

Introducción a las Tecnologías Web

Java y Servicios Web I
Master en Ingeniería Matemática

Manuel Montenegro
Dpto. Sistemas Informáticos y Computación

Desp. 467 (Mat)

montenegro@fdi.ucm.es



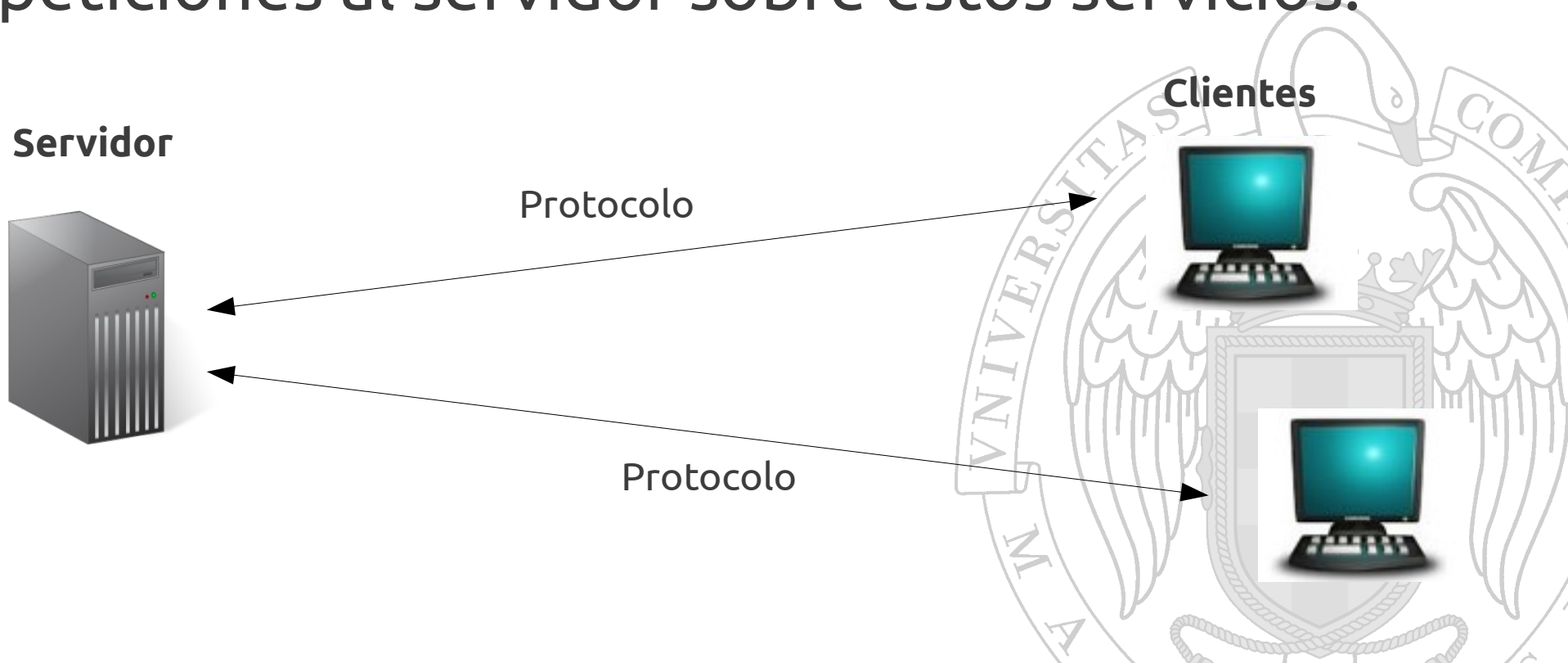
Contenidos

- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*



Clientes y servidores

- Un **servidor** es un programa que proporciona recursos o servicios.
- Un **cliente** es un programa que realiza peticiones al servidor sobre estos servicios.



Puertos

- Un servidor puede ofrecer distintos tipos de servicios. Cada uno de ellos se realiza a través de un **puerto** en la máquina servidor.
- Cada puerto está identificado con un número entre 0 y 65535.

Servidor



20

22

25

80

110

- **Puertos 1 – 1024**
Reservados para el S.O. y para protocolos conocidos.
- **Puertos 1024 – 49151**
Pueden ser usados para cualquier aplicación.
- **Puertos 49152 – 65535**
Dinámicos/privados.

Puertos

Servidor web



80

Protocolo HTTP

Cliente web



Servidor de correo



110

Protocolo POP3

25

Protocolo SMTP

143

Protocolo IMAP

Cliente de correo



Servidor MySQL



3306

SQL

Driver JDBC



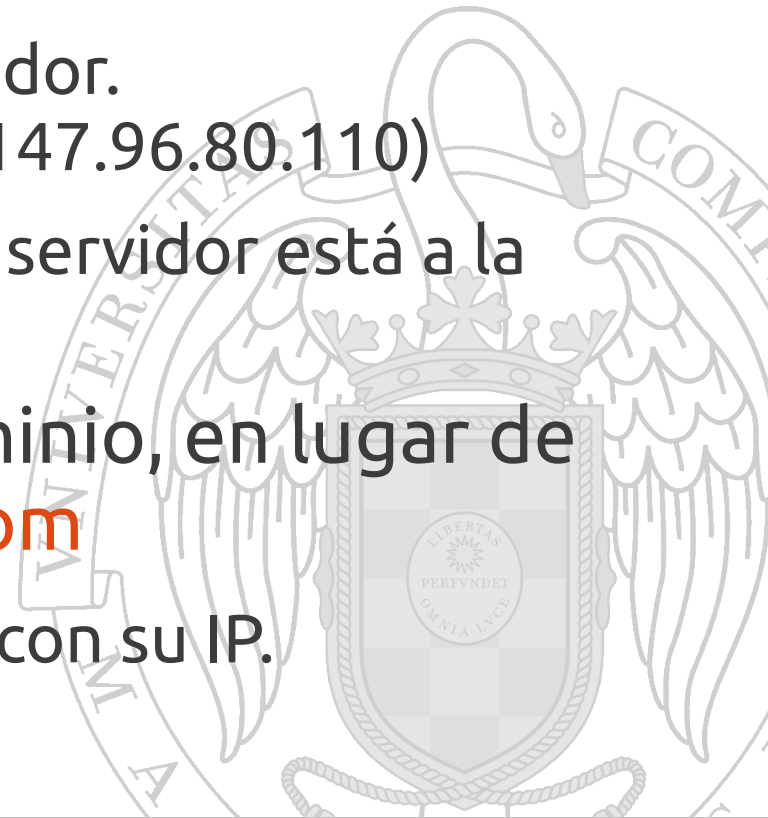
Contenidos

- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*



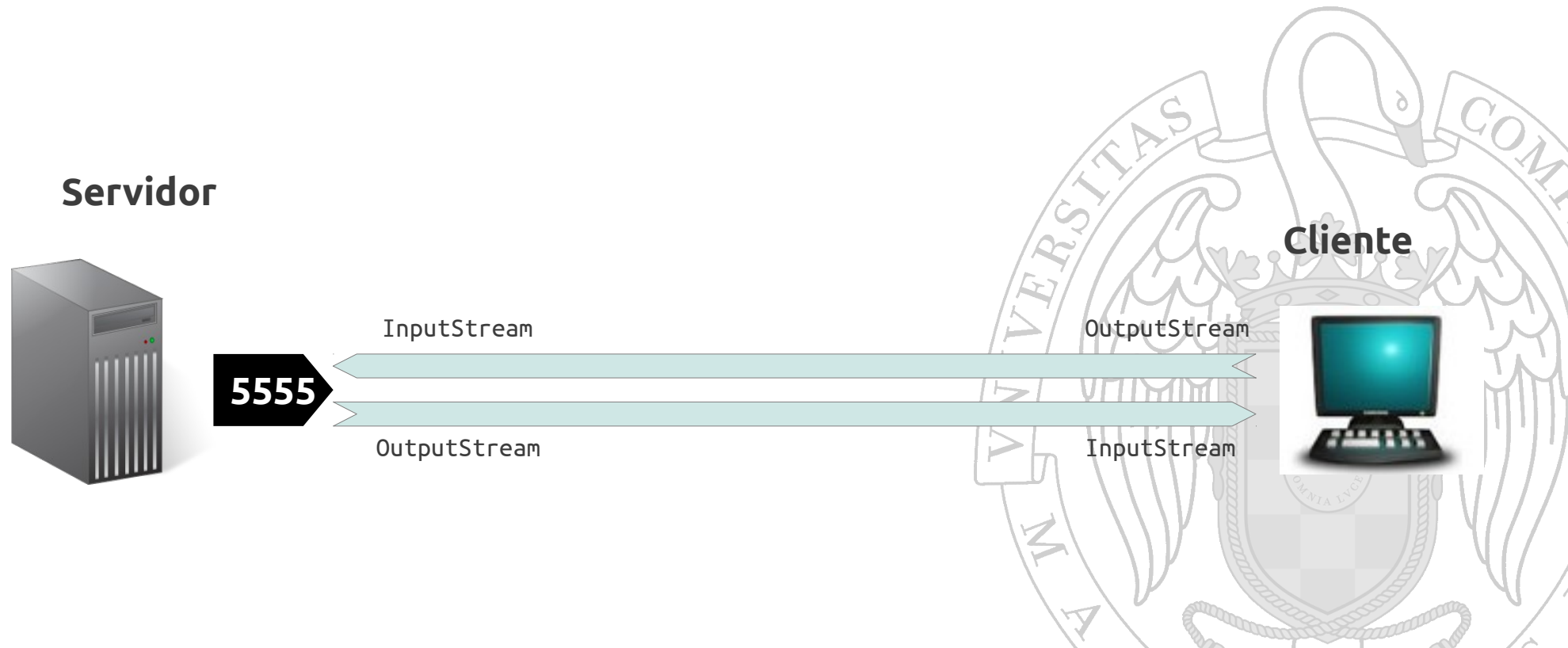
Comunicación mediante *sockets*

- Los *sockets* conforman el método más rudimentario de comunicación sobre los protocolos TCP/IP.
- Para que un cliente pueda conectarse a un servidor, necesita:
 - **Dirección IP** dónde localizar al servidor.
Cuatro números entre 0 y 255. (Ej: 147.96.80.110)
 - **Número de puerto**, en el programa servidor está a la espera.
- También sirve un *nombre* del dominio, en lugar de una dirección IP: **www.ejemplo.com**
 - **Servidor DNS**: Asocia cada nombre con su IP.



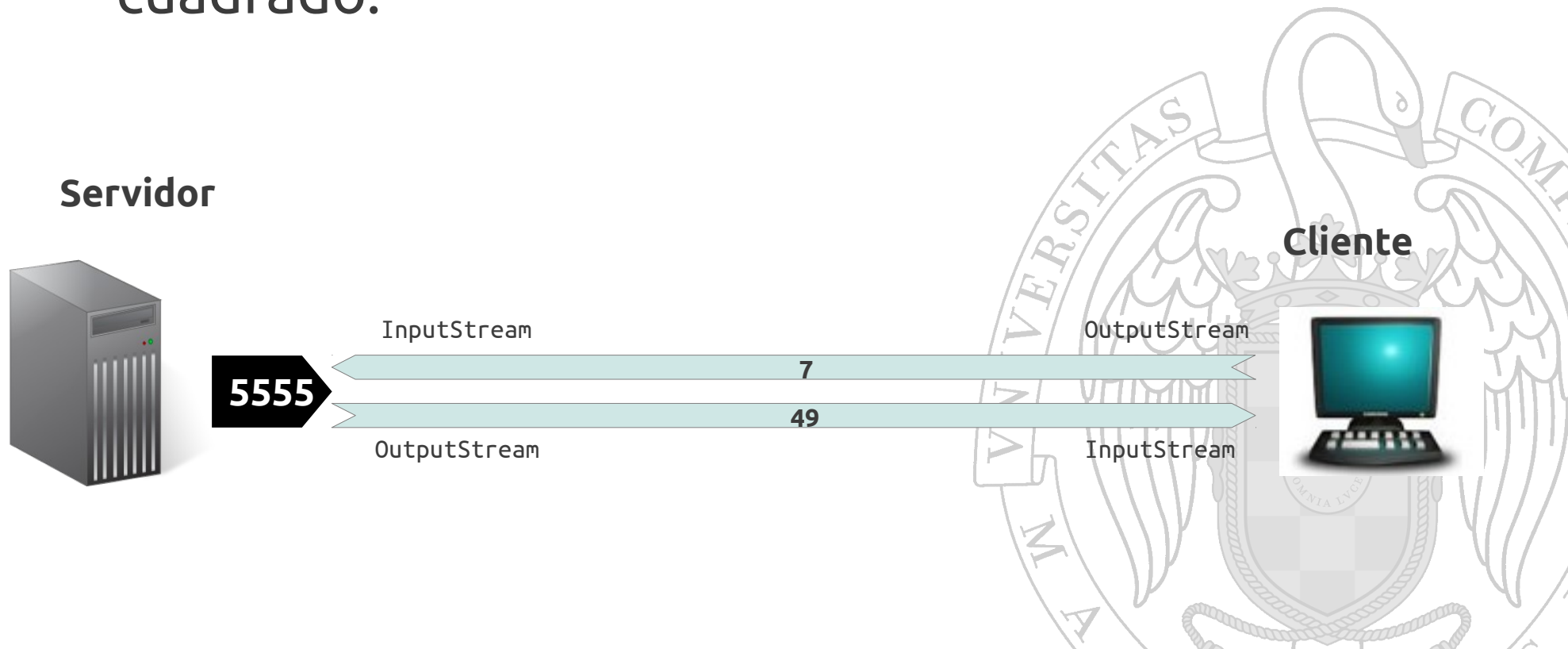
Comunicación mediante *sockets*

- Una vez establecido un **socket** entre dos ordenadores, la comunicación se realiza mediante dos **canales de datos**, de la misma manera en la que se accede un archivo en Java.



Creación de *sockets* en Java

- Caso sencillo:
 - El cliente envía un número al servidor.
 - El servidor devuelve el número recibido elevado al cuadrado.



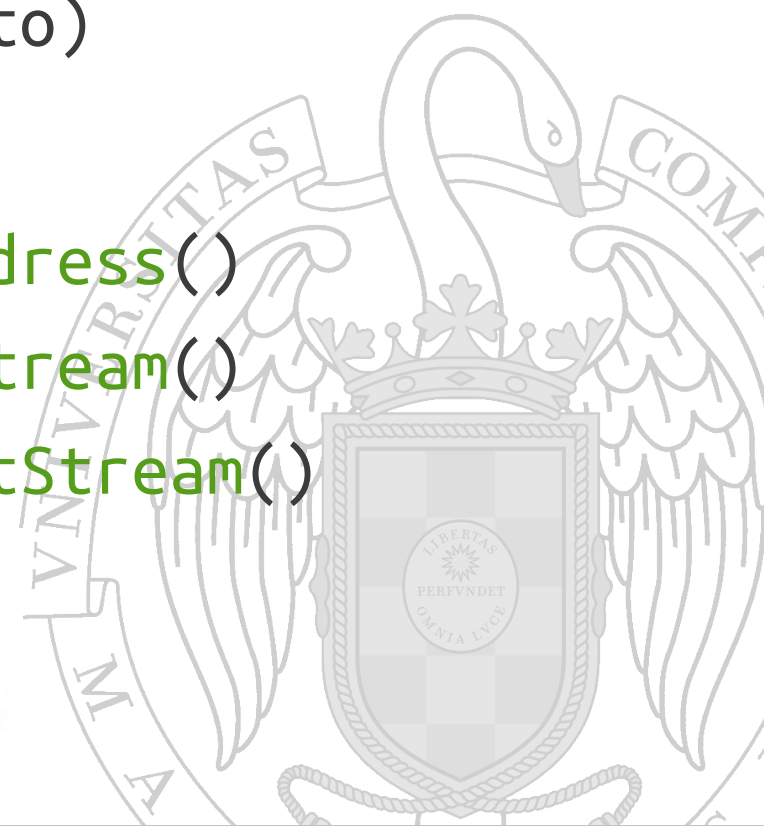
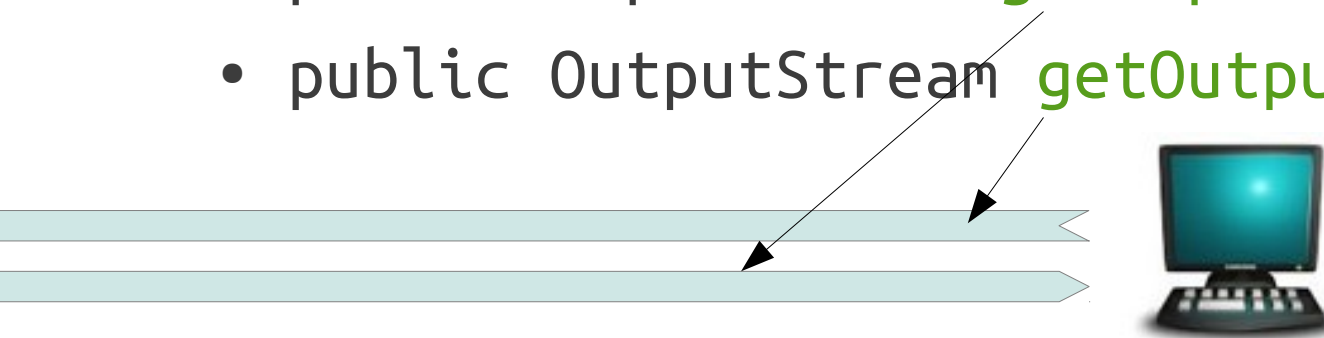
La clase ServerSocket

- Paquete `java.net`
- Un objeto `ServerSocket` espera la conexión de un cliente a través de la red.
- Constructores:
 - `ServerSocket(int puerto)`
 - `ServerSocket(int puerto, int longCola)`
- Métodos:
 - `public Socket accept()` throws `IOException`



La clase Socket

- Representa una conexión particular entre un cliente determinado y un servidor.
- Constructor:
 - `Socket(String host, int puerto)`
- Métodos:
 - `public InetAddress getAddress()`
 - `public InputStream getInputStream()`
 - `public OutputStream getOutputStream()`

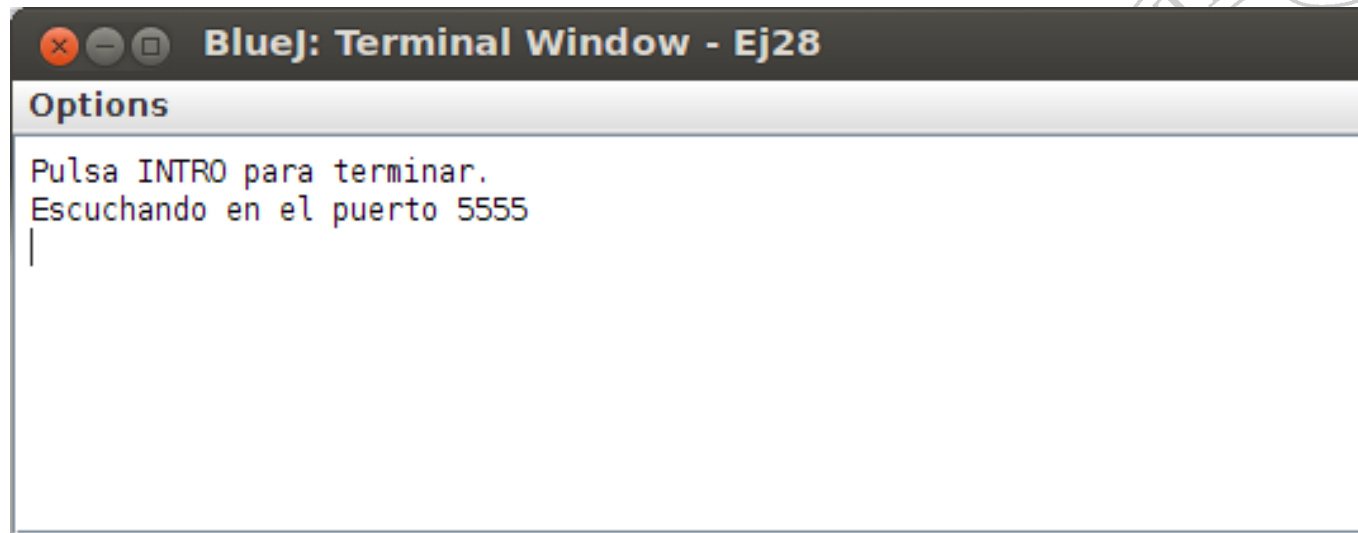


Ejemplo

```
public class ServidorCalculo implements Runnable {
    public void run() {
        ServerSocket server = null;
        Socket s = null;
        Scanner sc = null;
        PrintWriter pw = null;
        try {
            server = new ServerSocket(5555);
            System.out.println("Escuchando en el puerto 5555");
            while (true) {
                s = server.accept();
                System.out.println("Recibida conexión desde " + s.getInetAddress());
                sc = new Scanner(s.getInputStream());
                int n = sc.nextInt();
                System.out.println("He recibido un " + n);
                pw = new PrintWriter(s.getOutputStream());
                pw.print(n*n);
                pw.flush();
                pw.close();
                sc.close();
            }
        } catch (IOException e) { e.printStackTrace(); }
        finally {
            if (s != null) try { s.close(); } catch (IOException e) {}
            if (server != null) try {server.close();} catch (IOException e) {}
        }
    }
}
```

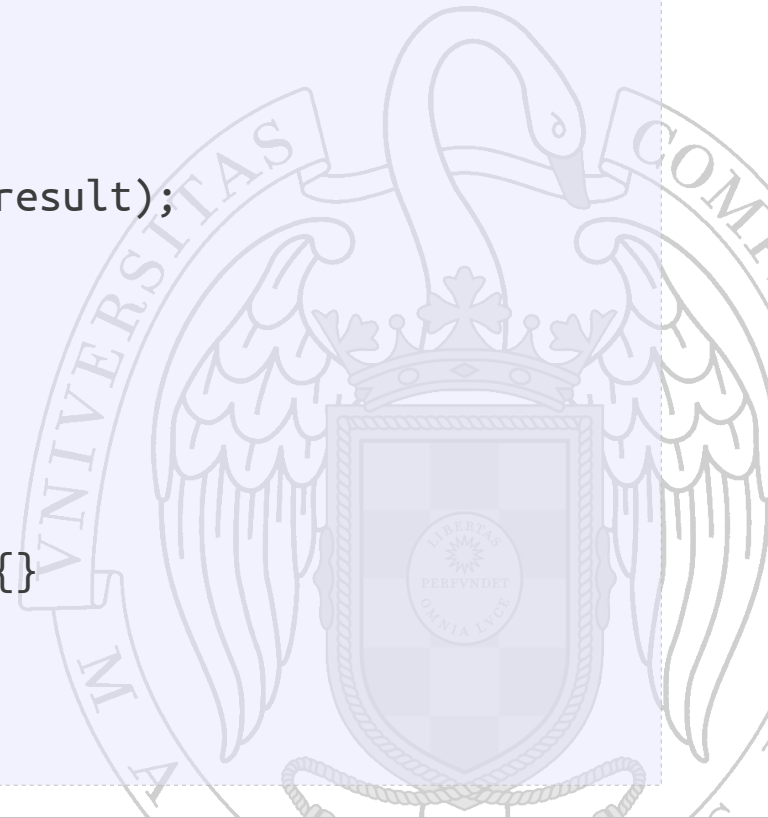
Ejemplo

```
public class ServidorCalculo implements Runnable {  
    ...  
    public static void main(String[] args) {  
        Thread t = new Thread(new ServidorCalculo());  
        t.setDaemon(true);  
        t.start();  
        System.out.println("Pulsa INTRO para terminar.");  
        Scanner sc = new Scanner(System.in);  
        sc.nextLine();  
    }  
}
```

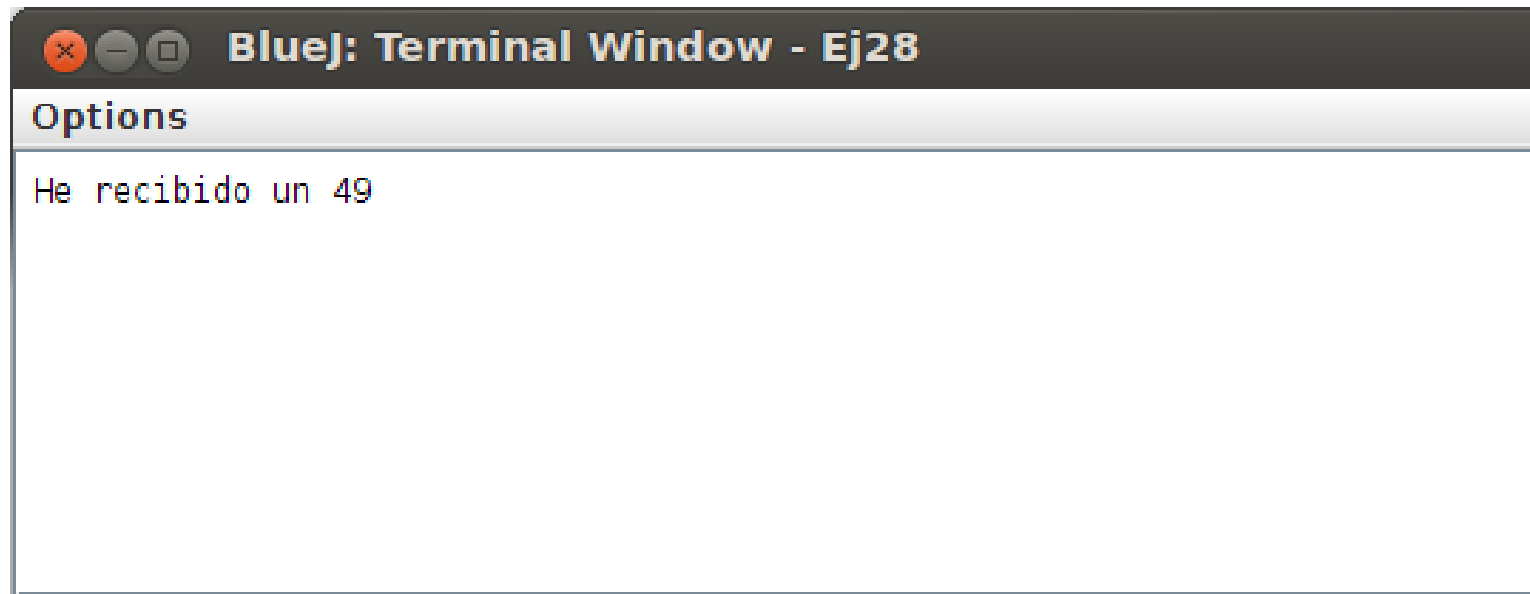


Ejemplo

```
public class ClienteCalculo {  
    public static void main(String[] args) {  
        Socket s = null;  
        PrintWriter pw = null;  
        Scanner sc = null;  
        try {  
            s = new Socket("localhost", 5555);  
            pw = new PrintWriter(s.getOutputStream());  
            pw.println(7);  
            pw.flush();  
            sc = new Scanner(s.getInputStream());  
            int result = sc.nextInt();  
            System.out.println("He recibido un " + result);  
        } catch (UnknownHostException e) {  
            e.printStackTrace();  
        } catch (IOException e) {  
            e.printStackTrace();  
        } finally {  
            sc.close();  
            pw.close();  
            try {s.close();} catch (IOException e) {}  
        }  
    }  
}
```



Ejemplo



Otro ejemplo: servidor

```
public class UppercaseServer {
    public static void main(String[] args) throws Exception {
        ServerSocket server = null;
        Scanner scIn = null;
        PrintWriter pwOut = null;
        try {
            // Esperar conexión con cliente
            server = new ServerSocket(5555);
            System.out.println("Esperando conexión");
            Socket s = server.accept();
            System.out.println("Conexión establecida con " + s.getInetAddress());

            // Obtener líneas de texto a partir del cliente,
            // hasta recibir una línea en blanco
            scIn = new Scanner(s.getInputStream());
            String str;
            ArrayList<String> lineas = new ArrayList<String>();
            do {
                str = scIn.nextLine();
                System.out.println("Recibido: " + str);
                lineas.add(str);
            } while (!str.isEmpty());
            ..
        }
    }
}
```

**ArrayList para
almacenar las
cadenas recibidas**

Otro ejemplo: servidor

```
// Recorrer las líneas de texto recibidas, convertirlas a
// mayúsculas, y devolverlas al cliente.
pwOut = new PrintWriter(s.getOutputStream());
for (String linea : lineas) {
    System.out.println("Envío: " + linea.toUpperCase());
    pwOut.println(linea.toUpperCase());
    pwOut.flush();
}
} finally {
    if (pwOut != null) pwOut.close();
    if (scIn != null) scIn.close();
    if (server != null) server.close();
}
}
```



Otro ejemplo: cliente

```
public class SingleClient {  
    public static void main(String[] args) throws Exception {  
        Socket s = new Socket("localhost", 5555);  
        Scanner sc = new Scanner(System.in);  
        PrintWriter pw = new PrintWriter(s.getOutputStream());  
        String str = null;  
        do {  
            System.out.print("CLIENTE >>> ");  
            str = sc.nextLine();  
            pw.println(str);  
            pw.flush();  
        } while (!str.isEmpty());  
  
        Scanner scOut = new Scanner(s.getInputStream());  
        while (scOut.hasNextLine()) {  
            System.out.println(scOut.nextLine());  
        }  
        scOut.close();  
        pw.close();  
        s.close();  
    }  
}
```

**Enviar al servidor
las líneas leídas
de teclado, hasta
encontrar una línea
en blanco**

**Imprimir la
respuesta del
servidor**

Ejemplo

```
manuel@UCMSAFE: ~/Docencia/JSW1/Ejemplos/  
Esperando conexión  
Conexión establecida con /127.0.0.1  
Recibido: Esta es una frase de prueba  
Recibido: Y esto es otra frase  
Recibido: La siguiente línea será en blanco  
Recibido:  
Envío: ESTA ES UNA FRASE DE PRUEBA  
Envío: Y ESTO ES OTRA FRASE  
Envío: LA SIGUIENTE LÍNEA SERÁ EN BLANCO  
Envío:
```

```
BlueJ: Terminal Window - Ej29  
Options  
CLIENTE >>> Esta es una frase de prueba  
CLIENTE >>> Y esto es otra frase  
CLIENTE >>> La siguiente línea será en blanco  
CLIENTE >>>  
ESTA ES UNA FRASE DE PRUEBA  
Y ESTO ES OTRA FRASE  
LA SIGUIENTE LÍNEA SERÁ EN BLANCO
```

¿Y si...?

```
public class SingleClient {  
    public static void main(String[] args) throws Exception {  
        Socket s = new Socket("www.mat.ucm.es", 80);  
        ...  
    }  
}
```

Servidor web



80

Protocolo HTTP

Cliente web



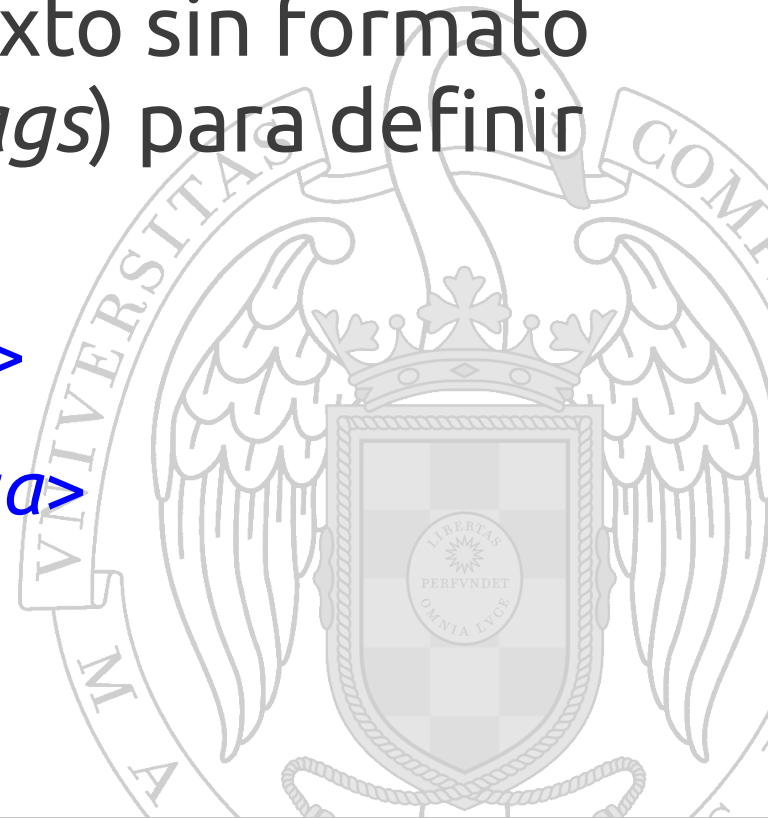
Contenidos

- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*



Lenguaje HTML

- **HTML** – *HyperText Markup Language*.
- Lenguaje para definición de la **estructura** y **contenido** de páginas web.
- Es un lenguaje de marcado: texto sin formato al que se añaden **etiquetas** (*tags*) para definir su estructura.
- Etiquetas de inicio: `<etiqueta>`
- Etiquetas de cierre: `</etiqueta>`




Primer ejemplo en HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Primer ejemplo en HTML</title>
  </head>
  <body>
    Sencillo, no?
    <!-- Esto es un comentario -->
  </body>
</html>
```



Primer ejemplo en HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Primer ejemplo en HTML</title>
  </head>
  <body>
    Sencillo, no?
    <!-- Esto es un comentario -->
  </body>
</html>
```



- La primera línea indica qué tipo de documento HTML se define con esta página.
 - HTML estándar, versión 4 (W3C)

Primer ejemplo en HTML


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Primer ejemplo en HTML</title>
  </head>
  <body>
    Sencillo, no?
    <!-- Esto es un comentario -->
  </body>
</html>
```

} Cabecera

- La cabecera incluye información general sobre la página:
 - Título.
 - Hojas de estilo, etc.

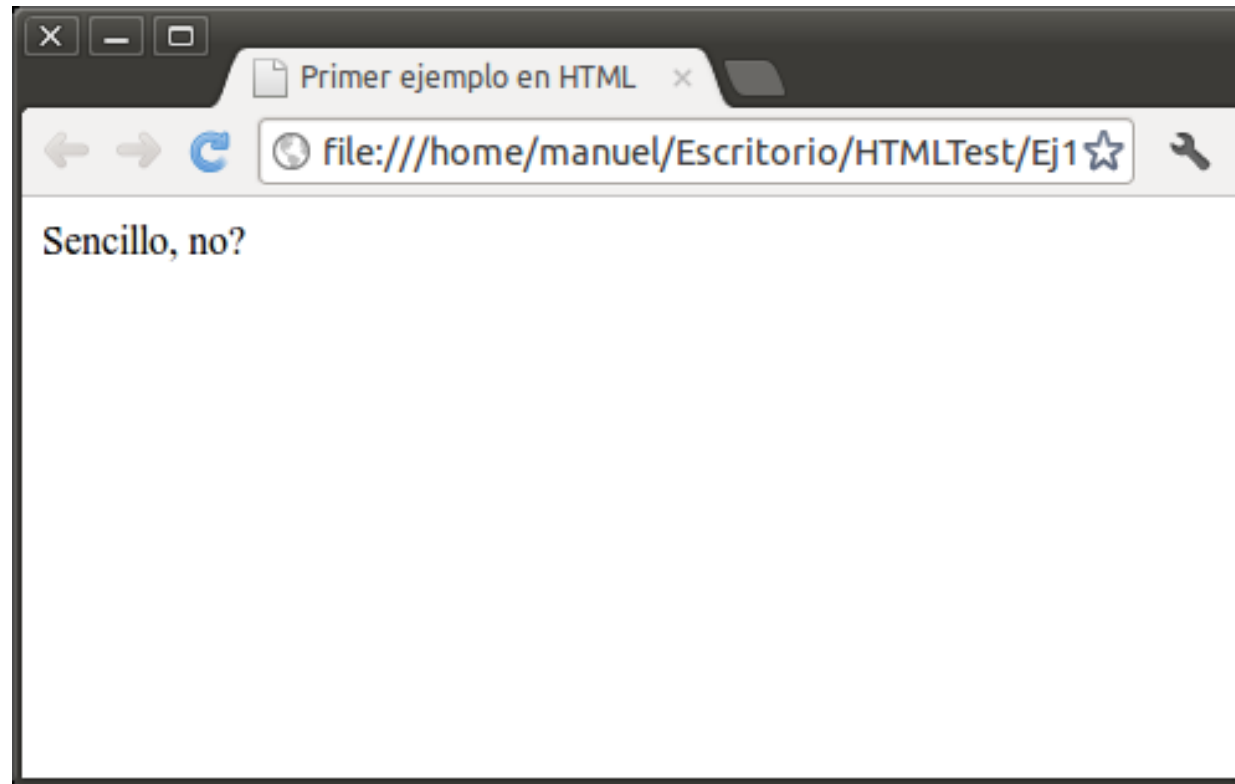
Primer ejemplo en HTML

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Primer ejemplo en HTML</title>
  </head>
  <body>
    Sencillo, no?
    <!-- Esto es un comentario -->
  </body>
</html>
```



- El cuerpo contiene el contenido de la página.

Primer ejemplo en HTML



Estructurar contenido

- Uno o más saltos de línea se consideran como un único espacio en blanco.
- Dos o más espacios en blanco se consideran como uno solo.
- Para estructurar el texto en párrafos, se utiliza la etiqueta <p>.

```
<body>  
  <p>Sencillo, no?</p>  
  <p>Esto es otro parrafo</p>  
</body>
```

Caracteres especiales

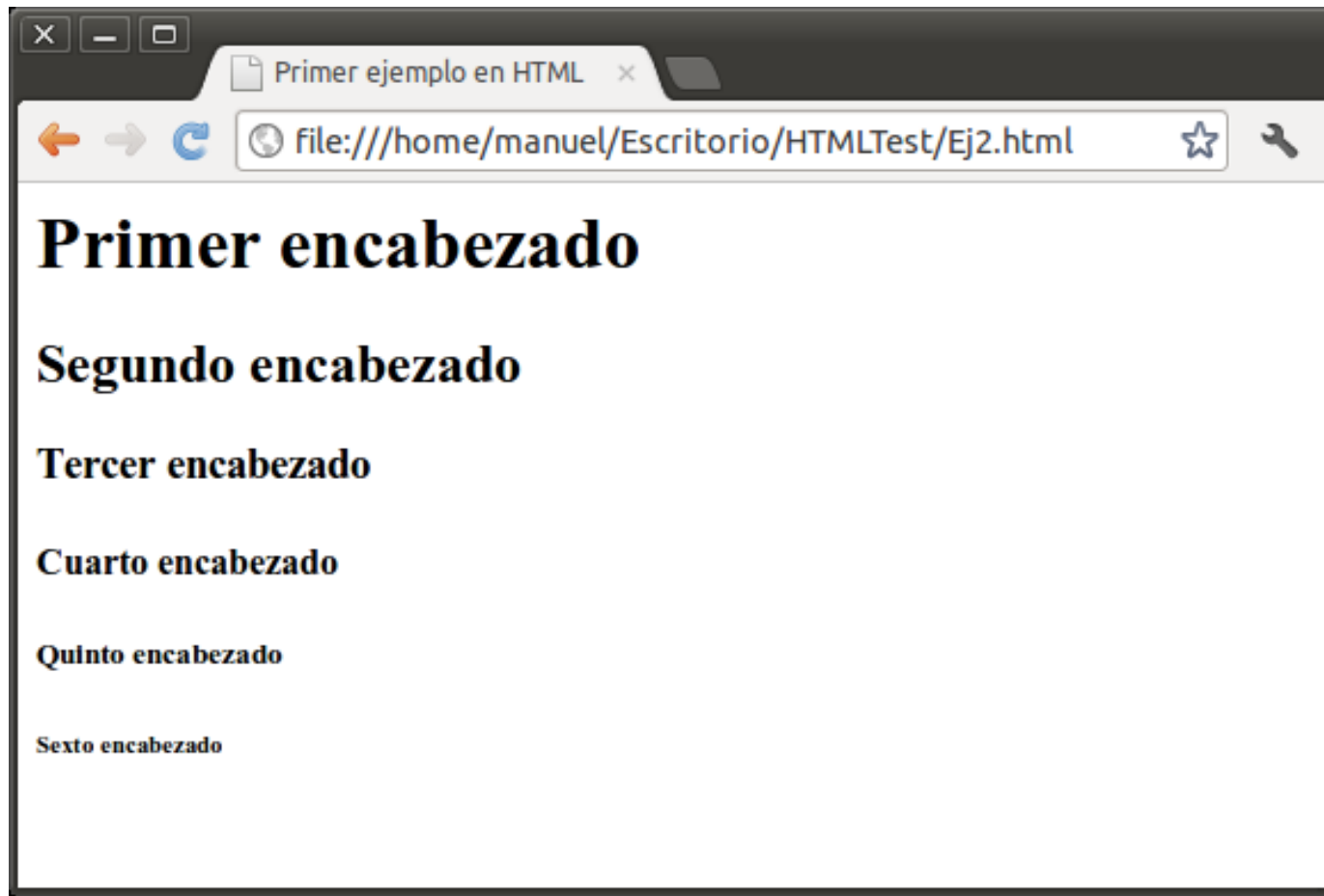
- Para introducir un carácter especial (incluyendo tildes), se utilizan las **entidades** HTML, precedidas por (&) y acabadas en (;)

| | | | | | | | |
|----------|----------|----------|----------|----------|----------|------|------|
| á | é | í | ó | ú | ñ | < | > |
| á | é | í | ó | ú | ñ | < | > |

```
<body>  
  <p>Sencillo, no?</p>  
  <p>Esto es otro p&aacute;rrafo</p>  
</body>
```

Encabezados

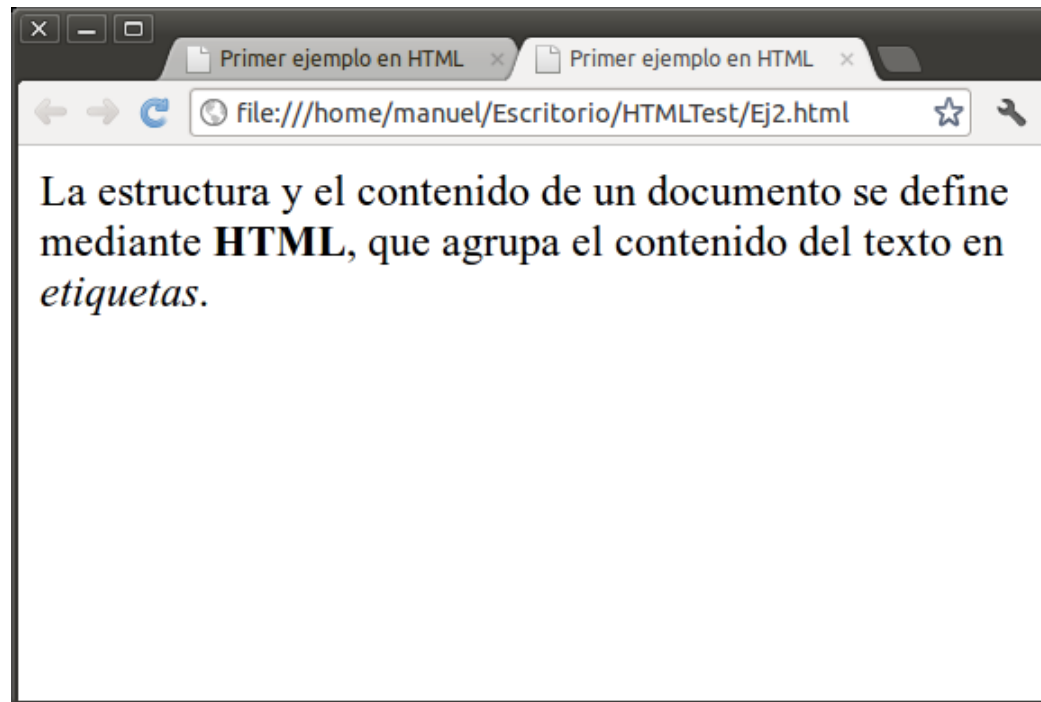
- Desde `<h1> ... </h1>` hasta `<h6> ... </h6>`



Resaltado de fragmentos de texto

- Se realiza mediante `` y ``.

```
<body>  
  <p>La estructura y el contenido de un documento se  
    define mediante <strong>HTML</strong>, que  
    agrupa el contenido del texto en <em>etiquetas</em>.</p>  
</body>
```



Hiperenlaces

- Se insertan mediante la etiqueta `<a>`.

```
<a href="direccion destino">Texto del enlace</a>
```

```
<body>  
  <p>Haga click <a href="http://www.google.com">aquí</a>  
    para ir al buscador</p>  
  <p>Mediante <a href="Ej1.html">este enlace</a>  
    se salta al archivo Ej1.html contenido en el mismo  
    directorio que &acute;ste</p>  
</body>
```


Imágenes

```
</img>
```

```

```

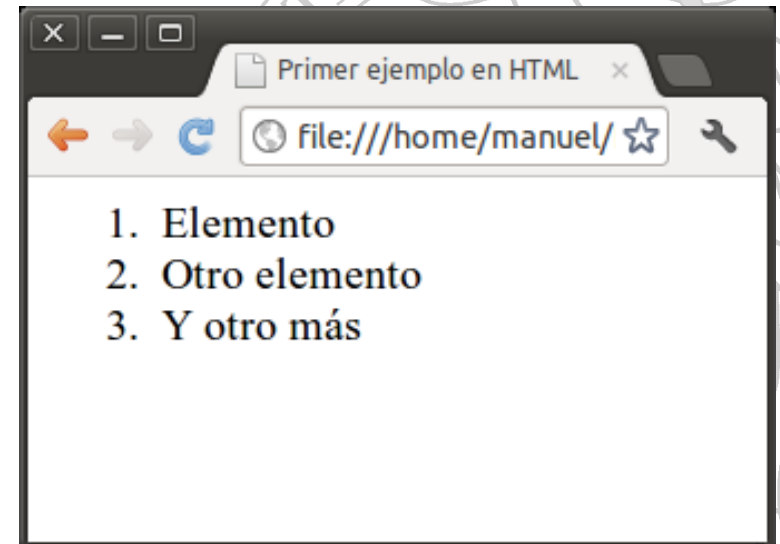
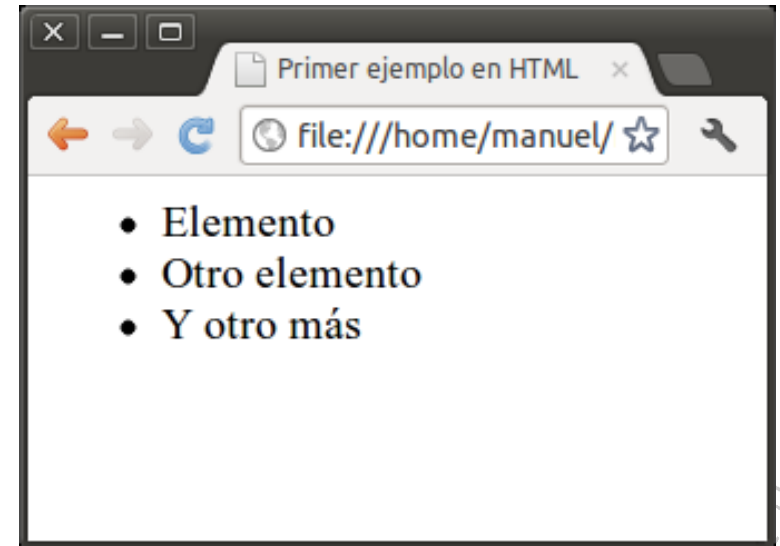
- Se permiten los formatos JPEG, GIF y PNG.

```
<body>  
  <p>Logotipo de la universidad</p>  
  <p></p>  
</body>
```

Listas numeradas y no numeradas

```
<ul>
  <li>Elemento</li>
  <li>Otro elemento</li>
  <li>Y otro m&acute;s</li>
</ul>
```

```
<ol>
  <li>Elemento</li>
  <li>Otro elemento</li>
  <li>Y otro m&acute;s</li>
</ol>
```



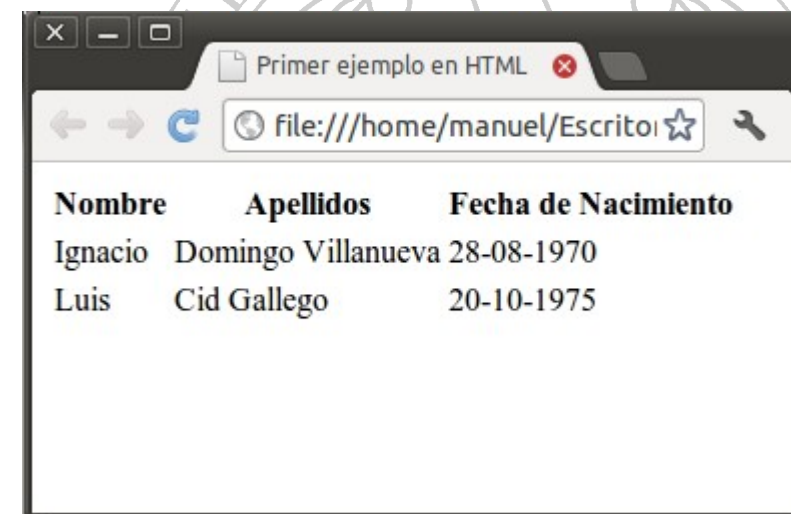
Tablas

```
<table>
  <tr>
    <th>Nombre</th>
    <th>Apellidos</th>
    <th>Fecha de Nacimiento</th>
  </tr>
  <tr>
    <td>Ignacio</td>
    <td>Domingo Villanueva</td>
    <td>28-08-1970</td>
  </tr>
  <tr>
    <td>Luis</td>
    <td>Cid Gallego</td>
    <td>20-10-1975</td>
  </tr>
</table>
```

Fila

Celda de cabecera

Celda



Primer ejemplo en HTML

file:///home/manuel/Escrito

| Nombre | Apellidos | Fecha de Nacimiento |
|---------|--------------------|---------------------|
| Ignacio | Domingo Villanueva | 28-08-1970 |
| Luis | Cid Gallego | 20-10-1975 |

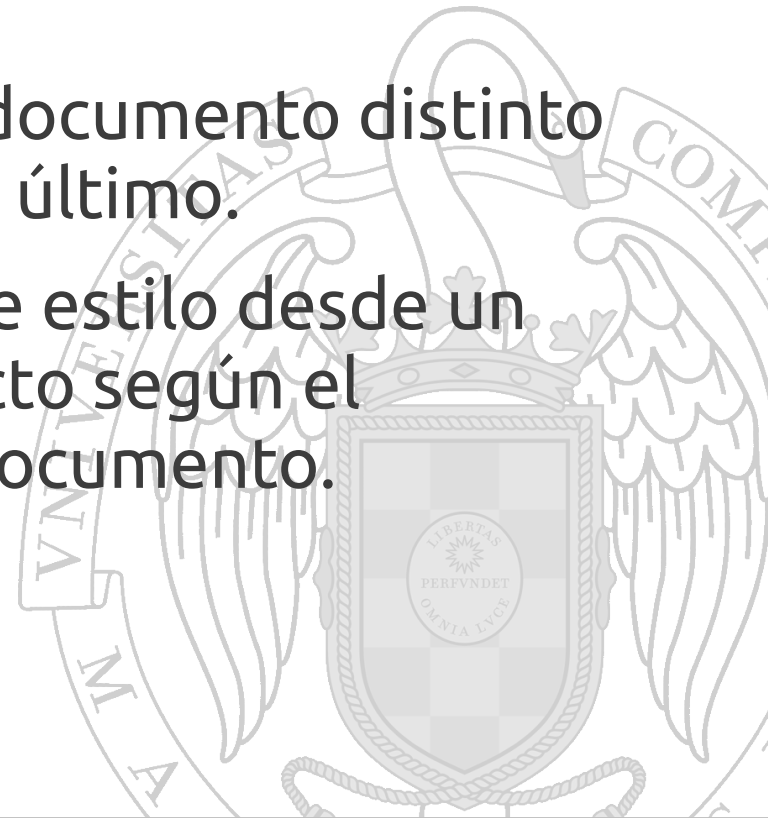
Contenidos

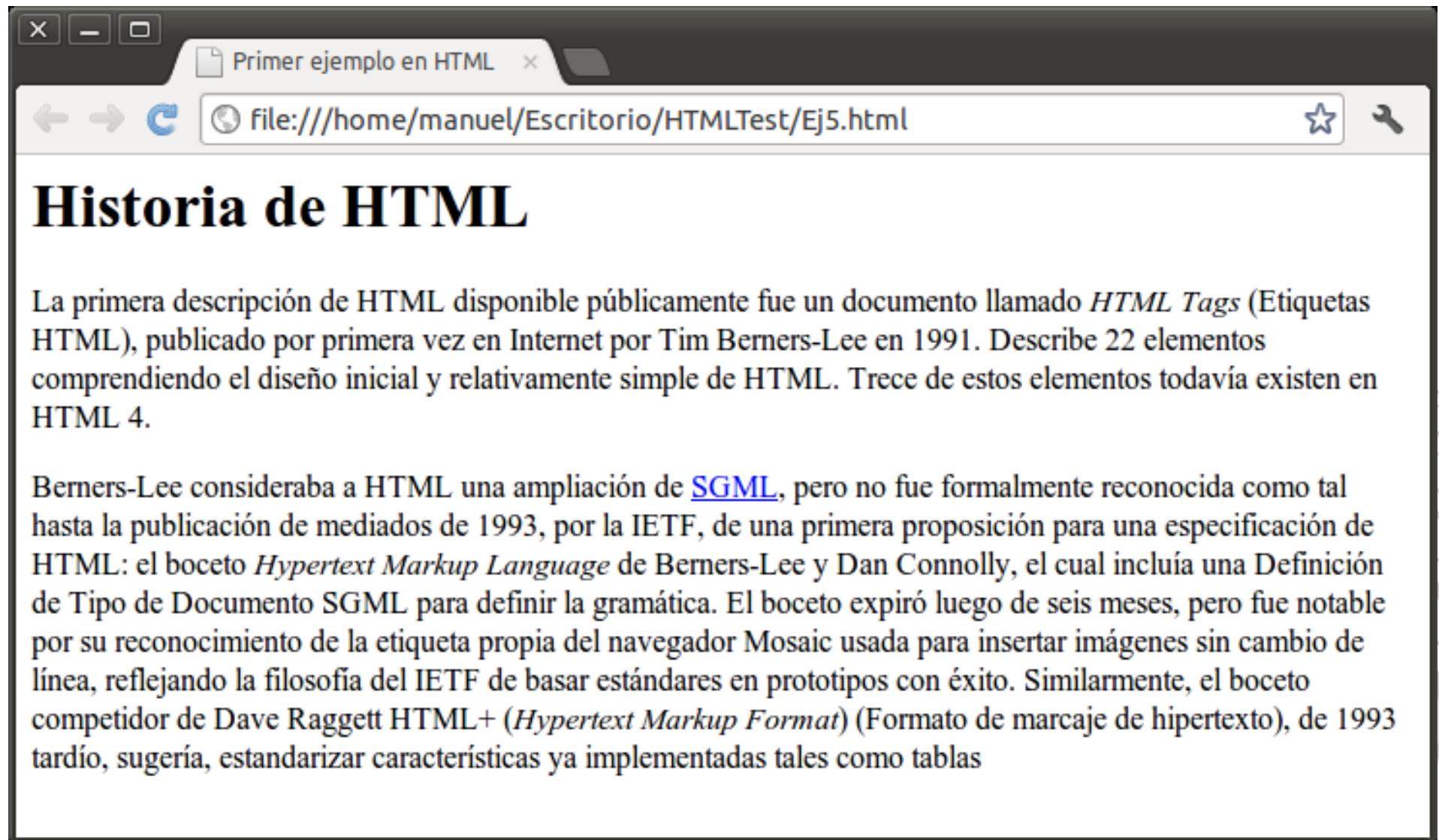
- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*



Hojas de estilo en cascada

- HTML tiene etiquetas para definir el aspecto de los elementos de la página, pero su uso está desaconsejado.
- CSS (*Cascading Style Sheets*) es un lenguaje utilizado para definir la **presentación** de un documento, mediante **hojas de estilo**.
- Las hojas de estilo se definen en un documento distinto al HTML, y se referencian desde este último.
- Se pueden referenciar varias hojas de estilo desde un único HTML, y hacer que tengan efecto según el dispositivo en el que se visualiza el documento.





Fuente: <http://es.wikipedia.org/wiki/HTML>

Ejemplo

- Una hoja de estilo CSS está formada por una serie de bloques de la siguientes forma:

```
etiqueta {  
    atributo1: valor1;  
    atributo2: valor2;  
    ...  
    atributon: valorn;  
}
```

- Lista de atributos:
 - <http://www.w3schools.com/css/default.asp>
 - <http://www.w3.org/Style/CSS/>



Ejemplo

```
body {  
  font-family: sans-serif;  
  background-color: #E0FFE0;  
}  
  
h1 {  
  background-color: #0AFF0A;  
  text-align:right;  
  padding:8px;  
  color: #006000;  
}  
  
em {  
  font-style: normal;  
  color: #00A000;  
}  
  
p {  
  text-align:justify;  
  margin-left:20px;  
  margin-right:20px;  
}  
  
a {  
  color: #A0A000;  
}
```

Tipo de letra
Color de fondo verde claro

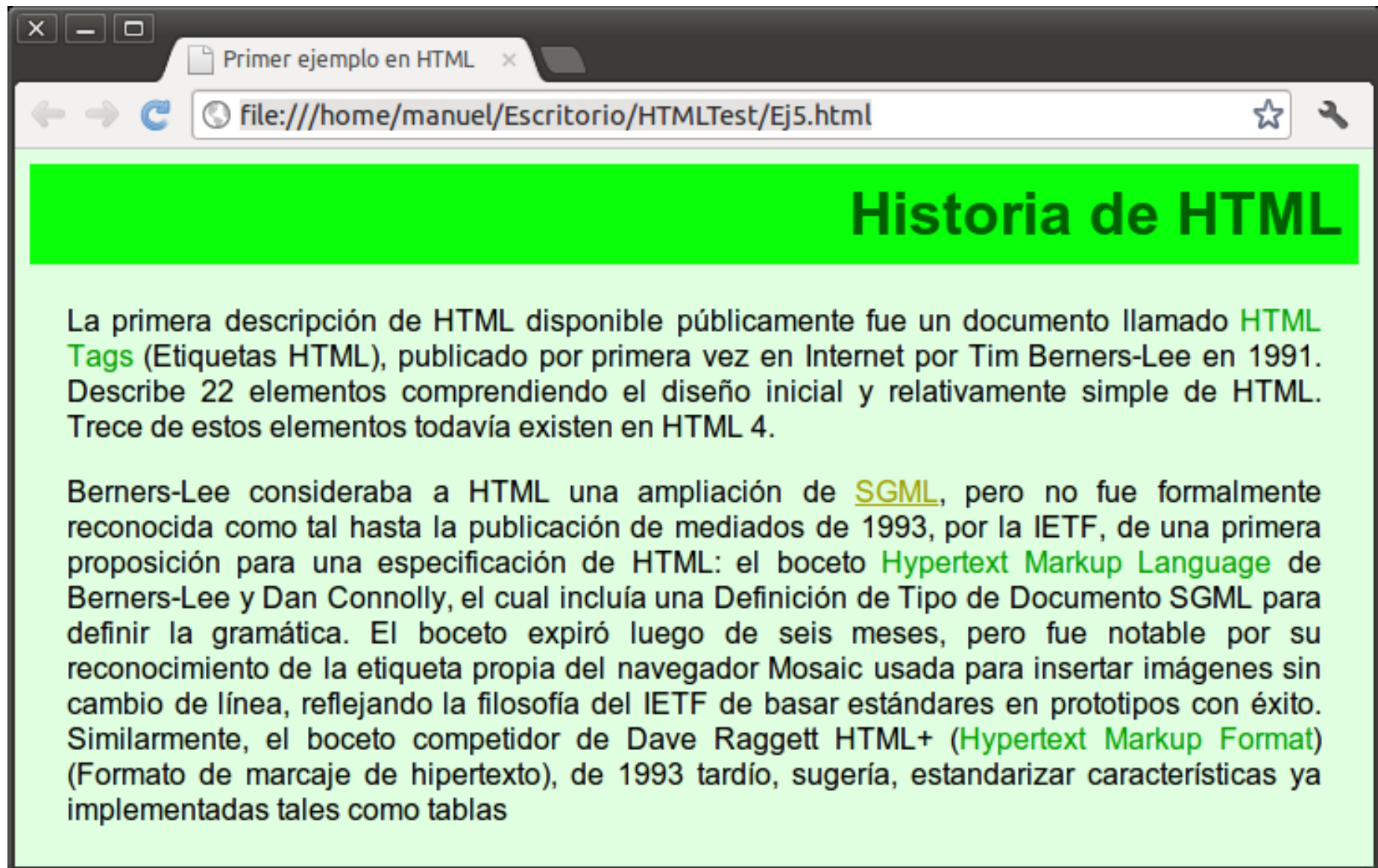
Color de fondo verde
Justificación a la derecha
Márgenes interiores
Color de texto

No utilizar *cursiva*
Color verde

Justificación a márgenes
Márgenes de 20 píxeles

Enlaces de color amarillo

Ejemplo



Enlaces a hojas de estilo

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Ejemplo con hojas de estilo</title>
    <link rel="stylesheet" href="Ej5.css" type="text/css"/>
  </head>
  <body>
    ...
  </body>
</html>
```



Clases de elementos

- Se pueden definir **clases** en los elementos HTML, y aplicar un determinado estilo a los elementos de esa clase.

```
etiqueta.clase {  
    atributo1: valor1;  
    atributo2: valor2;  
    ...  
    atributon: valorn;  
}
```

```
<etiqueta class="clase">
```

Clases de elementos

...

```
p.inicial {  
  border: 2px dashed green;  
  margin-left: 50px;  
  padding: 13px;  
  font-style: italic;  
}
```

*La primera descripción de HTML disponible públicamente fue un documento llamado **HTML Tags** (Etiquetas HTML), publicado por primera vez en Internet por Tim Berners-Lee en 1991. Describe 22 elementos comprendiendo el diseño inicial y relativamente simple de HTML. Trece de estos elementos todavía existen en HTML 4.*

Berners-Lee consideraba a HTML una ampliación de SGML, pero no fue formalmente reconocida como tal hasta la publicación de mediados de 1993, por la IETF, de una primera

Contenidos

- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*



Interactividad en páginas web

- El lenguaje de marcador HTML, por sí solo, no tiene etiquetas para generar contenido **dinámicamente**.
- Aunque proporciona etiquetas para manejo de formularios, el lenguaje no puede procesar la información introducida por el usuario.
 - Se envía al servidor, y éste la interpreta y procesa.
- Para crear páginas web de contenido no estático, hay que complementar HTML con lenguajes y tecnologías adicionales.

Tecnologías Web

Servidor web



80

Protocolo HTTP

Cliente

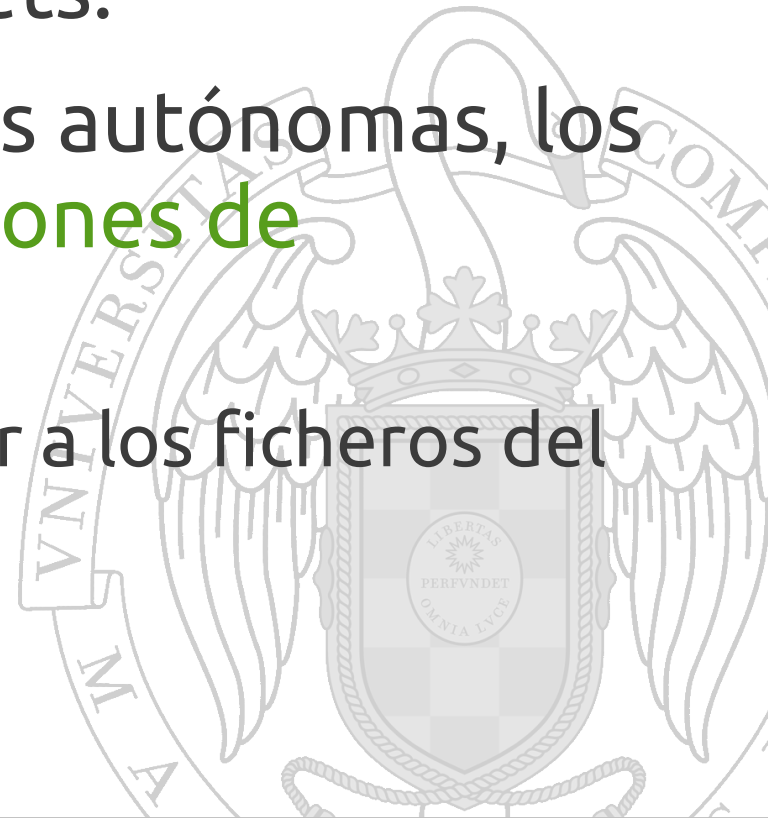


PHP
ASP
Servlets / JSP
...

Javascript
Actionscript / Flash
Applets Java
...

Applets de Java

- Los *applets* son aplicaciones Java integradas en el contenido de una página web.
- Requieren un **añadido** (*plug-in*) en el navegador para ejecutar applets.
- Con respecto a las aplicaciones autónomas, los *applets* tienen ciertas **restricciones de seguridad**.
 - Por ejemplo, no pueden acceder a los ficheros del cliente.



Creación de *applets*

- Para crear un *applet* hay que extender la clase JApplet.
- La clase utiliza unos métodos que pueden ser reescritos en la subclase.
 - `public void init()`
 - `public void start()`
 - `public void stop()`
 - `public void destroy()`



Ejemplo

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

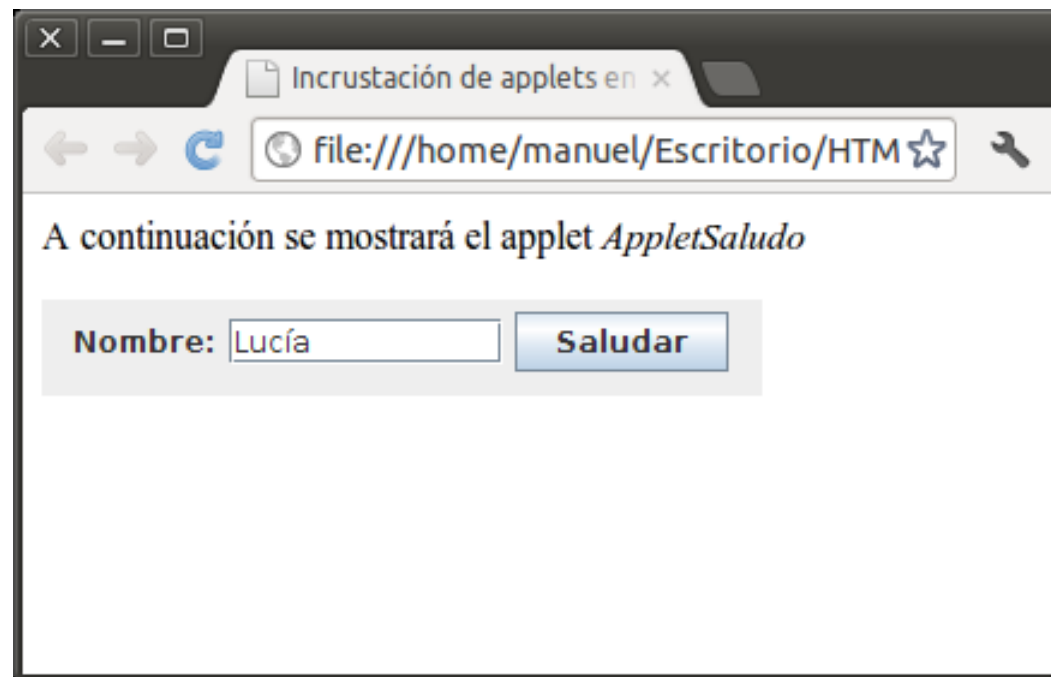
public class AppletSaludo extends JApplet implements ActionListener {
    public JTextField texto;

    public void init() {
        JPanel p = new JPanel();
        texto = new JTextField(10);
        p.setLayout(new FlowLayout());
        p.add(new JLabel("Nombre:"));
        p.add(texto);
        JButton b = new JButton("Saludar");
        b.addActionListener(this);
        p.add(b);
        Container content = getContentPane();
        content.setLayout(new BorderLayout());
        content.add(p);
    }

    public void actionPerformed(ActionEvent e) {
        JOptionPane.showMessageDialog(null, "¡Hola " + texto.getText() + "!");
    }
}
```

- Incrustar applets en páginas web: Etiqueta `<applet>`

```
<p>A continuación se mostrará el applet AppletSaludo</p>  
<applet code="AppletSaludo" width="300" height="40"/>
```



Contenidos

- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*



Servicios web

- Un **servicio web** es una aplicación que proporciona una funcionalidad a través del protocolo HTTP.
- Un ejemplo de ello son las **aplicaciones web** a las que accedemos a través de los **navegadores web**.
- Generalmente, el término servicios web hace referencia al acceso a estas aplicaciones desde un **programa** que no sea un navegador web.
 - ... y que, a veces, no tiene interfaz gráfica.

Servicios web

- Los servicios web fomentan el desarrollo de aplicaciones distribuidas, posiblemente escritas en lenguajes distintos.
 - interoperabilidad
- Los datos se transmiten en lenguaje XML.

Servidor web



80

Protocolo HTTP (leng. XML)

Cliente



Lenguaje XML

- **XML** – *Extensible Markup Language*.
- Generalización/simplificación de HTML.
- Desarrollado para el **intercambio de datos** entre aplicaciones.

```
<persona dni="74612153V">  
  <nombre>Alfonso</nombre>  
  <apellidos>Pérez Fernández</apellidos>  
  <fecha-nacimiento>  
    <dia>26</dia>  
    <mes>3</mes>  
    <año>1974</año>  
  </fecha-nacimiento>  
</persona>
```



Ejemplo de servicio web

- Servicio web que devuelve el día actual.
- Interfaz:

```
package fecha;  
  
public interface ServidorFecha {  
    Fecha getFechaHoy();  
    String getCadenaFecha();  
}
```



Ejemplo de servicio web

- Servicio web que devuelve el día actual.
- Interfaz (*SEI* – *Service Endpoint Interface*)

```
package fecha;  
import javax.jws.*;
```

Paquete donde están definidas
las anotaciones

```
@WebService
```

```
public interface ServidorFecha {
```

```
    @WebMethod Fecha getFechaHoy();
```

```
    @WebMethod String getCadenaFecha();
```

```
}
```

Anotaciones

Ejemplo de servicio web

- Servicio web que devuelve el día actual.
- Implementación (*SIB* – *Service Implementation Bean*):

```
package fecha;
import java.util.*;
import javax.jws.*;

@WebService(endpointInterface = "fecha.ServidorFecha")
public class ServidorFechaImpl implements ServidorFecha {
    public Fecha getFechaHoy() {
        GregorianCalendar gc = new GregorianCalendar();
        gc.setTime(new Date());
        return new Fecha(gc.get(GregorianCalendar.DAY_OF_MONTH),
                        gc.get(GregorianCalendar.MONTH) + 1,
                        gc.get(GregorianCalendar.YEAR));
    }
    public String getCadenaFecha() { return getFechaHoy().toString(); }
}
```

Nombre de la interfaz

La utilidad wsgen

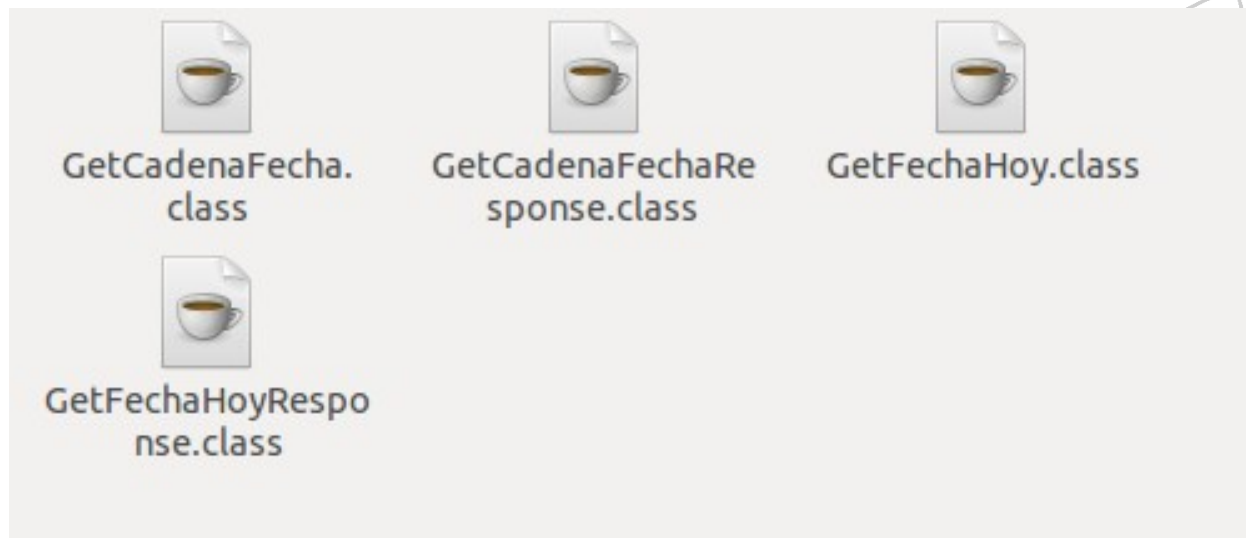
- Genera **artefactos** que sirven para la publicación de la clase como servicio web.
- En el directorio que contiene el paquete fecha, teclear lo siguiente desde una terminal:

```
wsgen -cp . fecha.ServidorFechaImpl
```

Classpath de
Java. Directorio
donde se encuentra
el paquete "raíz"

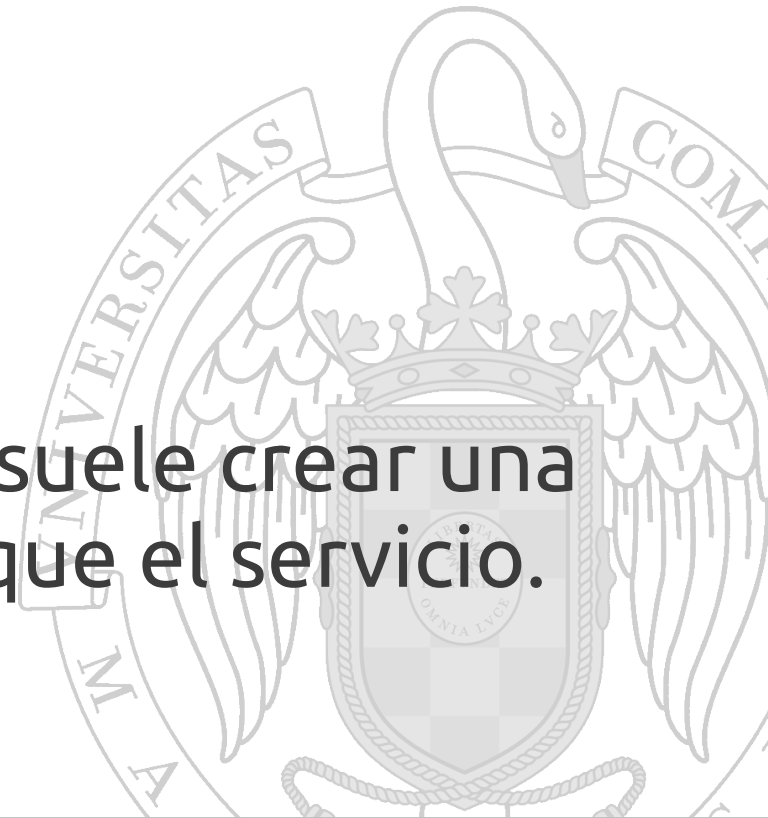
Nombre de la
clase que implementa
el servicio web (SIB)

La utilidad wsgen



Publicar el servicio web

- Existe software específico para la publicación de servicios web:
 - BEA WebLogic
 - JBoss
 - GlassFish
 - WebSphere
 - ...
- En entornos de **desarrollo**, se suele crear una pequeña clase Java que publique el servicio.



Publicar el servicio web

```
package fecha;  
import javax.xml.ws.Endpoint;  
  
class ServicioFechaPublisher {  
    public static void main(String[ ] args) {  
        Endpoint.publish("http://localhost:8888/fecha", new ServidorFechaImpl());  
        System.out.println("Servicio web publicado correctamente.");  
    }  
}
```



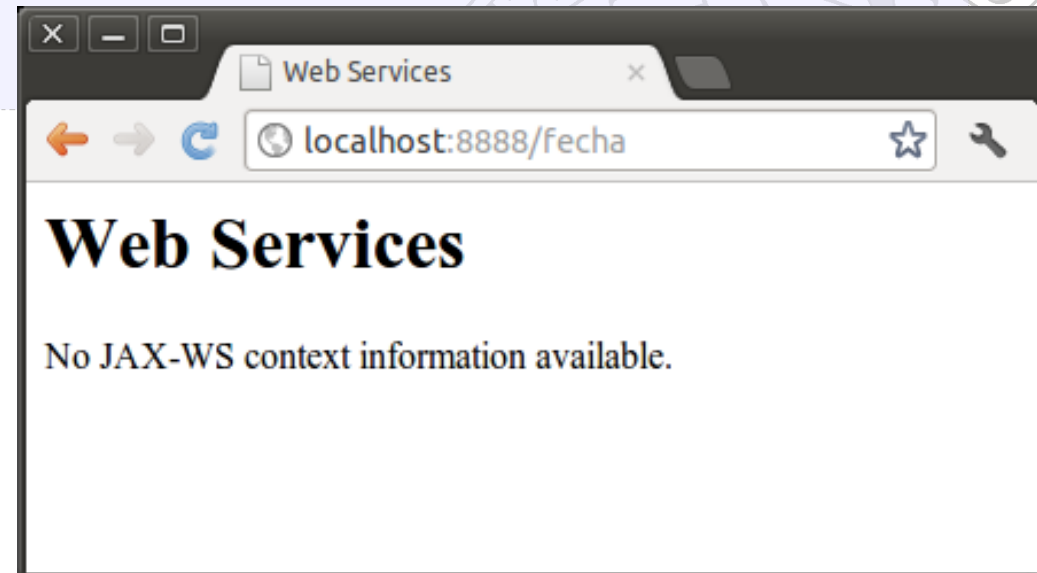
Publicar el servicio web

```
package fecha;  
import javax.xml.ws.Endpoint;
```

```
class ServicioFechaPublisher {  
    public static void main(String[] args) {  
        Endpoint.publish("http://localhost:8888/fecha", new ServidorFechaImpl());  
        System.out.println("Servicio web publicado correctamente.");  
    }  
}
```

URL de publicación

Instancia del SIB



- **WSDL** – *Web Services Description Language*
- Archivo en formato XML que describe la interfaz pública a los servicios web.
- Define las operaciones que ofrece el servicio web, así como el tipo de los parámetros y valor de retorno de cada una de ellas.
- Puede ser accedido a partir de la dirección <http://localhost:8888/fecha?wsdl>



```
<part name="parameters" element="tns:getFechaHoyResponse"/>
</message>
▼<message name="getCadenaFecha">
  <part name="parameters" element="tns:getCadenaFecha"/>
</message>
▼<message name="getCadenaFechaResponse">
  <part name="parameters" element="tns:getCadenaFechaResponse"/>
</message>
▼<portType name="ServidorFecha">
  ▼<operation name="getFechaHoy">
    <input message="tns:getFechaHoy"/>
    <output message="tns:getFechaHoyResponse"/>
  </operation>
  ▼<operation name="getCadenaFecha">
    <input message="tns:getCadenaFecha"/>
    <output message="tns:getCadenaFechaResponse"/>
  </operation>
</portType>
▼<binding name="ServidorFechaImplPortBinding" type="tns:ServidorFecha">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  ▼<operation name="getFechaHoy">
    <soap:operation soapAction=""/>
    ▼<input>
      <soap:body use="literal"/>
    </input>
  </operation>
  ▼<operation name="getCadenaFecha">
    <soap:operation soapAction=""/>
    ▼<input>
      <soap:body use="literal"/>
    </input>
  </operation>
</binding>
```

Cliente del servicio web

- La utilidad **wsimport** genera automáticamente las clases necesarias para desarrollar clientes de un servicio web, a partir de la descripción WSDL del mismo.

Mantener los .java
de las clases
generadas



```
wsimport -p fecha_client -keep http://localhost:8888/fecha?wsdl
```

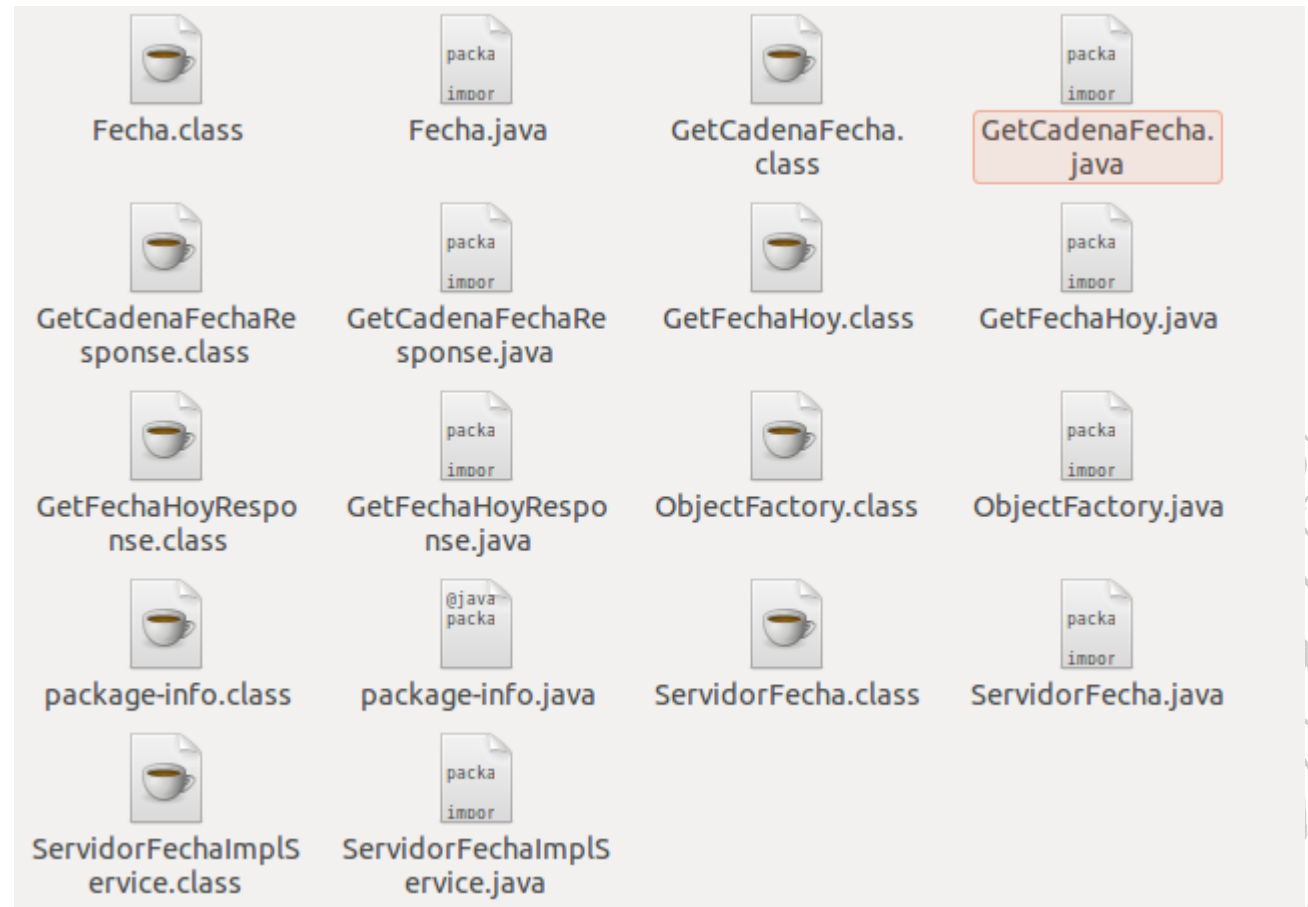
Nombre del paquete
donde guardar
las clases creadas



URL del archivo
WSDL



Cliente del servicio web



Cliente del servicio web

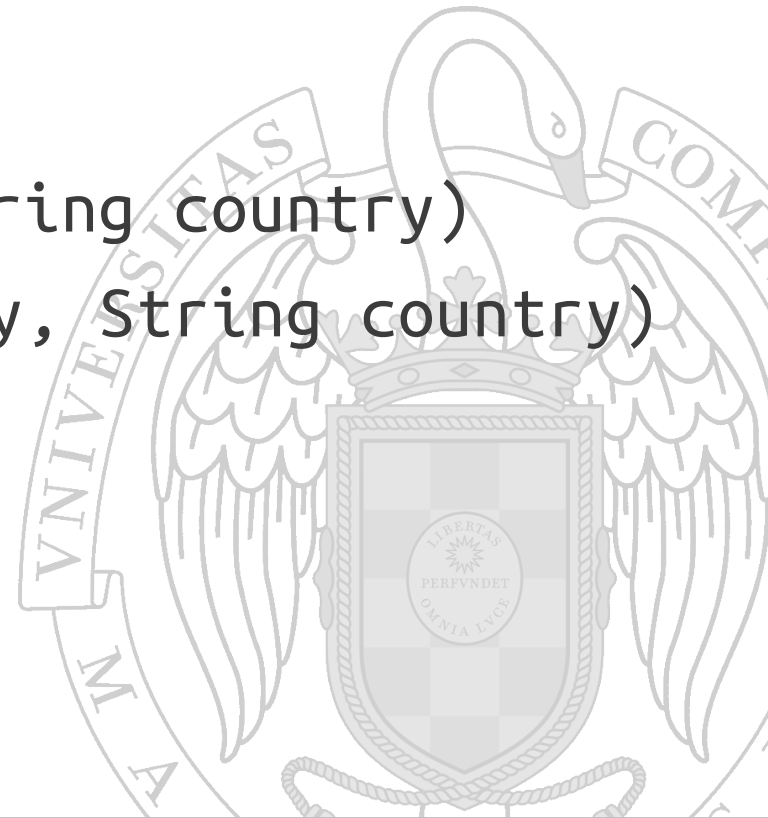
```
import fecha_client.*;

public class ClienteFecha {
    public static void main(String[] args) {
        ServidorFechaImplService servicio = new ServidorFechaImplService();
        ServidorFecha sf = servicio.getServidorFechaImplPort();
        System.out.println(sf.getCadenaFecha());
        Fecha f = sf.getFechaHoy();
        System.out.println("Año: " + f.getAño());
    }
}
```



Servicio de tiempo meteorológico

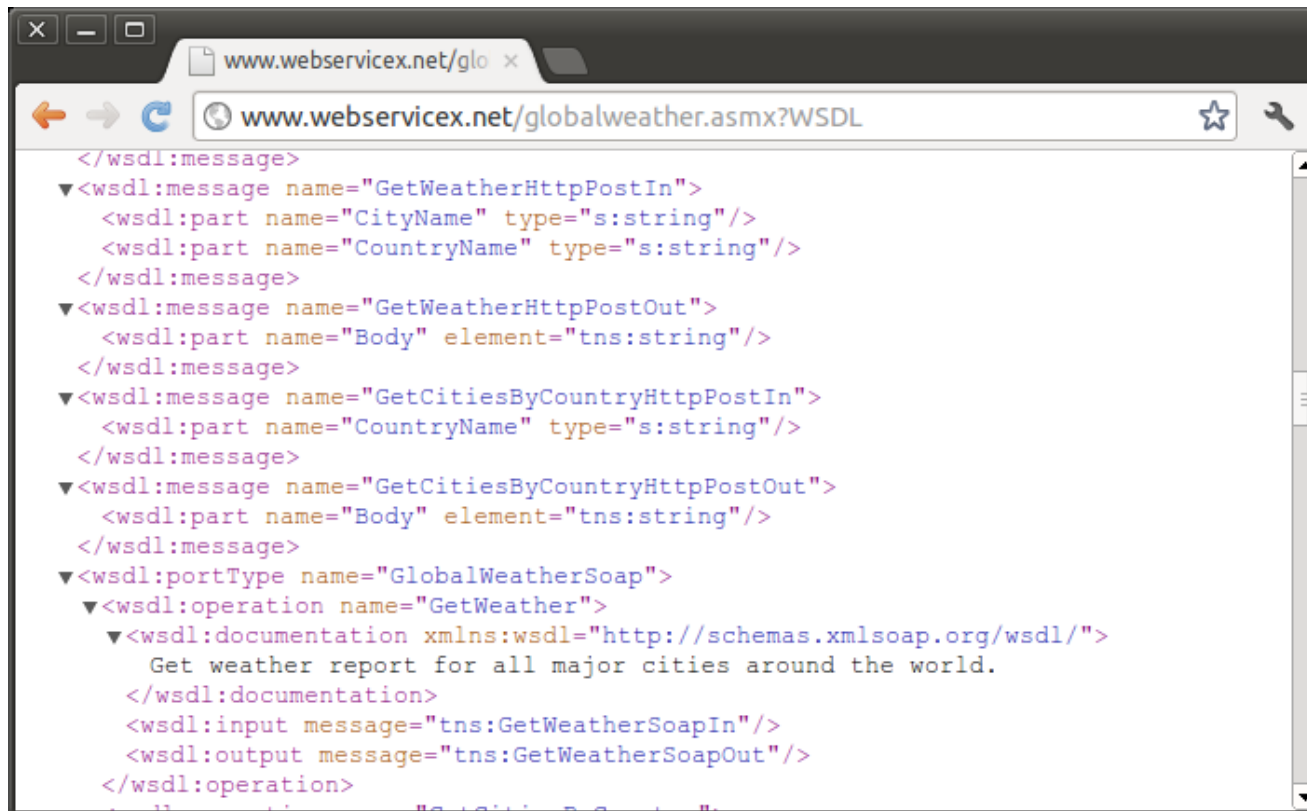
- Información en:
<http://www.webservicex.net/ws/WSDetails.aspx?WSID=56&CATID=12>
- Sirve para obtener la temperatura en una ciudad del mundo determinada.
- Métodos del servicio:
 - String `getCitiesByCountry`(String country)
 - String `getWeather`(String city, String country)



Acceso al WSDL

- Generamos los artefactos Java:

```
wsimport -p tiempo -keep -extension  
http://www.webservicex.net/globalweather.asmx?WSDL
```



The screenshot shows a web browser window with the address bar displaying `www.webservicex.net/globalweather.asmx?WSDL`. The main content area displays the WSDL XML document. The XML is structured as follows:

```
</wsdl:message>  
▼ <wsdl:message name="GetWeatherHttpPostIn">  
  <wsdl:part name="CityName" type="s:string"/>  
  <wsdl:part name="CountryName" type="s:string"/>  
</wsdl:message>  
▼ <wsdl:message name="GetWeatherHttpPostOut">  
  <wsdl:part name="Body" element="tns:string"/>  
</wsdl:message>  
▼ <wsdl:message name="GetCitiesByCountryHttpPostIn">  
  <wsdl:part name="CountryName" type="s:string"/>  
</wsdl:message>  
▼ <wsdl:message name="GetCitiesByCountryHttpPostOut">  
  <wsdl:part name="Body" element="tns:string"/>  
</wsdl:message>  
▼ <wsdl:portType name="GlobalWeatherSoap">  
  ▼ <wsdl:operation name="GetWeather">  
    ▼ <wsdl:documentation xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">  
      Get weather report for all major cities around the world.  
    </wsdl:documentation>  
    <wsdl:input message="tns:GetWeatherSoapIn"/>  
    <wsdl:output message="tns:GetWeatherSoapOut"/>  
  </wsdl:operation>
```

Desarrollo del cliente

```
import tiempo.*;

public class TiempoClient {
    public static void main(String[] args) {
        GlobalWeather servicio = new GlobalWeather();
        GlobalWeatherSoap gw = servicio.getGlobalWeatherSoap();
        String cad = gw.getCitiesByCountry("Spain");
        System.out.println(cad);
    }
}
```



Desarrollo del cliente



A screenshot of a BlueJ Terminal Window titled "BlueJ: Terminal Window - Ej35". The window contains XML data under the "Options" tab. The XML structure consists of a root element `</NewDataSet>` containing five `<Table>` elements. Each `<Table>` element contains two sub-elements: `<Country>` and `<City>`. The data is as follows:

| Country | City |
|---------|-----------------------|
| Spain | Vitoria |
| Spain | Vigo / Peinador |
| Spain | Santander / Parayas |
| Spain | Zaragoza / Aeropuerto |
| Spain | Sevilla / San Pablo |

The terminal window has a scrollbar on the right side, and the text is displayed in a monospaced font.

Formato XML

Intérpretes XML

- ¿Cómo extraer la información deseada a partir de un documento XML?
- Dos posibilidades:
 - 1) Extracción mediante **expresiones regulares**.
 - 2) Extracción mediante las librerías de Java para el manejo de XML: **JAXP**

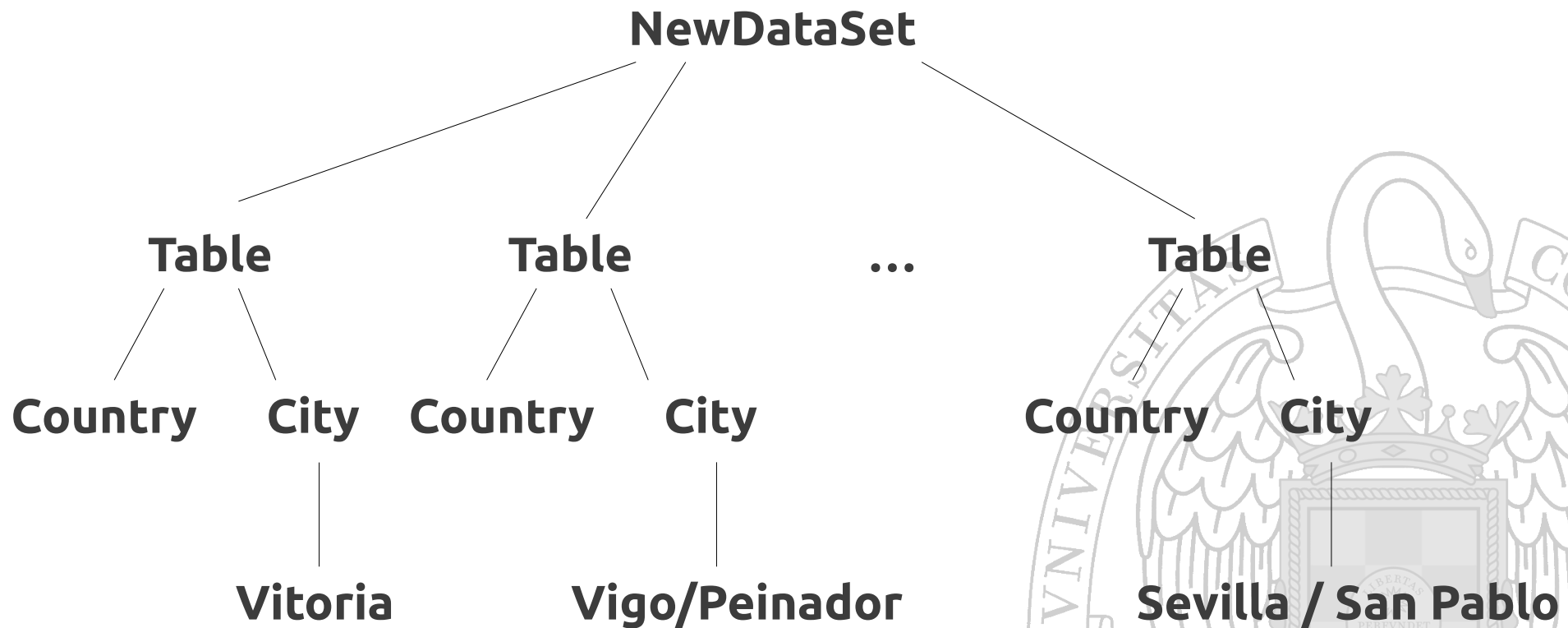


Método 1 – Expresiones regulares

```
import tiempo.*;
import java.util.regex.*;

public class TiempoClientRegEx {
    public static void main(String[] args) {
        GlobalWeather servicio = new GlobalWeather();
        GlobalWeatherSoap gw = servicio.getGlobalWeatherSoap();
        String ciudades = gw.getCitiesByCountry("Spain");
        Pattern pat = Pattern.compile("<City>(.*?)</City>");
        Matcher m = pat.matcher(ciudades);
        while(m.find()) {
            System.out.println(m.group(1));
        }
    }
}
```

Método 2 - JAXP



Método 2 – JAXP

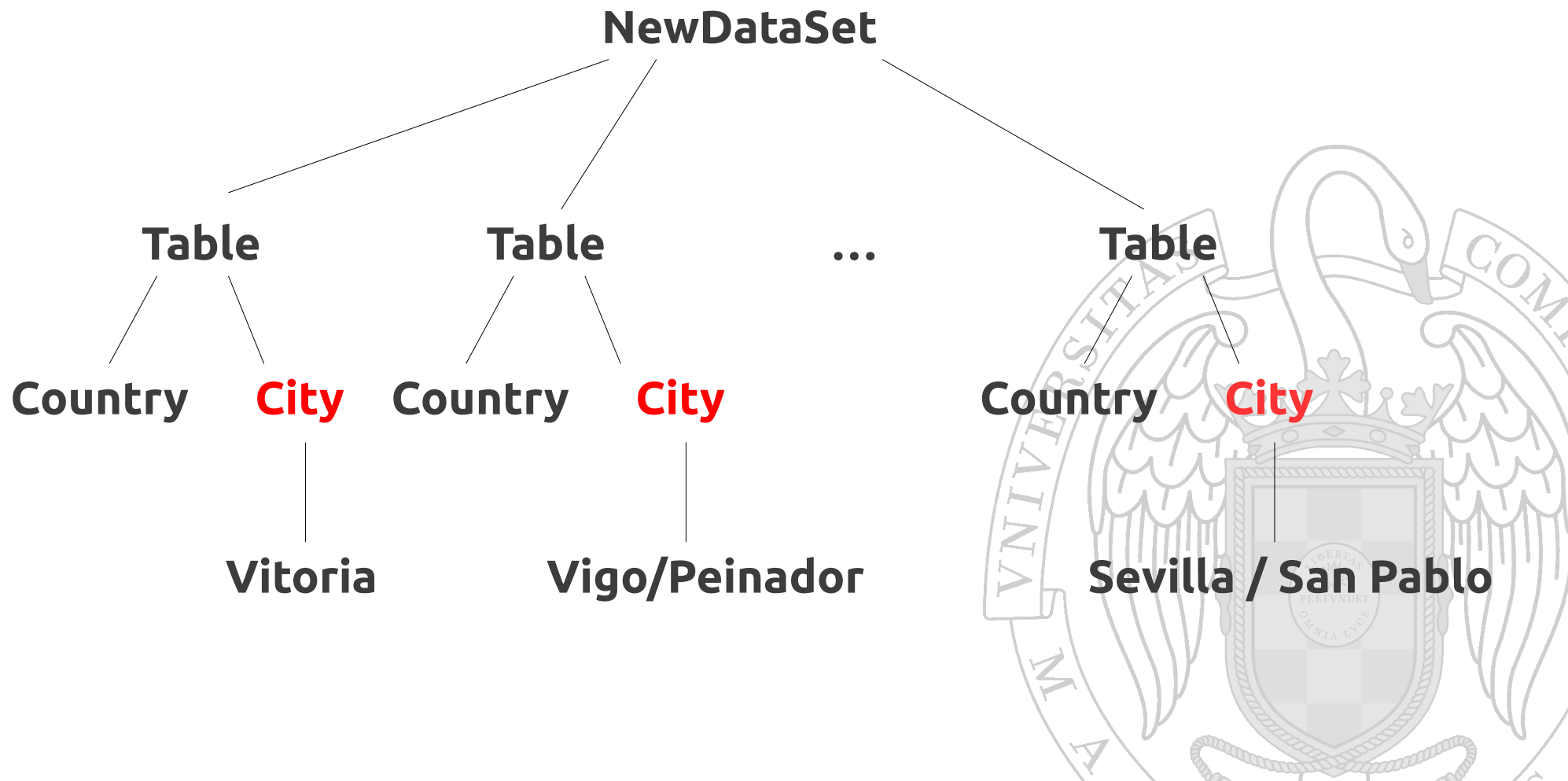
```
import tiempo.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.io.*;

public class TiempoClientJAXP {
    public static void main(String[] args) throws Exception {
        GlobalWeather servicio = new GlobalWeather();
        GlobalWeatherSoap gw = servicio.getGlobalWeatherSoap();
        String ciudades = gw.getCitiesByCountry("Spain");
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
        DocumentBuilder db = dbf.newDocumentBuilder();

        Document documento = db.parse(new ByteArrayInputStream(ciudades.getBytes()));
        NodeList elems = documento.getElementsByTagName("City");
        for (int i = 0; i < elems.getLength(); i++) {
            System.out.println(elems.item(i).getFirstChild().getNodeValue());
        }
    }
}
```

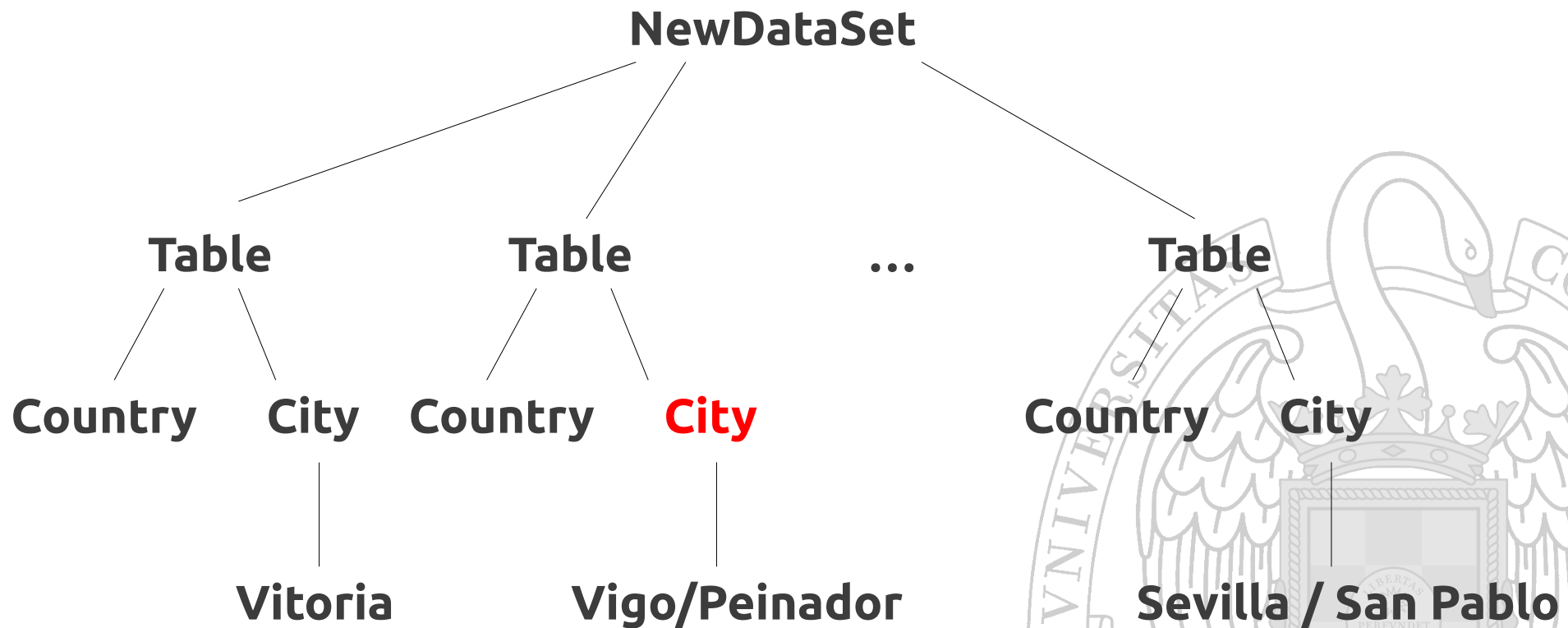
Método 2 - JAXP

```
documento.getElementsByTagName("City");
```



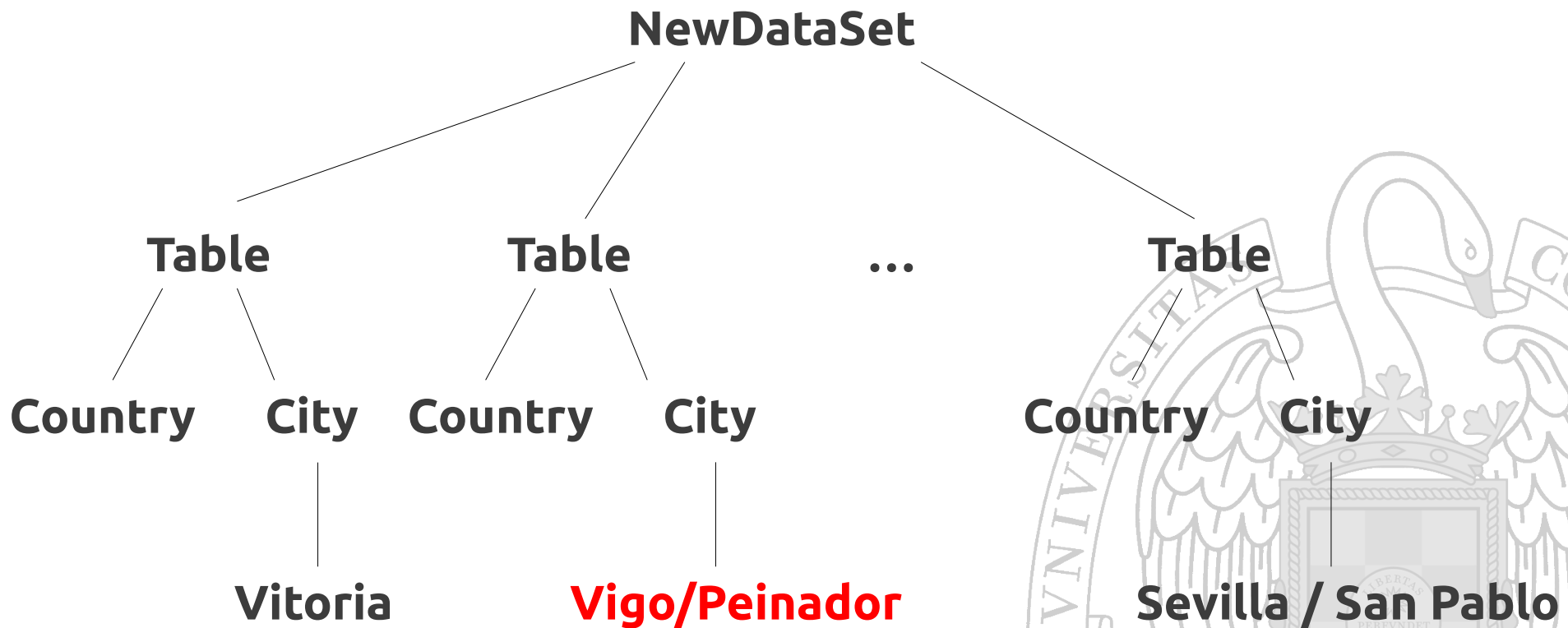
Método 2 - JAXP

`elems.item(1)`



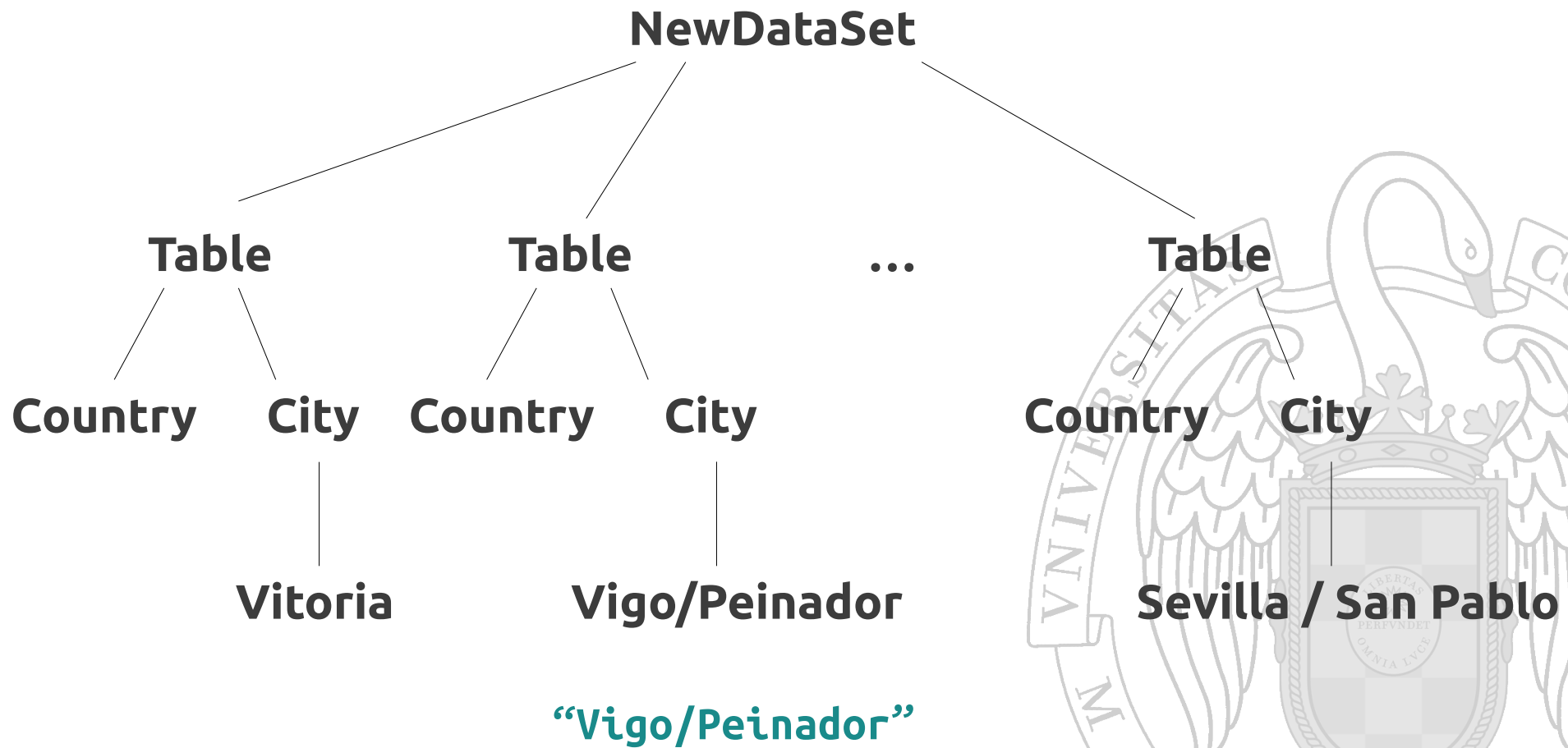
Método 2 - JAXP

```
elems.item(1).firstChild()
```



Método 2 - JAXP

```
elems.item(1).firstChild().getNodeValue()
```



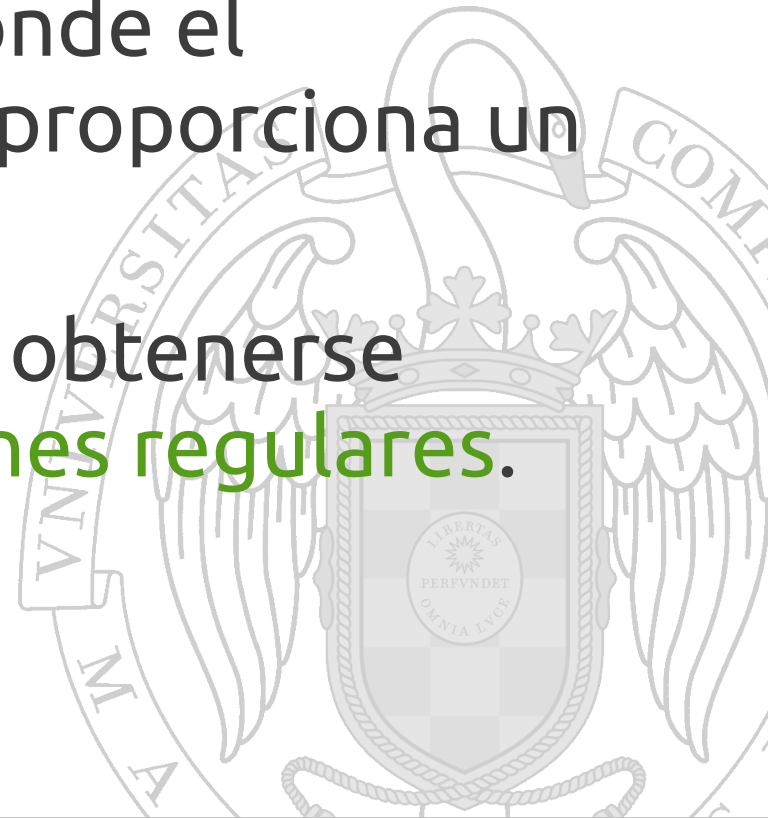
Contenidos

- Clientes, servidores y puertos.
- Comunicación mediante *sockets*.
- Tecnologías web
 - HTML
 - CSS
 - *Applets*
- Servicios web
 - SOAP: intercambio de datos
 - *Web scraping*

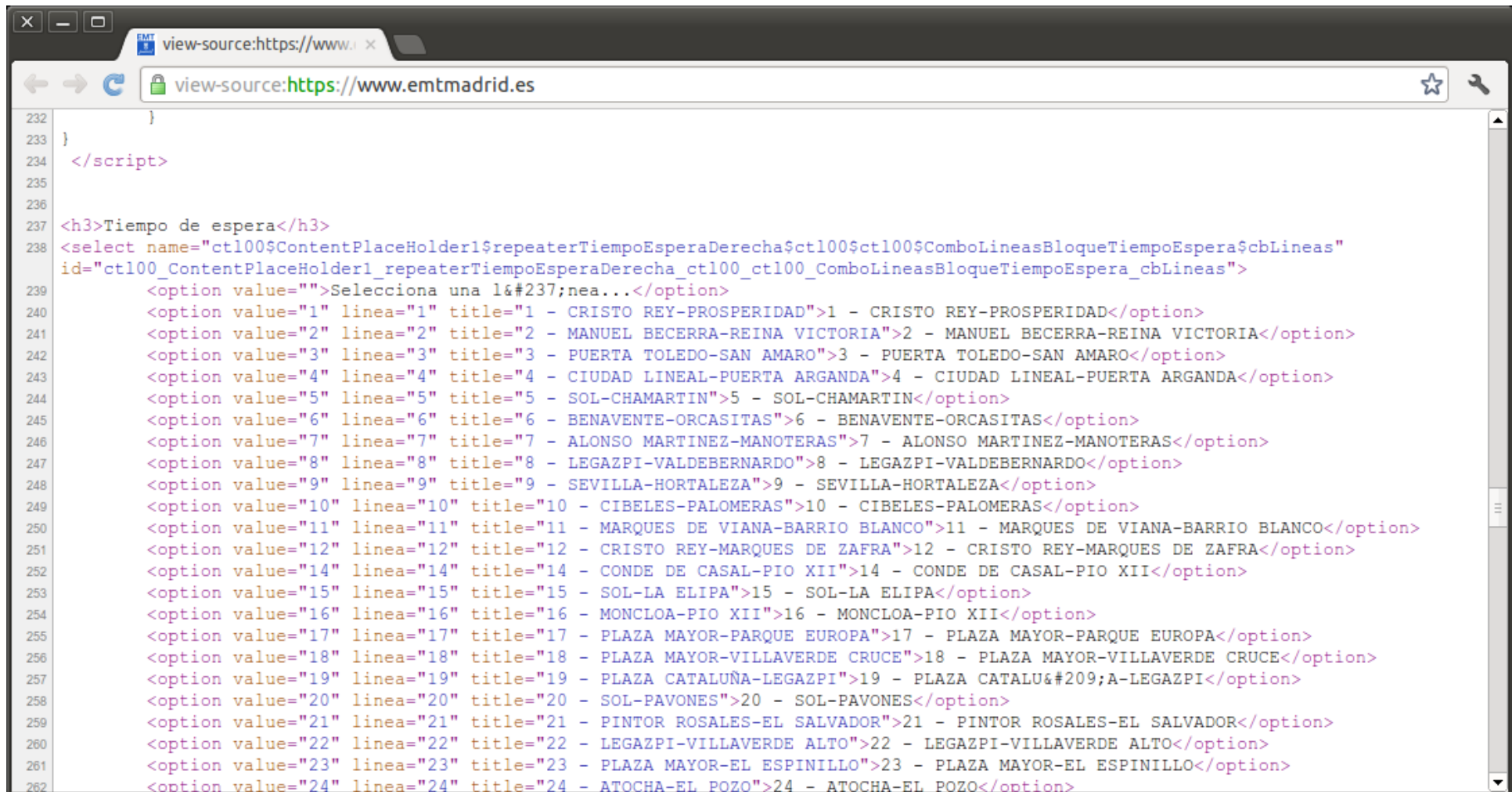


Web scraping

- *Poor man's web service.*
- **Extracción automática** de datos a partir de una página web en formato HTML.
- Se utiliza en aquellos casos donde el proveedor de información no proporciona un servicio de acceso a la misma.
- La información deseada suele obtenerse mediante ajustes de **expresiones regulares**.



- Acceso a líneas de EMT



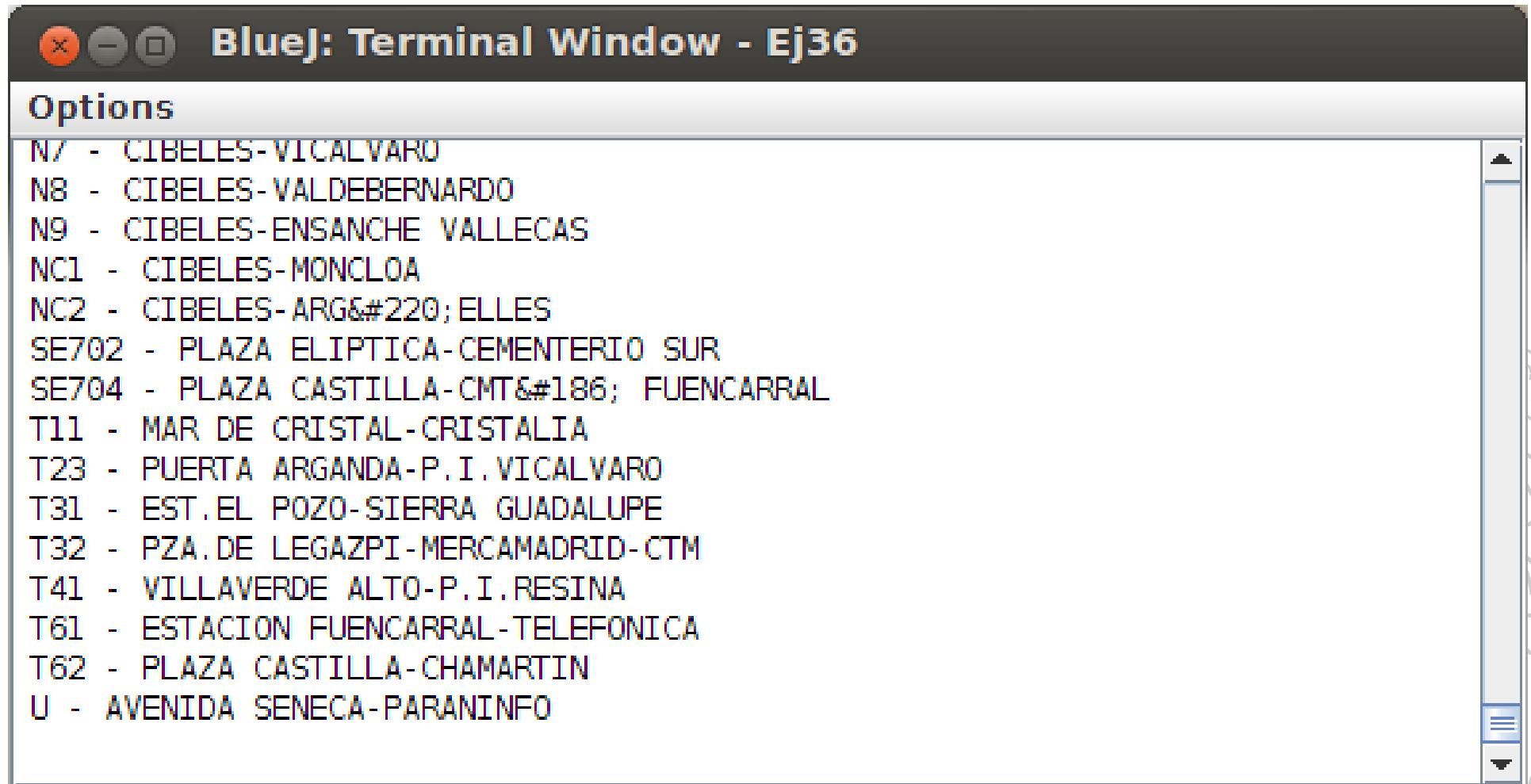
```
232 }
233 }
234 </script>
235
236
237 <h3>Tiempo de espera</h3>
238 <select name="ctl00$ContentPlaceholder1$repeaterTiempoEsperaDerecha$ctl00$ctl00$ComboLineasBloqueTiempoEspera$cbLineas"
239 id="ctl00_ContentPlaceholder1_repeaterTiempoEsperaDerecha_ctl00_ctl00_ComboLineasBloqueTiempoEspera_cbLineas">
240 <option value="">Selecciona una l&#237;nea...</option>
241 <option value="1" linea="1" title="1 - CRISTO REY-PROSPERIDAD">1 - CRISTO REY-PROSPERIDAD</option>
242 <option value="2" linea="2" title="2 - MANUEL BECERRA-REINA VICTORIA">2 - MANUEL BECERRA-REINA VICTORIA</option>
243 <option value="3" linea="3" title="3 - PUERTA TOLEDO-SAN AMARO">3 - PUERTA TOLEDO-SAN AMARO</option>
244 <option value="4" linea="4" title="4 - CIUDAD LINEAL-PUERTA ARGANDA">4 - CIUDAD LINEAL-PUERTA ARGANDA</option>
245 <option value="5" linea="5" title="5 - SOL-CHAMARTIN">5 - SOL-CHAMARTIN</option>
246 <option value="6" linea="6" title="6 - BENAVENTE-ORCASITAS">6 - BENAVENTE-ORCASITAS</option>
247 <option value="7" linea="7" title="7 - ALONSO MARTINEZ-MANOTERAS">7 - ALONSO MARTINEZ-MANOTERAS</option>
248 <option value="8" linea="8" title="8 - LEGAZPI-VALDEBERNARDO">8 - LEGAZPI-VALDEBERNARDO</option>
249 <option value="9" linea="9" title="9 - SEVILLA-HORTALEZA">9 - SEVILLA-HORTALEZA</option>
250 <option value="10" linea="10" title="10 - CIBELES-PALOMERAS">10 - CIBELES-PALOMERAS</option>
251 <option value="11" linea="11" title="11 - MARQUES DE VIANA-BARRIO BLANCO">11 - MARQUES DE VIANA-BARRIO BLANCO</option>
252 <option value="12" linea="12" title="12 - CRISTO REY-MARQUES DE ZAFRA">12 - CRISTO REY-MARQUES DE ZAFRA</option>
253 <option value="14" linea="14" title="14 - CONDE DE CASAL-PIO XII">14 - CONDE DE CASAL-PIO XII</option>
254 <option value="15" linea="15" title="15 - SOL-LA ELIPA">15 - SOL-LA ELIPA</option>
255 <option value="16" linea="16" title="16 - MONCLOA-PIO XII">16 - MONCLOA-PIO XII</option>
256 <option value="17" linea="17" title="17 - PLAZA MAYOR-PARQUE EUROPA">17 - PLAZA MAYOR-PARQUE EUROPA</option>
257 <option value="18" linea="18" title="18 - PLAZA MAYOR-VILLASVERDE CRUCE">18 - PLAZA MAYOR-VILLASVERDE CRUCE</option>
258 <option value="19" linea="19" title="19 - PLAZA CATALUÑA-LEGAZPI">19 - PLAZA CATALUÑA-LEGAZPI</option>
259 <option value="20" linea="20" title="20 - SOL-PAVONES">20 - SOL-PAVONES</option>
260 <option value="21" linea="21" title="21 - PINTOR ROSALES-EL SALVADOR">21 - PINTOR ROSALES-EL SALVADOR</option>
261 <option value="22" linea="22" title="22 - LEGAZPI-VILLASVERDE ALTO">22 - LEGAZPI-VILLASVERDE ALTO</option>
262 <option value="23" linea="23" title="23 - PLAZA MAYOR-EL ESPINILLO">23 - PLAZA MAYOR-EL ESPINILLO</option>
263 <option value="24" linea="24" title="24 - ATOCHA-EL POZO">24 - ATOCHA-EL POZO</option>
```

- Acceso a líneas de EMT

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

public class LineasEMT {
    public static void main(String[] args) throws Exception {
        URL url = new URL("https://www.emtmadrid.es");
        URLConnection con = url.openConnection();
        Scanner sc = new Scanner(con.getInputStream());
        Pattern p = Pattern.compile("<option value=\"[^\"]*\" linea=\"[^\"]*\" +\n" +
                                     "title=\"[^\"]*\">(.*?)</option>");
        while(sc.hasNextLine()) {
            String linea = sc.nextLine();
            Matcher m = p.matcher(linea);
            if (m.find()) { System.out.println(m.group(1)); }
        }
    }
}
```

Ejemplo



Referencias

- P. Deitel, H. Deitel
Java. How to Program (9th Edition)
Cap. 23, 27
- M. Kalin
Java web services: up and running
O'Reilly Media
- <http://www.w3schools.com/html/>
- <http://www.w3schools.com/css/>

