej. web.xml)

Responsabilidad de la Capa Cliente

- interfaz gráfica al usuario; soluciones posibles.
  - formulario HTML o XML; (alto tiempo de respuesta)
  - programa Java, (JApplet o Júrame); (Buen tiempo respuesta)
  - Definida lado del servidor.(mal tiempo de respuesta

Responsabilidad de la Capa Web

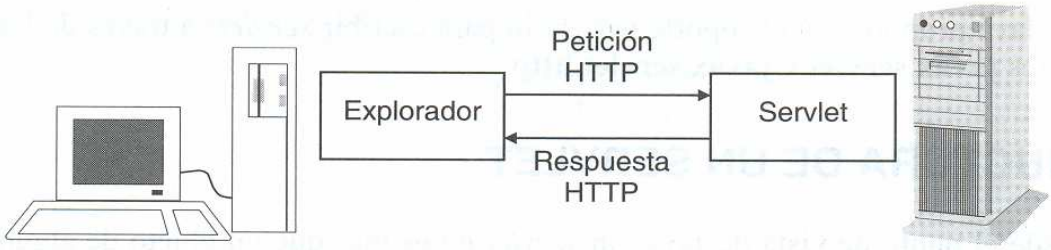
- componentes (servlets y JSP) ➔ clientes.
- peticiones ➔ servlets o JSP, vía Tomcat
- componentes se identifican vía URL asociados a un link.
  - link hubicado en página Web ➔ cliente.
  - cliente ➔ link, explorador ➔ componente Web.
  - componente web genera página ➔ cliente.
- **Usando servlets y JSP**,
  - separar presentación y procesamiento,
    - presentacion en JSP
    - procesamiento en servlet.
- controlar el acceso del cliente a los recursos.
  - medidas de seguridad de los DBMS o
  - un componente controlador (una JSP, un servlet, o un EJB),
    - definan de que menú de opciones dispondrá
    - arranquen un proceso determinado.
- evitar peticiones duplicadas de los clientes.

La capa EJB (Enterprice Java Beans)

- Permite construir componentes,
  - que contengan lógica de negocio
  - o que representen datos,
  - distribuidos entre distintas máquinas.

¿QUÉ ES UN SERVLET?

- programa que se ejecuta en el cont. Web de cont J2EE.
- puede invocarse vía HTTP.
- Navegador Web ➔ applet
- Contenedor Web ➔ servlet

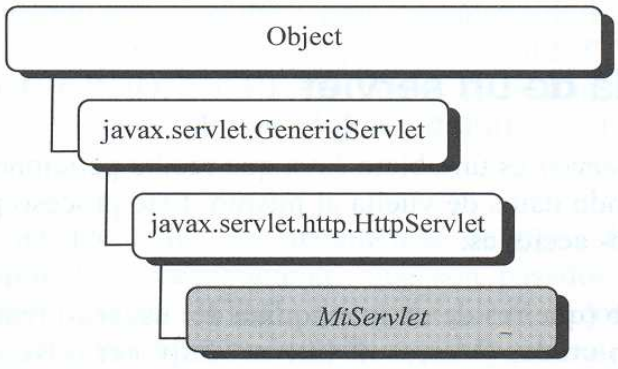


Características de un servlet

- alternativa de Sun Microsystems para sustituir CGI.
- Ambas tecnologías ➔ misma funcionalidad básica, diferencia:
- Cada petición cliente arranca:
  - un nuevo proceso (CGI)
  - un nuevo hilo (Servlet)
- son independientes de la plataforma.
- Consumen menos recursos (que CGI)
- más rápidos que CGI y scripts
- Son seguros y portables debido:
  - que se ejecutan bajo la máquina virtual de lava,
  - al mecanismo de excepciones de lava, y
  - al uso del administrador de seguridad de Java
  - envían los resultados en HTML.
  - Pueden interactuar con aplicaciones lava o applets.

ESTRUCTURA DE UN SERVLET

- es un objeto de alguna de las clases de la API Java
  - ➔ servicio genérico, ➔ GenericServlet.
  - ➔ servicios específicos, HttpServlet.



Un servlet http básico

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
public class MiServlet extends HttpServlet{
      public void init(ServletConfig config) throws ServletException{
            super.initCconfig); // ...
      }

      // Método doPost para responder a una petición POST
      public void doPost(HttpServletRequest request.
            HttpServletResponse response) throws
      ServletException.IOException{ // …
      }

      // Método doGet para responder a una petición GET
      public void doGet(HttpServletRequest request.
            HttpServletResponse response) throws ServletException.
      IOException{ // …
      }
      public void destroy(){// Liberar recursos
      }
}     //    public class MiServlet
```

Ciclo de vida de un servlet

- El cliente Web → una petición → URL
- Vía Internet, petición → servidor J2EE
- servidor J2EE, analiza, → contenedor Web.
- El contenedor Web ejecuta el servlet.
- Servlet produce HTML → servidor J2EE
- Vía Internet, servidor J2EE, HTML → cliente Web.
- cliente Web, página Web → muestra al usuario.

¿Cómo se ejecuta el servlet?

- debe implementar la interfaz Servlet (métodos: init, service y destroy)
  - **init:** es invocado por el contenedor Web
    - iniciar la ejecución del servlet.
    - (equivale a un constructor)
  - **service** se llama cada petición.
    - Este método puede recibir varias llamadas simultáneas.
    - Por cada una de ellas, crea un nuevo hilo
    - examina el tipo de petición HTTP (GET, POST, DELETE, etc)
    - llama al método adecuado para atenderla.
  - Service tiene dos objetos parámetro
    - **HttpServletRequest:** encapsula datos cliente → servidor.
    - **HttpServletResponse:** encapsula datos servidor → cliente (dos formas)
      - **PrintWriter**, método getWriter, Strings
      - **ServletOutputStream**, método getOutputStream, datos binarios.
  - **Destroy:** liberar los recursos adquiridos;

Un servlet sencillo

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
public class HolaMundo extends HttpServlet{
  // No está codificado init(); heredamos de la super clase

  protected void doGet(HttpServletRequest request,
    HttpServletResponse response)throws ServletException, IOException{
        procesarPeticion(request, response,1);
  }

  protected void doPost(HttpServletRequest request,
     HttpServletResponse response)throws ServletException, IOException{
      procesarPeticion(request, response,2);
  }

  // Manipular las posibles peticiones enviadas por el cliente:
  // utilizando el atributo method=get o method=post.
  protected void procesarPeticion(HttpServletRequest request,
    HttpServletResponse response, int tipo)throws ServletException,
                                                    IOException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head>");
    out.println("<title>Servlet HolaMundo</title>");
    out.println("</head>");
    out.println("<body>");
    out.println("<font size=7>");
    out.println("¡¡¡Hola mundo cruel!!!");
    out.println("<font size=5>");
    out.println("<br>");
    if(tipo == 1)out.println(" via metodo Get(Default)");
    else         out.println(" via metodo Post");
    out.println("</font>");
    out.println("</body>");
    out.println("</html>");
    out.close();
  }

  // Devuelve una descripción breve.
  public String getServletInfo(){
    return "Servlet HolaMundo";
  }
}
```

en el explorador tipeamos: **http://localhost:8080/servlet/HolaMundo**

Y la respuesta ES:

Software necesario para ejecutar un servlet

Como se instala Tomcat, ver Apéndice A, al final de esta Unidad
Como se ejecuta un servlet, ver Apéndice B, idem.

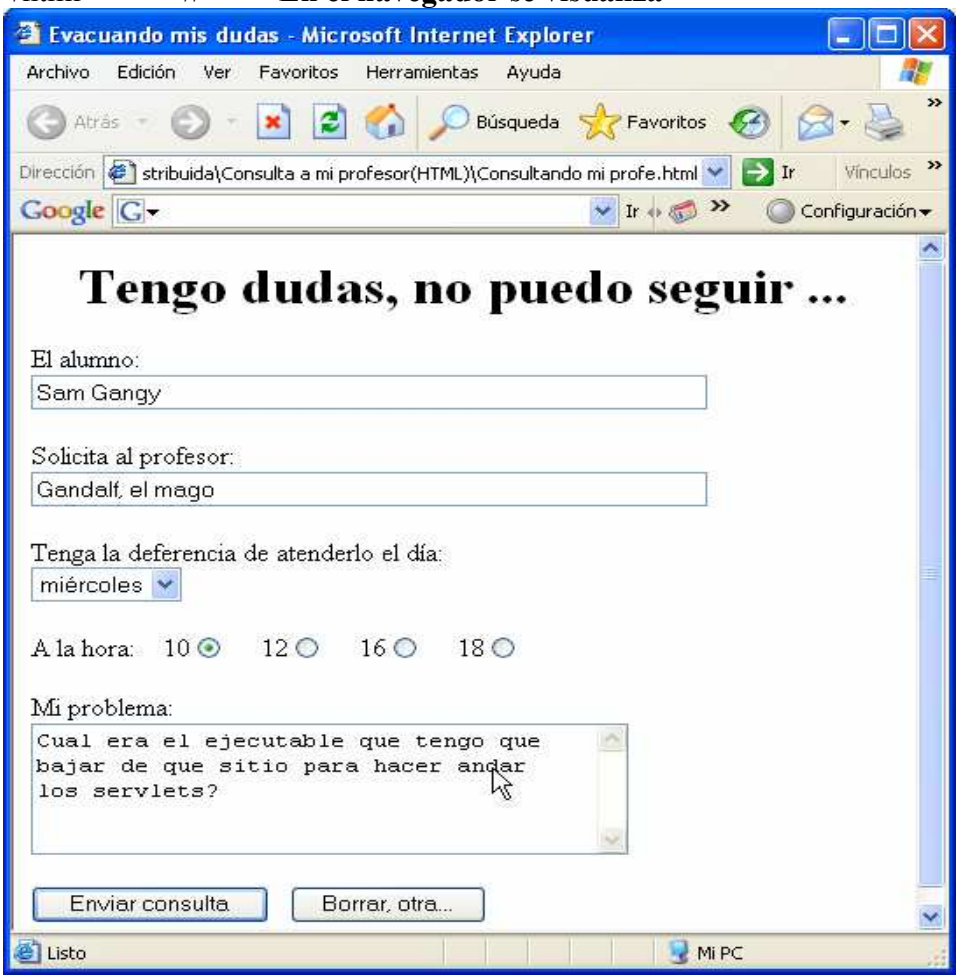
Una variante del servlet anterior

```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
import java.util.*;
public class FechaHora extends HttpServlet{
  protected void doGet(HttpServletRequest request,
    HttpServletResponse response)throws ServletException, IOException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>"); out.println("<head>");
    out.println("<title>Servlet Saludo</title>"); out.println("</head>");
    out.println("<body>"); out.println("<center>");
    Random rnd = new Random(); int tamaño;
    for (int i = 1; i <= 7; ++i){
      tamaño = rnd.nextInt(7)+1;
      out.println("<font size="+tamaño+" color=blue>");
      out.println("¡¡¡Hola mundo!!!"+tamaño);
      out.println("<br>");
    }
    out.println("<br>");
    Calendar fechahora = new GregorianCalendar();
    tamaño = rnd.nextInt(7)+1;
    out.println("<font size="+tamaño+">");
    out.println("<font color=green>");
    out.println("Son las  " + fechahora.get(Calendar.HOUR_OF_DAY) + ":" +
                fechahora.get(Calendar.MINUTE) + ":" +
                fechahora.get(Calendar.SECOND) );
    out.println("<br>");
    out.println("<font color=gray>");
    out.println("Del día " + fechahora.get(Calendar.DAY_OF_MONTH) + "/" +
                (fechahora.get(Calendar.MONTH)+1) + "/" +
                fechahora.get(Calendar.YEAR) );
    out.println("</font>");
    out.println("</center>");
    out.println("</body>");
    out.println("</html>");
    out.close();
  }
}
```

Sea un formulario Consultando mi Profe.HTML

```
<html>
  <head> <title>Evacuando mis dudas</title> </head>
  <body>
    <h1 align="center">Tengo dudas, no puedo seguir ...</h1>
    <form action="http:\\localhost:8080/servlet/ConsultaProfesor"
          method=post>
      El alumno:<br><input type="text" name="alumno" size="60">
      <br><br>Solicita al profesor:
      <br><input type="text" name="profesor" size="60">
      <br><br>Tenga la deferencia de atenderlo el día:<br>
      <select name="día">
        <option>lunes <option>miércoles <option>jueves
      </select>
      <br><br>A la hora:   
      10<input type="radio" name="hora" value="10" checked>
         
      12<input type="radio" name="hora" value="12">
         
      16<input type="radio" name="hora" value="16">
         
      18<input type="radio" name="hora" value="18">
      <br><br>Mi problema:<br><textarea name="miDuda"
                          rows="5" cols="40" wrap></textarea>
      <br><br><input type="submit" value="Enviar consulta">
        <input type="reset" value="Borrar, otra...">
    </form>
  </body>
</html>
```

// **En el navegador se visualiza**



Tipos de peticiones (Especificada en method)

- service de HttpServlet ➔ servidor recibe una petición.
- service examina el tipo de petición HTTP (GET, POST, PUT, DELETE, TRACE, OPTIONS o HEAD)
- sólo utilizamos los métodos **doGet y doPost.**

Petición HTTP GET

- (en el .HTML anterior suponemos **method=get)**
- <Consultado mi profe.HTML> ➔ navegador Web
- llenamos datos, <Enviar consulta>
- **URL + datos** (cabecera) ➔ servidor, ejemplo:

```
http://localhost:8080/servlet/ConsultaProfesor
   ?alumno=Sam+Gangy&
   profesor=Gandalf+el+mago&
   d%EDa=Miercoles&
   hora=10&
   miDuda=Cual+era+el+ejecutable+que+ …
```

Formato de la hilera enviada

- a continuación del URL los parámetros:
- alumno, profesor, día, hora y miDuda van
- después del símbolo ? y separados por el símbolo &.
- parámetros = valores, están compuestos por:
  - caracteres ASCII alfanuméricos más los caracteres
  - . (punto), - (guión), * (asterisco) y _ (subrayado);
  - los espacios en blanco son sustituidos por +
  - demás %dd (dd: código ASCII del carácter).

Petición HTTP POST

- los datos ➔ servidor en el **cuerpo** de la petición.
- método usual de enviar los datos de un formulario

Cual método usar?

- Si los datos son pocos y no confidenciales, GET.
- Si son largos, privados o importantes, POST.

Leer los datos enviados por el cliente

- **Un solo valor**, método **getParameter** del objeto **HttpServletRequest** (métodos doGet y doPost lo reciben como parámetro)
  - devuelve String **valor** parámetro argumento.

```
String valor = request.getParameter("profesor");
```

  - **getParameter** devuelve:
    - Parámetro no enviado ➔ null;
    - Parámetro sin valor ➔ cadena vacía

- **conjunto de valores**, método **getParameterValues**
  - devuelve un array String todos valores parámetro argumento

```
String[] valor = request.getParameterValues("transporte");
("transporte" = {"automovil", "tren", "avion", "bicicleta"}
(implementada con casillas de verificación en el formulario)
```

- nombres de **todos los parámetros,** → método **getParameterNames**
  - retorna una enumeración de Strings con los nombres.

```
String param, valor;
Enumeration nombresParams = request.getParameterNames();
while (nombresParams.hasMoreElements()){
      param = (String)nombresParams.nextElement();
      valor = request.getParameter(param);
      out.println(param + ": " + valor);
}
```

ejemplificar → servlet, ConsultaProfesor.java

- almacenar en un fichero de texto las consultas de los alumnos
- usaremos petición modalidad **post**
- para crear un fichero de texto:

```
FileWriter fw = new FileWriter("consultas.txt", true);
PrintWriter fichConsultas = new PrintWriter(fw);
```

- escribir en "**consultas.txt**" parejas "param" = "valor":

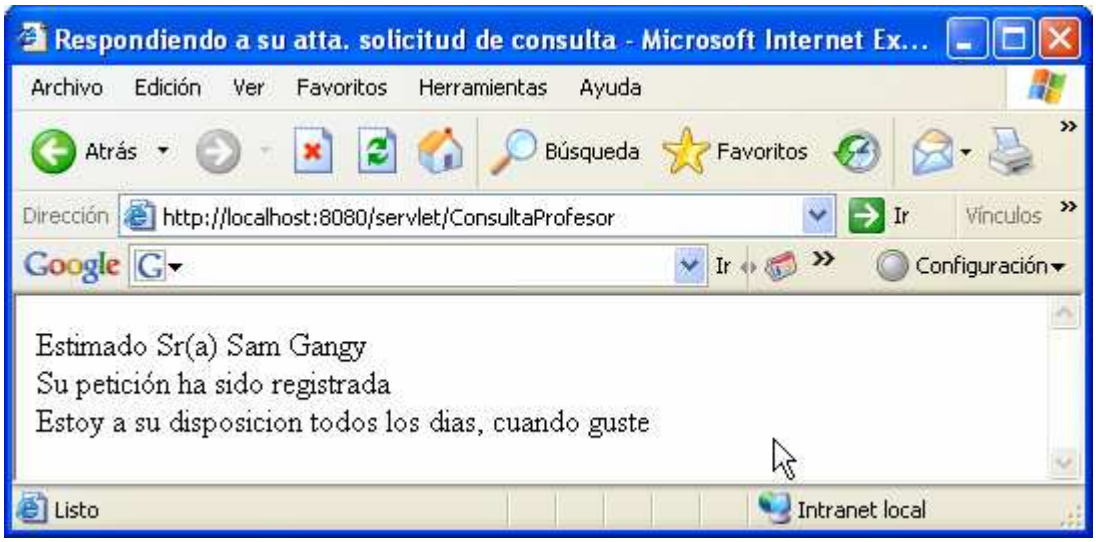
```
fichConsultas.println(param + ": " + valor);
```

Va servlet ConsultaProfesor

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

public class ConsultaProfesor extends HttpServlet{
    // variedad a los mensajes con que el servlet responde
    String[] msgs = {"Gustosamente lo atenderemos el dia del parcial",
             "Al volver de mis vacaciones tendremos  el placer ",
             "Estoy a su disposicion todos los dias, cuando guste",
             "Le recomiendo la lectura del material de la catedra",
             "Esas dudas suyas se deben a falta de base. Repase ...",
             "Su pregunta me sorprende. En realidad, no deberia ...",
             "Sus dudas corresponden a mi clase de mañana. No falte"};

    String mensaje; int indice;

    protected void doPost(HttpServletRequest request,
            HttpServletResponse response)
            throws ServletException, IOException{

        response.setContentType("text/html");//Tipo respuesta → cliente
        PrintWriter out = response.getWriter();// objeto para dev. Resp.

        try{    // Registrar la consulta del alumno
            // Abrir el fichero para el registro de la consulta
            FileWriter fw = new FileWriter("Consultas.txt", true);
            PrintWriter fichConsultas = new PrintWriter(fw);

            // Tomar los datos recibidos del cliente y escribirlos
            Enumeration nombresParams = request.getParameterNames();
```

```
            while (nombresParams.hasMoreElements()){
                 String param = (String)nombresParams.nextElement();
                 String valor = request.getParameter(param);
                 fichConsultas.println(param + ": " + valor);
             }
             fichConsultas.println("<FIN>");
             // Cerrar el fichero fichConsultas
             fichConsultas.close();
             fw.close();

             // Respondiendo al alumno
             out.println("<html>");
             out.println("<title>Respondiendo a su atta. solicitud de
                          consulta</title>");
             String nome = request.getParameter("alumno");
             out.println("Estimado Sr(a) "+ nome);
             out.println("<br>");
             out.println("Su petición ha sido registrada");
             out.println("<br>");
             indice = (int)(7*Math.random());
             mensaje = msgs[indice]; // Una respuesta al azar
             out.println(mensaje);
             out.println("<br>");
             out.println("</html>");
         }
         catch(IOException e){
             out.println("Hubo problemas cursando su solicitud.");
             out.println("<br>Por favor, inténtelo otra vez.");
         }
         out.close();
     }
     // Cerrar el flujo
     // Devuelve una descripción breve.
     public String getServletInfo(){return "Servlet Tutorías";}
}
```

Ante la consulta en formulario **Consultando mi Profe.HTML**, filmina pg 44, la respuesta del servlet **ConsultaProfesor**



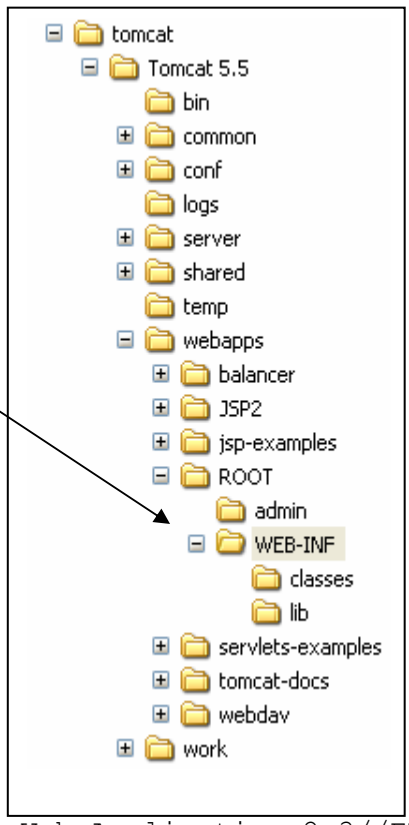
(No siempre es tan amable …)

Descriptor de despliegue

- fichero de texto basado en XML:
  - Como utilizar los componentes.
  - Parámetros de iniciación
  - opciones de seguridad.

- Para servlets, JSP, etc.),
  - Recibe el nombre de **web.xml**
  - Está en la carpeta WEB-INF.

- Para ConsultaProfesor contiene:
  - el nombre y la descripción
  - la clase del servlet
  - los parámetros de iniciación
  - alguna otra información




```
<?xml version="1.O" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
      PUBLIC "-//Sun Mierosystems, Ine.//DTD Web Applieation 2.3//EN"
      "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
      <display-name>Servlet ConsultaProfesor</display-name>
      <servlet>
            <servlet-name>ConsultaProfesor</servlet-name>
            <servlet-elass>ConsultaProfesor</servlet-elass>
            <init-param>
                  <param-name>carpeta</param-name>
                  <param-value>D:/Servlets/Ejemplos PPR</param-value>
            </init-param>
      </servlet>
      <servlet-mapping>
            <servlet-name>ConsultaProfesor</servlet-name>
            <url-pattern>/servlet/ConsultaProfesor</url-pattern>
      </servlet-mapping>
      <welcome-file-list>
            <welcome-file>ConsultaProfesor.html</welcome-file>
      </welcome-file-list>
</web-app>
```

- <**display-name**>: lo que mostrará administrador aplicaciones Web
- <**servlet**> definen los datos relativos al servlet.
- <**init-param**> definen parámetros de iniciación y sus valores.
- <**url-pattern**> resto path luego del especificado en TOMCAT_HOME
- <**welcome-file**> fichero HTML/JSP utilizado para arrancar servicio.

http://localhost:8080/ConsultaProfesor

Nota: Donde será que se grabó **consultas.txt?**
Investigamos: D:\tomcat\Tomcat5.5.
Valor de la variable de entorno **TOMCAT_HOME**.

Iniciación de un servlet

contenedor Web carga/crea un objeto de la clase del servlet

- se ejecuta su método **init.**
- iniciación personalizada?: redefinir el heredado de **HttpServlet.**
- Por ejemplo, el fichero Consultas.txt, se grabó en <**TOMCAT_HOME>**
- en web.xml habiamos especificado

```
<init-param>
     <param-name>carpeta</param-name>
     <param-value>D:/Servlets/Ejemplos PPR</param-value>
</init-param>
```

- pero en public class **ConsultaProfesor** no dijimos nada …

```
try{    // Registrar la consulta del alumno
    // Abrir el fichero para el registro de la consulta
    FileWriter fw = new FileWriter("Consultas.txt", true);
    PrintWriter fichConsultas = new PrintWriter(fw);
```

- **Que debriamos haber hecho?**

```
public class ConsultaProfesor extends HttpServlet{
    String carpeta = null;
    public void init(ServletConfig config) throws ServletException{
        super.init(config);
        carpeta = getInitParameter("carpeta");
        // parámetro establecido en el fichero web.xml que se
        // localiza en la carpeta WEB-INF del servidor Web Tomcat
        if (carpeta == null)
            System.err.println("Destino no especificado");
    }
    // codificación anterior
    // Abrir el fichero para el registro de la consulta
    FileWriter fw = new FileWriter(carpeta*"Consultas.txt", true);
    …
}
```

Seguimiento de una sesión

- HTTP es un protocolo sin estado:
- cada petición es tratada de forma independiente por el servidor.
- el servidor **no sabe** si n peticiones están relacionadas.
  Existen varias soluciones a este problema:
  - cookies,
  - reescritura del URL
  - campos ocultos en los formularios.
  - Usando objeto HttpSession (HttoServletRequest.getSession())

Cookies

- pequeña cantidad de información (no más de 4K)
- almacenada en la máquina cliente por el servlet(dueño)
- consultada a través de la API de cookies de los servlets.
- Solucion muy usada para seguimiento de una sesión.
- contenedor Web ➔ al cliente cookie con un id Sesión
- id Sesión, cliente ➔ servidor cada nueva petición
- Java proporciona la clase **Cookie,** paquete javax.servlet.http

- Una cookie se compone: **un nombre y un valor;**
  - Nombre: la identifica entre todas las demás del mismo cliente
  - valor: es un dato asociado con la cookie.

- Un servlet crea una cookie:

```
Cookie miCookie = new Cookie("nombre-cookie", "valor-asociado");
```

- Y la agrega a la cabecera de la respuesta HTTP.

```
response.addCookie(miCookie);
```

- Para recuperar cookie(s):

```
Cookie cookies[] = request.getCookies();
if (cookies == null) out.println("No hay cookies");
```

- **petición a un servidor**
  - cliente ➔ servidor cookies propias unicamente.
  - *servlets* ejecutan en un servidor comparten las *cookies*.

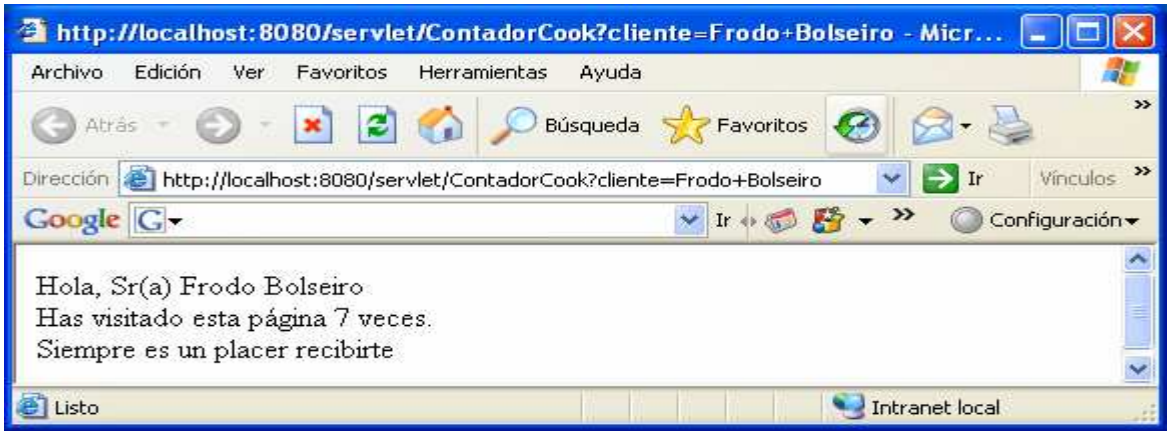
- El servlet puede recorrer el array de cookies

```
String nombre, valor;

for (int i = 0; i < cookies.length; ++i)
   nombre = cookies[i].getName();
   valor = cookies[i].getValue();
   // Operaciones …
 }
```

ejemplo: veces que este servicio es accedido por cada cliente

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

public class ContadorCook extends HttpServlet{
  protected void doGet(HttpServletRequest request,
           HttpServletResponse response)throws ServletException,
                                                IOException{
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    // Obtener el valor actual de la cookie "contador.cook"
    String sCuenta  = null;  String clieNome  = null;
           clieNome  = request.getParameter("cliente");
    Cookie[] cookies = request.getCookies();
```

```
    if (cookies != null){ // hay cookies...
        for (int i = 0; i < cookies.length; i++){
            if (cookies[i].getName().equals("contador.cook")){
                sCuenta = cookies[i].getValue(); // valor asociado
               break;
            }
        }   //  for
    }   //  if (cookies != null)

    Integer objCuenta = null; // contador
    if (sCuenta == null) // no existía tal cookie todavía
       objCuenta = new Integer(1);
    else                 // ya existía "cuenta.cook"
        objCuenta = new Integer(Integer.parseInt(sCuenta)+1);

    Cookie c = new Cookie("contador.cook", objCuenta.toString());
    response.addCookie(c); // Añadir la cookie a la respuesta HTTP

    out.println("<html>");// Responder al cliente
      out.println("Hola, Sr(a) "+clieNome);
      out.println("<br>");
      out.println("Has visitado esta página " + objCuenta.toString() +
                ((objCuenta.intValue() == 1) ? " vez." : " veces."));
      out.println("<br>");
      out.println("Siempre es un placer recibirte");
    out.println("</html>");
    out.close();// Cerrar el flujo
  }  //   protected void doGet
  public String getServletInfo(){return "Servlet ContadorCook";}
}
```



Combinando documentos HTML, Servers, Cookies …

- Un documento aComprar.HTML, datos del cliente ➔ servlet Atención.
- Atención ➔ cliente: combo de tres articulos; cliente puede:
  - Volver atraz, (Para optar por otro combo)
  - Ir al servlet Despacho (Para pedir el envio)

- El servlet Despacho solo sirve para:
  - Volver atraz, (Para optar por otro combo)
  - Ir al servlet Envio (No implementado, Tomcat nos lo dirá)

aComprar.Html

Source Editor [aComprar]

```
1  <html>
2    <head>
3      <title>Atendiendo clientes queridos...</title>
4    </head>
5    <body>
6      <hl align="center">Tengo hambre!!! ... no puedo m...</hl>
7      <form action="http:\\localhost:8080/servlet/Atencion">
8        El hambriento: <input type="text" name="hambriento" size="20">  
9          Solicita al Hiper: <input type="text" name="hiper" size="20">
10       <br><br><font size=5>Tenga la deferencia de atenderlo :
11       <font size=6>¡¡¡Urgente!!!</font>
12       <br><br><input type="submit" value="Atender ya!">
13         <input type="reset" value="Desistir...">
14     </form>
15   </body>
16 </html>
17
```

14:12 INS

Produce la página

Clickeando **Atender ya**, el servlet invocado es **Atención**, su codigo:

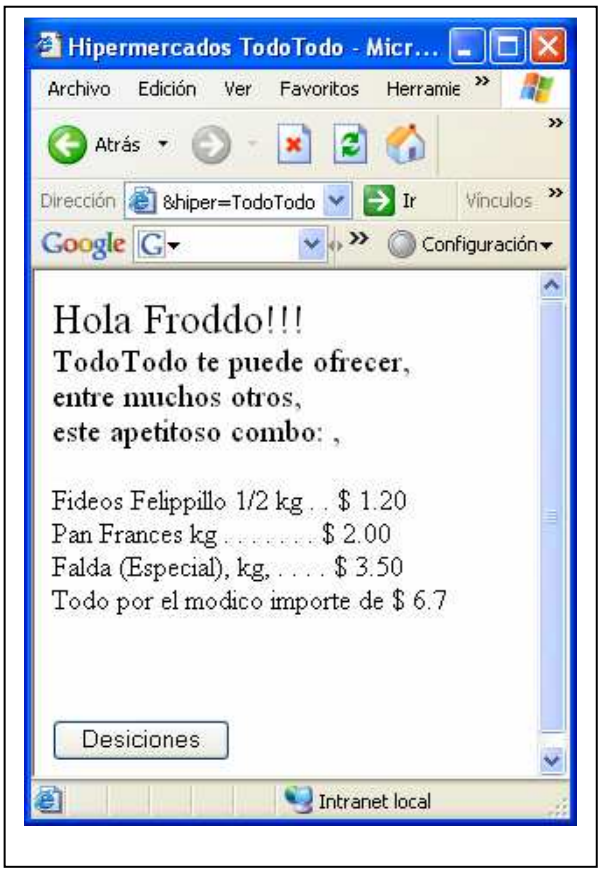
```
import java.io.*; import javax.servlet.*; import javax.servlet.http.*;
import java.util.*; import java.net.URLEncoder;

public class Atencion extends HttpServlet{
    private String hungry, hipMerc;

    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
                                             IOException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // obtenemos datos del formulario
        hungry = request.getParameter("hambriento");
        hipMerc= request.getParameter("hiper");

        // Preparamos informacion acerca de comidas
        Comidas comida = new Comidas();

        // Necesitamos idSesion,  buscamos por nombre cliente
        // (getName())entre las cookies recibidas.
        String idSesion = null;
        Cookie[] cookies = request.getCookies();
        if (cookies != null){
            for (int i = 0; i < cookies.length; i++){
                if (cookies[i].getName().equals(hungry)){
                    idSesion = cookies[i].getValue();
                    break;
                }
            }   //  for
        }   //  if (cookies ...

        // idSesion no existe?: generarlo.
        if (idSesion == null){ // Es la primera conexion del cliente
            idSesion = generarIdSesion(); // le generamos su idSesion
            Cookie c = new Cookie(hungry, idSesion); // su Cookie
            response.addCookie(c);  // incorporamos a response
        }

        //  Generamos html para el cliente
        out.println("<head>");
        out.println("<title>Hipermercados " + hipMerc + "</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<font size="+5+">");
        out.println("Hola " + hungry + "!!!<br>");
        out.println("<font size=" + 4 + ">");
        out.println(hipMerc + " te puede ofrecer,<br>");
        out.println("entre muchos otros,<br>");
        out.println("este apetitoso combo: ,<br><br>");

        int i, j; float total = 0; Float precioIt;
        String auxStr;
        out.println("<font size=" + 3 + ">");
```

```
          for (i = 0; i < 3; i++){    // El combo
             j = (int)(10*Math.random());
             out.println(comida.carta[j]+"<br>");
             auxStr = new String(comida.carta[j]);
             auxStr = auxStr.substring(31);
             auxStr = auxStr.trim();
             precioIt = new Float(auxStr);
             total+= precioIt.floatValue();
          }
          out.println("Todo por el modico importe de $ "+total);
          out.println("</font>");

          //  Vamos al servlet Despacho, donde le
          //  preguntaremos si continua o finaliza
          out.println("<form action=/servlet/Despacho?&idSesion="+idSesion
                      + "tuPedido="+tuPedido
                      + " method=get>");
          out.println("<br><br>");
          out.println("<input type=submit value=\"Decisiones\">");
          out.println("</form>");
          out.println("</body>");
          out.println("</html>");
      }

      private static String generarIdSesion(){
          // Garantizar un id único
          String uid = new java.rmi.server.UID().toString();
          // Codificar cualquier carácter especial
          return java.net.URLEncoder.encode(uid);
      }
}     //  public class Atención
```

El servlet Atención usa **class Comidas**, hela aquí:

```
class Comidas{
    String[] carta =
    {"Amburguesas, cja 10 Un, . . . $10.00",
     "Fideos Macarron, pk 1 kg, . . $ 1.99",
     "Cerveza BadHein, env 780cc  . $ 2.80",
     "Falda (Especial), kg, . . . . $ 3.50",
     "Bife lomo tern, kg, . . . . . $12.50",
     "Pan Frances kg . . . . . .  $ 2.00",
     "Polenta PrestPres 1/2 kg .  . $ 1.60",
     "Fideos Felippillo 1/2 kg . .  $ 1.20",
     "Tinto FacuRelli, 1 lt . . . . $ 2.80",
     "Salch. VienAus, pk 200 grs, . $ 2.70"};
     // 23456789 123456789 123456789 12345
     String pedido ="";

     public String[] getComidas(){return carta;}
     public String   getComida(int cual){return carta[cual];}
     public void addItem(String item){pedido+=item+", <br>";}
     public String getPedido(){return pedido;}
}    //  class Comidas
```

Y la página que se genera para el cliente,
titulada **Hipermercados TodoTodo**



<decisiones> llama al servlet **Despacho**,
quien presenta la siguiente página _____

```
import java.util.*; import java.net.URLEncoder;
public class Despacho extends HttpServlet{
    private int ind; private String idSesion;
    private String itemPed;
    public void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
                                             IOException{
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        // Necesito datos del cliente
        idSesion = request.getParameter("idSesion");

        //  Generando html para el cliente
        out.println("<head>");
        out.println("<title>Despacho</title>");
        out.println("</head>");
        out.println("<body>");

        out.println("<font size="+5+">");
        out.println("Despacho<br>");

        out.println("<font size="+3+">");
        out.println("Tenemos muchos otros combos<br>");
        out.println("Para verlos, boton <-- Atraz<br>");
```

```
        out.println("<form action=\"/servlet/Envio\"" +
                    " method=get>");
        out.println("<br><br>");
        out.println("<input type=submit value=\"Enviar combo\">");
        out.println("</form>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

<Enviar Combo> ➔ servlet **Envio**, no implementado. **Tomcat nos dice:**

Java Server Pages (JSP)

- tecnología ➔ crear **páginas Web** con **programación en Java.**
- compuestas de código **HTML/XML**
- + etiquetas especiales (órdenes)
- + código Java (scriptlets).

El éxito de esta tecnología ➔ JSP puede:

- Procesar petición: presentacion + lógica de negocio:
- Separar presentación de lógica de negocio:
  - Utilizando tecnología servlets,
  - EJB (enterprice java Beans),
  - Otras: *Apache* ***Struts, Spring Frameworks, JBoss Seam,*** *etc,*

El motor JSP, está basado en servlets.

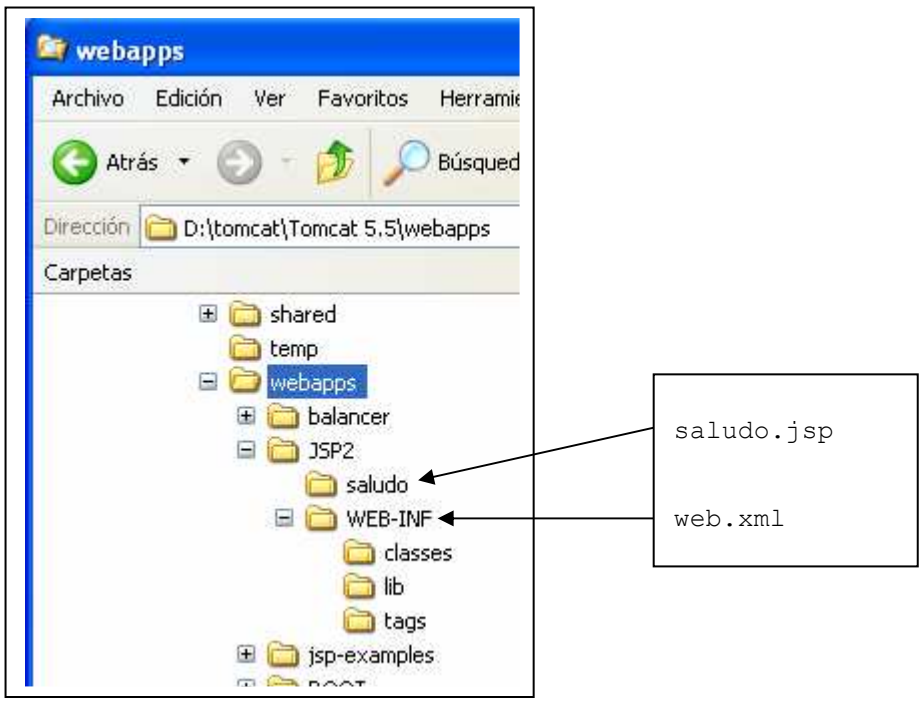
¿Cómo trabaja una JSP?

- Se crean parecido a idem **ASP** (Active Server Page de Microsoft)
- **PHP** (Personal Home Page)
- El código se almacena en ficheros.jsp

```
<%@page contentType="text/html"%>
<html>
<head><title>Ejemplo JSP</title></head>
      <body bgcolor="ffffcc">
            <center>
            <h2>Hola mundo</h2>
                  <%java.util.Date hoy = new java.util.Date();%>
                  La fecha de hoy es: <%=hoy%>
            </center>
      </body>
</html>
```

Como ejecutar una JSP?

- instalarla en el servidor de aplicaciones (Tomcat):
- Debajo de < TOMCAT_HOME>\webapps añada **JSP2**
- Debajo **JSP2** añada saludo y **WEB-INF**
- Debajo de **WEB-INF** añada classes, lib y tags.
- Pegue, (donde indicado), web.xml y saludo.jsp.



- Verifique contenido de web.xml:

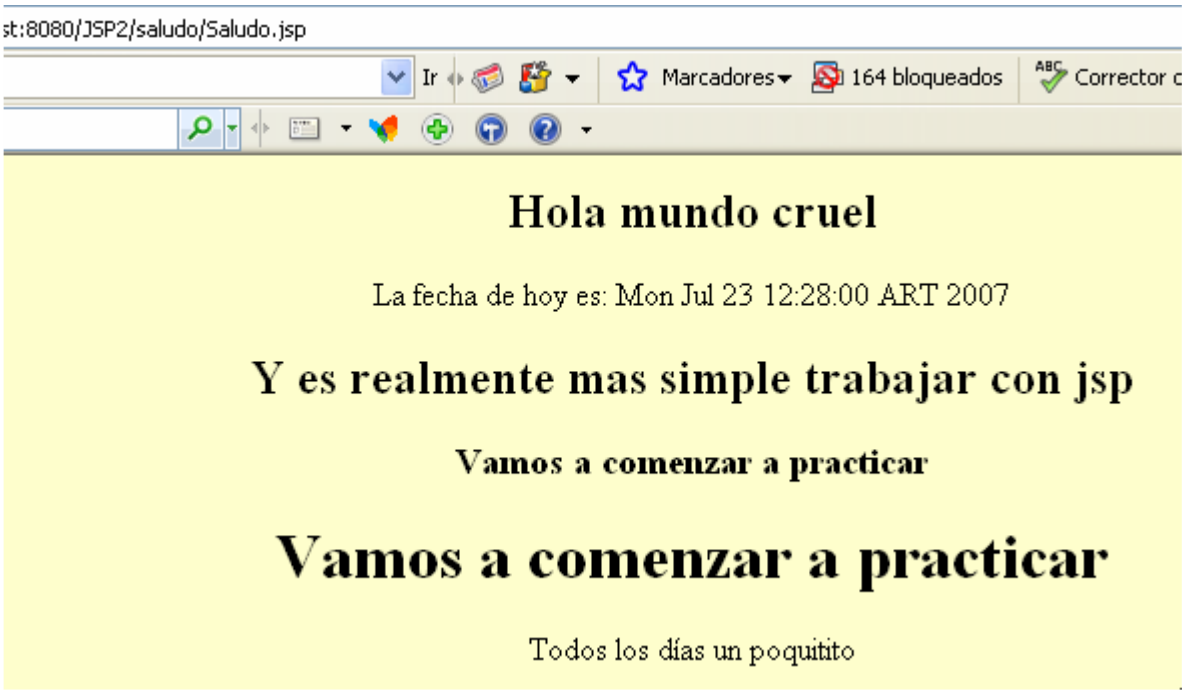
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app
      xmlns=http://java.sun.com/xml/ns/j2ee
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi :schemaLocation="http://java.sun.com/xml/ns/j2ee web-
      app_2_4.xsd"
      version="2.4">
</web-app>
```

- Arranque el Tomcat

- Ejecute la página JSP.

  http://localhost:8080/JSP2/saludo/Saludo.jsp

  - **Primera vez o página fuente modificada (motor JSP)**
    - El motor JSP traduce el fuente .jsp a fuente .java
  - **Veces siguientes (Motor Servlet)**
    - El fuente java es compilado, se genera el .class;
    - el motor de servlets actua normalmente



- Tomcat Saludo.jsp ➔ Saludo.java; almacenado en
  en Tomcat 5.5\work\Catalina\localhost\JSP2\org\apache\jsp\saludo.java
  72 líneas de java!!! Las 10 primeras:

```
package org.apache.jsp.saludo;
import javax.servlet.*; import javax.servlet.http.*;
import javax.servlet.jsp.*;
public final class Saludo_jsp
      extends org.apache.jasper.runtime.HttpJspBase
      implements org.apache.jasper.runtime.JspSourceDependent{
  private static java.util.List _jspx_dependants;
  public Object getDependants(){return _jspx_dependants;}

  public void _jspService(HttpServletRequest request,
                          HttpServletResponse response)
                throws java.io.IOException, ServletException {
```

Ciclo de vida de una página JSP

- class **Saludo_jsp ➔ HttpJspBase**, (quien implementa **Servlet**).
- existirá un método **_JspService**
  - contiene JSP ➔ HTML dinámico generado,
  - que **no se puede sobrescribir.**
- es posible **agregar** código
  - _jsp**Init** de iniciación
  - _jsp**Destroy** de finalización

Objetos implícitos - JSP utiliza una serie de objetos:

- **pageContext:** proporciona acceso a objetos relacionados con servlets
- **request:** objeto de la HttpServletRequest
- **response:** objeto de la HttpServletResponse
- a**pplication:** objeto de la clase PageContext
- hay mas: **config, session, out, page, exception.**

Ámbito de los atributos

- los objetos httpSession, servletContext o pageContext ➔ **atributos**
- **atributos** tienen **ámbito:** página, petición, sesión y aplicación
- El **ámbito** de un atributo determina su duración y visibilidad
- JSP ➔ atributos ➔ métodos de la interfaz PageContext.
  - setAttribute y getAttribute.

```
pageContext.setAttribute(nombre, valor, ámbito);
```

El ámbito de los atributos dependen del objeto implícito de la interfaz

Interfaz	**objeto implícito**	**ámbito**
ServletContext	application	aplicación
HttpSession	session	sesión
HttpServletRequest	request	petición
PageContext	page	página

Cuándo definir uno u otro ámbito ? según se use:

- sólo en una página, ➔ página;
- más de una página, ➔ petición;
- múltiples peticiones, ➔ sesión;
- distintas sesiones, ➔ aplicación.

Ejemplo

- **Cuenta.jsp** es parecido a lo que hace **ContadorCook.java** (Pg 50):
- Cuenta el numero de veces que accesamos a ess página
- **Si atrCuenta** es atributo del objeto implícito **application,** contamos todos los accesos(<> sesiones) en un único contador
- **Si atrCuenta** es atributo del objeto implícito **session,** contamos los accesos de cada cliente en la sesion por separado

```
<%-- página JSP Cuenta.jsp --%>
<%@page contentType="text/html"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head><title>Página JSP cuenta</title></head>
<body>
<!--Incrementar el contador para esta página. El valor es
    guardado en la variable "atrCuenta" definida en el ámbito
    de aplicación.
-->
  <c:set var="atrCuenta" scope="application" value="${atrCuenta + 1}"/>
    <h3>Demostración de seguimiento a nivel de aplicación</h3>
  <!-- Visualizar la cuenta para esta página -->
  ${param.parNombre}, has visitado esta página
  ${atrCuenta} ${(atrCuenta > 1) ? " veces." : " vez."}
</body>
</html>
```

Demostración de seguimiento a nivel de aplicación

, has visitado esta página 1 vez.

Demostración de seguimiento a nivel de aplicación

, has visitado esta página 2 veces.

A nivel de **aplicación,**

- varias sesiones (<> clientes) suman en un único contador
- el conteo se pierde(reinicia) cuando
  - Desistalamos aplicación (Bajando el Tomcat, por ejemplo)
  - Modificando el fuente de Cuenta.jsp

Modificamos el fuente, ahora el ambito es sesión

A nivel de **sesión,**

- varias sesiones (<> clientes) suman en contadores separados
- el conteo se pierde(reinicia) cuando
  - Desistalamos aplicación (Bajando el Tomcat, por ejemplo)
  - Modificando el fuente de Cuenta.jsp
  - Transcurrió el tiempo fijado por servidor
  - Sesión del cliente es cancelada manualmente.